



PAC Learning of Deterministic One-Clock Timed Automata*

Wei Shen¹, Jie An¹ , Bohua Zhan^{2,3} , Miaomiao Zhang¹ ,
Bai Xue^{2,3} , and Naijun Zhan^{2,3} 

¹ School of Software Engineering, Tongji University, Shanghai, China
{weishen, 1510796, miaomiao}@tongji.edu.cn

² State Key Lab. of Computer Science, Institute of Software, CAS, Beijing, China
{bzhan, xuebai, znj}@ios.ac.cn

³ University of Chinese Academy of Sciences, Beijing, China

Abstract. We study the problem of learning deterministic one-clock timed automata in the framework of PAC (probably approximately correct) learning. The use of PAC learning relaxes the assumption of having a teacher that can answer equivalence queries exactly, replacing it with approximate answers from testing on a set of samples. The framework provides correctness guarantees in terms of error and confidence parameters. We further discuss several improvements to the basic PAC algorithm. This includes a special sampling method, the use of a comparator to reduce the number of equivalence queries, and the use of counterexample minimization. We implemented a prototype for our learning algorithm, and conducted experiments using the TCP protocol as well as a number of randomly generated automata. The results demonstrate the effectiveness of our approach, as well as the importance of the various improvements for learning complex models.

Keywords: Timed automata · Active learning · Automata learning · Probably approximately correct learning.

1 Introduction

In recent years, model learning [30] is emerging as a highly effective technique for learning black-box systems from system observations. It generally is divided into two categories: *active learning* and *passive learning*. Active learning under the L^* framework [7] can be viewed as an interaction between a *learner* and a *teacher*, where the learner asks *membership queries* and *equivalence queries* to a teacher who holds oracles to answer these queries. This is distinguished from passive learning, i.e., generating a model consistent with a given data set. Recently, active learning has been extended to many formal models.

In previous work [5], we introduced active learning algorithms for deterministic one-clock timed automata (DOTA). There are two variants of the algorithm. The first

* This work has been partially funded by NSFC under grant No. 61972284, No. 61625206, No. 61732001 and No. 61836005, and by the CAS Pioneer Hundred Talents Program under grant No. Y9RC585036.

variant is based on the assumption of a *smart teacher* who can provide clock-reset information along queries. The idea then is to use the reset information to convert the learning problem to that of learning the corresponding *reset-logical-timed language*, which can be solved following the approaches to learning symbolic automata [12,20]. The second variant assumes only a *normal teacher* who does not provide reset information. The learner then needs to *guess* reset information on transitions discovered in the observation table. Due to these guesses, the second variant has exponential complexity in the size of the learned automata, while the first variant has polynomial complexity.

In both variants, we assumed that the equivalence queries can be answered exactly. In the experiments, this is implemented using a decision procedure for language inclusion. This kind of equivalence queries is difficult to realize in practical applications, as it essentially require a teacher to have the power to compare two systems exactly. This problem is addressed in [1] using conformance testing. Another way is to follow the PAC (probably approximately correct) framework, which is studied in existing work [7,10,21] for other kinds of models. Under this framework, for a given *error* ϵ and *confidence* δ , we can determine the number of test cases needed for each equivalence query. If the current hypothesis passes all test cases, then with probability $1 - \delta$, it agrees with the target model on at least $1 - \epsilon$ proportion of behaviours.

In this paper, we integrate PAC learning into the framework for learning DOTAs. This involves replacing the exact equivalence query with PAC-style equivalence query. To further reduce the number of such equivalence queries, we also integrate the idea of *comparators* [28,9] into the learning framework. The comparator enforces that the quality of intermediate hypotheses obtained during learning does not decrease, by finding the smallest difference between successive hypotheses and then perform one membership query. This has the advantage of replacing some equivalence queries by membership queries, which accelerates the learning process. Replacing exact equivalence queries with PAC-style equivalence queries also introduces other problems. In particular, the distribution of inputs from which the test cases are sampled become very important. In general, sampling from a naïve uniform distribution of action and delay times is unlikely to yield good results, as too few of the samples are focused on the “interesting” parts of system behaviors. Hence, we design special sampling techniques adapted to our setting. Second, in contrast to exact decision procedures for equivalence which are likely to produce minimal counterexamples, there are no such guarantees for PAC-style testing. While this does not affect theoretical correctness of the algorithm, it can lead the algorithm to produce unnecessarily large learned models. Hence, we introduce a method for minimizing counterexamples that involves only membership queries to the teacher. In summary, the contributions of this paper are as follows.

- We describe the PAC learning of deterministic one-clock timed automata. In this setting, both membership and equivalence queries are conducted via testing, with PAC-style equivalence checking replacing exact equivalence queries.
- We accelerate learning by adding a comparator component to reduce the number of equivalence queries. We also propose approaches for better sampling and counterexample minimization to improve learning performance.
- We produce a prototype implementation of our methods, and perform experiments on a number of randomly generated automata, as well as a model of the functional

specification of the TCP protocol. These experiments suggest that DOTAs can be learned under the more realistic assumptions of this paper.

The rest of the paper is organized as follows. In Section 2, we review the learning algorithm for deterministic one-clock timed automata and PAC learning of DFA. Section 3 describes the PAC learning framework for DOTA in detail, including improvements such as comparators, a special sampling method and the counterexample minimization. In Section 4, we extend this PAC framework to the case of normal teachers. The experimental results are reported in Section 5. Section 6 discusses related work. Finally, Section 7 concludes this paper.

2 Preliminaries

Let \mathbb{N} be the natural numbers and $\mathbb{R}_{\geq 0}$ be the non-negative reals numbers. We use \top to stand for true and \perp for false. Let $\mathbb{B} = \{\top, \perp\}$.

2.1 Deterministic one-clock timed automata

In this paper, we consider a subclass of timed automata [2] that are deterministic and contain only a single clock, called *Deterministic One-Clock Timed Automata* (DOTA). Let c be the 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 $\bowtie \in \{=, <, >, \leq, \geq\}$.

Definition 1 (One-clock timed automata). A one-clock timed automata is a 6-tuple $\mathcal{A} = (\Sigma, Q, q_0, F, c, \Delta)$, where Σ is a finite set of actions, Q is a finite set of locations, q_0 is the initial location, $F \subseteq Q$ is a set of final 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 $\delta \in \Delta$ is a 5-tuple (q, σ, ϕ, b, q') , where $q, q' \in Q$ are the source and target locations, $\sigma \in \Sigma$ is an action, $\phi \in \Phi_c$ is a clock constraint, and b is the reset indicator. Such δ allows a jump from q to q' by performing an action σ if the current clock valuation ν satisfies the constraint ϕ . Meanwhile, clock c is reset to zero if $b = \top$ and remains unchanged otherwise. A *clock valuation* is a function $\nu : c \mapsto \mathbb{R}_{>0}$ that assigns a non-negative real number to the clock. For $t \in \mathbb{R}_{\geq 0}$, let $\nu + t$ be the clock valuation with $(\nu + t)(c) = \nu(c) + t$. A *timed state* of \mathcal{A} is a pair (q, ν) , where $q \in Q$ and ν is a clock valuation. A *timed action* is a pair (σ, t) that indicates the action σ is applied after t time units since the occurrence of the previous action. A *run* ρ of \mathcal{A} is a finite sequence $\rho = (q_0, \nu_0) \xrightarrow{\sigma_1, t_1} (q_1, \nu_1) \xrightarrow{\sigma_2, t_2} \dots \xrightarrow{\sigma_n, t_n} (q_n, \nu_n)$ where $\nu_0 = 0$, and for all $1 \leq i \leq n$ there exists a transitions $(q_{i-1}, \sigma_i, \phi_i, b_i, q_i) \in \Delta$ such that $\nu_{i-1} + t_i$ satisfies ϕ_i , and $\nu_i(c) = 0$ if $b_i = \top$, $\nu_i(c) = \nu_{i-1}(c) + t_i$ otherwise. The *timed trace* of a run ρ is a *timed word trace* $\text{trace}(\rho) = (\sigma_1, t_1) (\sigma_2, t_2) \dots (\sigma_n, t_n)$.

Since time values t_i represent *delay* times, we call such a timed trace a *delay-timed word*, denoted as ω . If ρ is an accepting run of \mathcal{A} , $\text{trace}(\rho)$ is called an *accepting timed word*. The *recognized timed language* of \mathcal{A} is the set of accepting delay-timed words, i.e., $\mathcal{L}(\mathcal{A}) = \{\text{trace}(\rho) \mid \rho \text{ is an accepting run of } \mathcal{A}\}$. The corresponding *reset-delay-timed word* can be defined as $\text{trace}_r(\rho) = (\sigma_1, t_1, b_1) (\sigma_2, t_2, b_2) \dots (\sigma_n, t_n, b_n)$,

denoted as ω_r , where each b_i is the reset indicator for δ_i . The *recognized reset-timed language* $\mathcal{L}_r(\mathcal{A})$ is defined as $\{\text{trace}_r(\rho) \mid \rho \text{ is an accepting run of } \mathcal{A}\}$.

The delay-timed word $\omega = (\sigma_1, t_1)(\sigma_2, t_2) \cdots (\sigma_n, t_n)$ is observed outside, from the view of the global clock. On the other hand, the behaviour can also be observed inside, from the view of the local clock. This results in a *logical-timed word* of the form $\gamma = (\sigma_1, \mu_1)(\sigma_2, \mu_2) \cdots (\sigma_n, \mu_n)$ with $\mu_i = t_i$ if $i = 1 \vee b_{i-1} = \top$ and $\mu_i = \mu_{i-1} + t_i$ otherwise. We will denote the mapping from delay-timed words to logical-timed words above by Γ . Similarly, we introduce *reset-logical-timed word* $\gamma_r = (\sigma_1, \mu_1, b_1)(\sigma_2, \mu_2, b_2) \cdots (\sigma_n, \mu_n, b_n)$ as the counterpart of $\omega_r = (\sigma_1, t_1, b_1)(\sigma_2, t_2, b_2) \cdots (\sigma_n, t_n, b_n)$ in terms of the local clock. Without any substantial change, we can extend the mapping Γ to map reset-delay-timed words to reset-logical-timed words. The *recognized logical-timed language* of \mathcal{A} is given as $L(\mathcal{A}) = \{\Gamma(\text{trace}(\rho)) \mid \rho \text{ is an accepting run of } \mathcal{A}\}$, and the *recognized reset-logical-timed language* of \mathcal{A} as $L_r(\mathcal{A}) = \{\Gamma(\text{trace}_r(\rho)) \mid \rho \text{ is an accepting run of } \mathcal{A}\}$.

Definition 2 (Deterministic OTA). *An OTA is a deterministic one-clock timed automaton (DOTA) if there is at most one run for a given delay-timed word.*

We say a DOTA \mathcal{A} is *complete* if for any location q and action σ , the constraints form a partition of $\mathbb{R}_{\geq 0}$. Any incomplete DOTA \mathcal{A} can be transformed into a complete DOTA accepting the same timed language by adding a non-accepting *sink* location (see more details in [5]).

2.2 Exact learning algorithm for DOTAs

In this section, we describe the active learning problem for DOTA and the learning algorithms. We refer to [5] for more details. Active learning of a DOTA assumes the existence of a teacher who can answer two kinds of queries: membership and equivalence queries. We will consider two different settings, depending on whether the teacher also provides clock-reset information along with answers to queries.

A *smart teacher* permits a *logical-timed word* as input to a membership query, and returns whether the timed-word is accepted, as well as reset information at each transition along the trace. Moreover, if the equivalence query yields a counterexample, the counterexample is provided as a reset-delay-timed word. In practical applications, this corresponds to the case where some parts of the model (information of clock-reset) are known by testing or watchdogs (refer to the concept of testable system in [14,13]). This also conforms with the idea of combining black-box learning with white-box techniques, as exploited in [17].

A *normal teacher* corresponds to the usual case for active learning of automata. The teacher permits a delay-timed word as input to a membership query, and only returns whether the timed word is accepted. The equivalence query returns a delay-timed word as a counterexample in the non-equivalent case. The active learning problem in both settings is to learn DOTAs by asking only these two kinds of queries.

The algorithm converts the learning problem to that of learning the reset-logical-timed language, based on the following theorem in [5].

Theorem 1. *Given two DOTAs \mathcal{A} and \mathcal{B} , if $L_r(\mathcal{A}) = L_r(\mathcal{B})$, then $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.*

In the smart teacher setting, the conversion is direct. The problem of learning the reset-logical-timed language follows existing techniques for learning symbolic automata. The algorithm maintains a *timed observation table* \mathbf{T} to store answers from all previous queries. Once the learner has gained sufficient information, i.e., \mathbf{T} is *closed* and *consistent*, a hypothesis \mathcal{H} is constructed. Then the learner poses an equivalence query to the teacher to judge the equivalence between the hypothesis and the target model. If equivalent, the algorithm terminates with the answer \mathcal{H} . Otherwise, the teacher responds with a reset-delay-timed word ω_r as a counterexample. After processing ω_r , the algorithm starts a new round of learning. The whole procedure repeats until the teacher gives a positive answer for an equivalence query.

In the case of normal teacher, the learner needs to guess the reset information on each transition discovered in the observation table. At each iteration, the learner guesses all needed reset information and forms a number of table candidates. These table candidates are put into a priority queue, ordered by the number of needed guesses. Each iteration begins by taking the first table candidate from the queue. Operations on the table is then the same as the smart teacher case. Termination of the algorithm is due to the fact that the learner will eventually consider the case where all guesses are correct. Due to the needed guesses, the complexity of the algorithm is exponential in the size of the learned model.

2.3 PAC learning of DFA

In reality, even in the case of DFA, it is difficult to implement teachers that can answer the equivalence query exactly. Hence, Angluin also introduced the concept of PAC learning for DFA in [7]. We review the basic ideas here.

Assume we are given a probability distribution \mathcal{P} over elements of the language Σ^* . Fix a target regular language $L \subseteq \Sigma^*$. Let L_H be the recognized regular language of an hypothesis H . The quality of H is defined by its distance from L , that is, the probability of choosing a mismatched word $\omega \in \Sigma^*$ that belongs to one language but not the other. The set of all mismatched words is exactly the symmetric difference of the languages L and L_H . Hence, the distance is defined as $\mathcal{P}(L \oplus L_H)$, where $L \oplus L_H = L \setminus L_H \cup L_H \setminus L$.

Definition 3 (PAC-style correctness for DFA). *Let ϵ be the error parameter and δ the confidence parameter. We say a learning algorithm is PAC(ϵ, δ)-correct if its output DFA hypothesis H satisfies $\Pr(\mathcal{P}(L \oplus L_H) \leq \epsilon) \geq 1 - \delta$, where \Pr represents the probability of the event $\mathcal{P}(L \oplus L_H) \leq \epsilon$.*

Under this setting, we replace exact equivalence checking, i.e, whether $L_H = L$, with the checking of approximate equivalence. In other words, we check approximate equivalence by randomly sampling test sequences according to a certain distribution. The minimum number of tests required for each equivalence query to ensure the above PAC-style correctness depends on the error and confidence parameters as well as the number of previous equivalence queries. This number was first introduced in [7] for learning DFA and then used in the PAC learning of symbolic automata [21].

Theorem 2. *The DFA learning algorithm PAC-learns a regular language L if the i -th equivalence query tests $r_i = \frac{1}{\epsilon} (\ln \frac{1}{\delta} + (i + 1) \ln 2)$ random words from a fixed distribution over Σ^* without finding a counterexample.*

3 PAC learning of DOTA

In this section, we explain the PAC learning algorithm utilized to obtain a DOTA approximating the target timed language. In contrast to the learning algorithm given in [5], where equivalence checking is conducted between a hypothesis and the target model, here we allow more flexible implementation of the teacher via testing on the target system. In our PAC learning, membership queries as well as equivalence queries are conducted by testing on the implementation of the system.

3.1 PAC-style correctness

Let \mathcal{P} be a probability distribution over elements of the delay-timed language $(\Sigma \times \mathbb{R}_{\geq 0})^*$. Again, let $\mathcal{L} \subseteq (\Sigma \times \mathbb{R}_{\geq 0})^*$ be the timed language of the target system, and $\mathcal{L}(\mathcal{H})$ be the timed language of the hypothesis \mathcal{H} . As before, the quality of \mathcal{H} is defined as $\mathcal{P}(\mathcal{L} \oplus \mathcal{L}(\mathcal{H}))$, where $\mathcal{L} \oplus \mathcal{L}(\mathcal{H}) = \mathcal{L} \setminus \mathcal{L}(\mathcal{H}) \uplus \mathcal{L}(\mathcal{H}) \setminus \mathcal{L}$.

Definition 4 (PAC-style correctness for DOTA). *Let ϵ be the error parameter and δ the confidence parameter. We say a learning algorithm for DOTA is PAC(ϵ, δ)-correct if its output timed hypothesis \mathcal{H} satisfies:*

$$\Pr(\mathcal{P}(\mathcal{L} \oplus \mathcal{L}(\mathcal{H})) \leq \epsilon) \geq 1 - \delta, \quad (1)$$

where \Pr represents the probability of the event $\mathcal{P}(\mathcal{L} \oplus \mathcal{L}(\mathcal{H})) \leq \epsilon$.

As before, PAC-style correctness can be obtained by performing a sufficient number of tests for each equivalence query. The main result for the DOTA case is given below, following [23].

Theorem 3. *The DOTA learning algorithm PAC-learns a timed language \mathcal{L} if the i -th equivalence query tests*

$$r_i = \frac{1}{\epsilon} \left(\ln \frac{1}{\delta} + (i + 1) \ln 2 \right) \quad (2)$$

random delay-timed words from a fixed distribution over $(\Sigma \times \mathbb{R}_{\geq 0})^$ without finding a counterexample.*

3.2 PAC-style equivalence query

In this section, we present the overall procedure for the PAC-style equivalence query. The procedure is shown in Algorithm 1. Three improvements to the procedure will be discussed in the next three subsections.

The equivalence query accepts as input the hypothesis \mathcal{H} , the count i of the current query, the error parameter ϵ , and the confidence parameter δ . We first compute the number of samples needed according to Equation (2). Then, we repeatedly draw samples from a distribution. The choice of distribution is significant for the learning performance, and will be discussed in detail in Section 3.3. For each sample ω (a delay-timed word), we test it on both the target system \mathcal{S} and the hypothesis \mathcal{H} (testing on the target system uses a membership query). The test on \mathcal{S} returns a pair v, ω_r , where v represents whether ω is an accepted timed word according \mathcal{S} , and ω_r is the reset-delay-timed

Algorithm 1: PAC-style equivalence query `pac_equivalence`($\mathcal{H}, i, \epsilon, \delta$)

input : a hypothesis \mathcal{H} ; the count i of current equivalent query; error parameter ϵ ; confidence parameter δ .
output: *equivalent*: a boolean value to identify whether \mathcal{H} passes all tests;
ctx: a counterexample.

```

1 equivalent  $\leftarrow \top$ ;
2 counter  $\leftarrow 1$ ;
3 testNum  $\leftarrow \frac{1}{\epsilon} (\ln \frac{1}{\delta} + (i + 1) \ln 2)$ ;
4 while counter < testNum do
5    $\omega \leftarrow \text{sample}(\mathcal{P})$ ; //  $\mathcal{P}$  is a distribution over  $(\Sigma \times \mathbb{R}_{\geq 0})^*$ 
6    $v, \omega_r \leftarrow \text{test\_dtw}(\omega, \mathcal{S})$ ; // test a timed word  $\omega$  on system  $\mathcal{S}$ 
7    $v', \omega'_r \leftarrow \text{test\_dtw}(\omega, \mathcal{H})$ ; // test a timed word  $\omega$  on hypothesis  $\mathcal{H}$ 
8   if  $v \neq v'$  then
9     equivalent  $\leftarrow \perp$ ; ctx  $\leftarrow \omega_r$ ;
10    return equivalent, ctx;
11   counter  $\leftarrow$  counter + 1;
12 return equivalent, ctx

```

word corresponding to ω . Likewise, the test on \mathcal{H} returns a pair v', ω'_r . If $v \neq v'$, then ω is a counterexample to the equivalence between \mathcal{H} and \mathcal{S} , and is returned directly. Otherwise, if all tests pass, we conclude \mathcal{H} is $PAC(\epsilon, \delta)$ -correct, based on Theorem 3.

Two further improvements reduce the number of needed equivalence queries by adding a *comparator* (Section 3.4) and the *counterexample minimization* (Section 3.5).

3.3 Sampling mechanism

The choice of the sampling distribution over $(\Sigma \times \mathbb{R}_{\geq 0})^*$ is important to whether PAC learning yields good results in real applications. While the theory guarantees the success of learning under any distribution, an inappropriate choice of distribution may lead to models that are not useful in practice. In particular, we observe that a naïve uniform distribution of action and time values is not useful in our case. The reason is that for many examples, e.g. the TCP protocol and the randomly generated automata on which we performed experiments, the vast majority of timed traces under uniform distribution are invalid for the automata. Hence, only a very small proportion lead to valid paths and test interesting behaviours of the system. This situation may also occur for other real-life systems. For many reactive systems and protocols, an input that is completely randomly generated will most likely be invalid in the current state, and hence will not test the interesting aspect of the system.

We address this problem by designing a custom sampling mechanism. Our aim is for one half of the overall distribution to consist of timed words that are guaranteed to be valid for the system. The other half consists of possibly invalid timed words, obtained from the valid ones by introducing a random change. In more detail, for both valid and possibly invalid timed traces, we first choose the length uniformly between 1 and an upper bound M (i.e. 1.5 times the number of locations in the experiments). For a given length, we could sample valid timed traces by repeatedly sampling random timed words, testing each on the system, and taking only the valid traces. This method is inefficient if the vast majority of timed words are invalid. We design a more efficient sampling method as follows. First, we perform a random walk on the locations of the system starting from the initial location. This gives a sequence of actions and bounds on the *logical* time of the timed trace. Next, we uniformly sample the logical time, subject

to the obtained bounds, as well as the constraint that if the previous transition is not a reset, then the logical time should be greater than or equal to the previous logical time. To make sure that we will test traces with integer time values, we actually sample from the allowed *regions* of logical time, so that about half of sampled time values are integers. Otherwise, most of the sampled time values will contain fractions. Finally, the resulting logical-timed word is converted to delay-timed word, which is guaranteed to be valid. To sample possibly invalid timed traces of a given length, we first sample a valid timed trace of the same length using the above procedure, then randomly change one transition to a timed word with uniformly chosen action and time.

This sampling mechanism yield timed traces that are more likely to reflect interesting behaviours of the system. We note that while the sampling depends on the target system, it does not reveal the internal structure of the system to the learner. It only helps the learner by providing counterexamples that are more likely to be relevant. In real applications, this sampling distribution can be approximated by sampling from user inputs (which are more likely to be valid) and their slight variations. Another way to approximate the distribution is to first sample random timed words, then remove most of the invalid ones as mentioned before. The target system continues to be viewed as a black-box. In Section 5, we will show that while the learning algorithm succeeds with any sampling distribution, the model learned using the distribution described above are far more likely to be completely correct (or at least very close to the target system from a human point of view) than using a naïve uniform distribution.

3.4 Comparator

During the learning process, the aforementioned algorithm generates a series of hypotheses. Ideally, we would prefer that each successive hypothesis gradually approaches the target system according to some measure. However, this may not be the case. As observed in [23] for symbolic automata, processing of counterexamples will generate two kinds of changes to the hypothesis. The first kind is called *expansive modification*, which means the latter hypothesis \mathcal{H}' has more states and/or transitions than the former hypothesis \mathcal{H} . The second is called *non-expansive modification*, which implies that between the two hypotheses, only the symbols of the alphabet on the transitions differ. It is noted in [23] that \mathcal{H}' is closer to the target system than \mathcal{H} .

However, in the case of expansive modification, this cannot be guaranteed. Vaandrager et al. showed in [9,28] that the successive hypothesis is not always better than the previous one, under a well-known metric based on minimal length counterexamples. To correct this, they proposed a modification to L^* to make sure that each *stable* hypothesis (see Definition 6) is at least as good as the previous one. Although the modification is for the DFA setting, we find that it is still applicable to the DOTA case. The distance metric to measure the quality of a hypothesis is defined as follows.

Definition 5 (Metric function). Let $\mathcal{L}(\mathcal{H})$ and $\mathcal{L}(\mathcal{H}')$ be timed languages of two DOTAs \mathcal{H} and \mathcal{H}' . The ultrametric function d is

$$d(\mathcal{L}(\mathcal{H}), \mathcal{L}(\mathcal{H}')) = \begin{cases} 0 & \text{if } \mathcal{L}(\mathcal{H}) = \mathcal{L}(\mathcal{H}') \\ 2^{-n} & \text{otherwise,} \end{cases} \quad (3)$$

where n is the length of a minimal timed word that distinguishes $\mathcal{L}(\mathcal{H})$ and $\mathcal{L}(\mathcal{H}')$.

Algorithm 2: Find new stable hypothesis $\text{comparator}(\mathcal{H}, \mathcal{H}')$

```

input : current stable hypothesis  $\mathcal{H}$ ; new hypothesis  $\mathcal{H}'$ .
output: new stable hypothesis  $\mathcal{H}$ .
1 compareFlag  $\leftarrow \perp$ ;
2 repeat
3   /* obtain minimal timed word  $\omega$  that distinguishes  $\mathcal{H}$  and  $\mathcal{H}'$ . */
4    $\omega \leftarrow \text{min\_distinguishing\_dtw}(\mathcal{H}, \mathcal{H}')$ ;
5    $v, \omega_r \leftarrow \text{test\_dtw}(\omega, \mathcal{S})$ ;           // test a timed word  $\omega$  on system  $\mathcal{S}$ 
6    $v', \omega'_r \leftarrow \text{test\_dtw}(\omega, \mathcal{H}')$ ;   // test a timed word  $\omega$  on hypothesis  $\mathcal{H}'$ 
7   if  $v \neq v'$  then
8      $\text{ctx} \leftarrow \omega_r$ ;                       // found a counterexample
9      $\text{ctx\_processing}(\mathbf{T}, \text{ctx})$ ;             // handle the counterexample
10    Make table  $\mathbf{T}$  closed and consistent;
11    Construct new hypothesis  $\mathcal{H}'$ ;
12  else
13     $\mathcal{H} \leftarrow \mathcal{H}'$ ;                         // set  $\mathcal{H}'$  as new stable hypothesis
14    compareFlag  $\leftarrow \top$ ;
15 until compareFlag =  $\top$ ;
return new stable hypothesis  $\mathcal{H}$ ;

```

Definition 6 (Stable hypothesis). Let \mathcal{S} be the target system, and let \mathcal{H} and \mathcal{H}' be two hypotheses in the learning process. Then \mathcal{H}' is called stable if $d(\mathcal{L}(\mathcal{H}), \mathcal{L}(\mathcal{S})) \geq d(\mathcal{L}(\mathcal{H}'), \mathcal{L}(\mathcal{S}))$.

The procedure for finding stable hypotheses with a *comparator* is shown in Algorithm 2. For each newly learned hypothesis \mathcal{H}' , before asking the teacher an equivalence query, it is compared with the current stable hypothesis \mathcal{H} . This involves first generating a minimal-length sequence ω (a delay-timed word) distinguishing \mathcal{H} and \mathcal{H}' , which can be achieved via a language equivalence checking since the model \mathcal{H} and \mathcal{H}' are known. Then ω is tested against the target system \mathcal{S} . If the result is inconsistent with that of \mathcal{H}' , the comparator found a counterexample to \mathcal{H}' and returns the corresponding reset-delay-timed word ω_r to the learner to construct a new (and bigger) hypothesis. Otherwise (when the outputs are consistent), we promote \mathcal{H}' to be the new stable hypothesis, and proceeds to perform a PAC-style equivalence query. This ensures that each stable hypothesis is at least as good as the previous one according to the metric function. It also has the practical effect of reducing the number of equivalence queries (replacing some of them by membership queries). This is particularly significant in the PAC learning setting as the number of tests of each equivalence query increases with the number of previously performed equivalence queries.

Because \mathcal{H} and \mathcal{H}' are both explicit DOTAs (in contrast to the target system which is a black box), finding the minimal distinguishing timed word between them can use the same timed language inclusion tests in [5,24] (or using the technique of complementation and intersection of automata). The following theorem is adapted from [28].

Theorem 4. *The execution of Algorithm 2 terminates, and each stable hypothesis is at least as good as the previous one according to the metric function.*

3.5 Counterexample minimization

When the equivalence query is answered using a decision procedure, the decision procedure can usually return counterexamples of small size. In fact, existing work on sym-

bolic automata [23] introduces the concept of *helpful teacher* to indicate the ability of the teacher to return a minimal counterexample (which is a counterexample of minimal length and also minimal with respect to lexicographic order). Under this assumption, the learning algorithm for symbolic automata has better theoretical properties.

In the case of exact learning of DOTA, the correctness and termination of the algorithm [5] do not depend on being provided minimal counterexamples. However, the actual performance of the algorithm can still be significantly affected. In particular, if the counterexample is not minimal, it can lead to unnecessary splitting of edges in the learned model. For example, a guard $[5, \infty)$ on a transition can be unnecessarily split into $[5, 7)$ and $[7, \infty)$ on two transitions, if the learner is provided with a counterexample with time value 7 first, whereas directly providing a counterexample with time value 5 will not lead to splitting. This is particularly significant in the case of normal teacher, as its complexity is exponential in the number of edges in the learned model.

Hence, we propose a simple heuristic for improving a counterexample using only membership queries. First, when performing PAC-style equivalence queries, samples are tested in increasing order by length. When a minimal length counterexample (as a delay-timed word) is found, it is minimized according to lexicographic order as follows. We first run the timed word on the hypothesis, obtaining the corresponding logical-timed word. Then, for each transition of the logical-timed word starting from the beginning, we decrease the logical time step-by-step, at each step converting back to delay-timed word using the reset information and send the result as a membership query. The new delay-timed word is kept only if it is still a counterexample. Note this procedure finds locally minimal counterexamples, but is not guaranteed to find the globally minimal one.

3.6 The whole procedure

Integrating the previously introduced techniques, the overall learning framework is summarized in Algorithm 3. As described in Section 2.2, the learner performs several rounds of membership queries to make the observation table \mathbf{T} prepared (closed and consistent) before constructing a new hypothesis. Then, the comparator is used to make sure that the current stable hypothesis always approaches the target system according to the metric function in Definition 5, which reduces the number of equivalence queries. On the stable hypothesis, PAC-style equivalence query is performed by testing. The whole procedure repeats until the PAC-style equivalence query terminates without finding a counterexample, so the hypothesis is considered correct with some probability of error. Since the new learning procedure only modifies the equivalence query, the main theoretical results from [5] still hold. This allows us to state the following main correctness theorem for the new procedure. Note that in [21,23] for the case of symbolic automata, termination is only with probability 1 if the alphabet is infinitely divisible. In our case, the endpoints of guards are integers, hence the algorithm is guaranteed to terminate.

Theorem 5. *Algorithm 3 terminates after polynomial number of membership and PAC-style equivalence queries, and learns the target system in a probably approximately correct manner with error ϵ and confidence δ .*

Algorithm 3: PAC Learning of DOTAs

```

input : timed observation table  $\mathbf{T}$ ; error  $\epsilon$ ; confidence  $\delta$ .
output: hypothesis  $\mathcal{H}$ , which is a PAC( $\epsilon, \delta$ )-correct output for the target timed language  $\mathcal{L}$ .
1 Initialize timed observation table  $\mathbf{T}$ ;
2 Make  $\mathbf{T}$  closed and consistent;
3 Construct a hypothesis  $\mathcal{H}'$  from  $\mathbf{T}$ ;
4  $\mathcal{H} \leftarrow \mathcal{H}'$ ; // initial stable hypothesis
5  $equivalent \leftarrow \perp$ ;
6  $i \leftarrow 0$ ; // the number of PAC-style equivalence queries
7 while  $equivalent = \perp$  do
8   if  $i > 0$  then
9      $\mathcal{H} \leftarrow \text{comparator}(\mathcal{H}, \mathcal{H}')$ ; // current stable hypothesis
10     $i \leftarrow i + 1$ ;
11     $equivalent, ctx \leftarrow \text{pac\_equivalence}(\mathcal{H}, i, \epsilon, \delta)$ ; // PAC equivalence query
12    if  $equivalent = \perp$  then
13       $\text{ctx\_processing}(\mathbf{T}, ctx)$ ; // handle counterexample
14      Make table  $\mathbf{T}$  closed and consistent;
15      Construct a hypothesis  $\mathcal{H}'$  from  $\mathbf{T}$ ;
16 return  $\mathcal{H}$ ;
```

4 Extending PAC learning to normal teacher

In this section, we extend the algorithm given in [5] for the case of normal teacher to the PAC learning setting. The needed changes are similar to the smart teacher case, with each equivalence query for a hypothesis constructed from a prepared table candidate replaced by a PAC-style equivalence query. It should be noted that the count i of current equivalence query still increases with each query, regardless of the tree structure caused by the guesses. This can be justified as follows: in the derivation of Equation (3), the number of needed queries is set so that the total probability of making a mistake (resulting in a model with error greater than ϵ) is at most δ , with $\delta/2^{i+1}$ being the bound on the probability of making a mistake at the i -th equivalence query. In the normal teacher setting, we should still accumulate the probabilities of making mistakes along different branches of the tree, so the derivation of Equation (3) is still the same as before. As for the improvements reported in Section 3, they are still applied to the normal teacher setting. While in counterexample minimization, we run the timed word on the hypothesis, obtaining the corresponding logical-timed word based on the guessed resets.

The theoretical results (following [5]) are similar to the smart teacher case, except the complexity is now exponential due to the guessing of resets.

Theorem 6. *The learning process for the normal teacher terminates after exponential number of membership and PAC-style equivalence queries, and learns the target system in a probably approximately correct manner with error ϵ and confidence δ .*

In a variant of the above procedure, we can also group prepared table candidates by level, for example by the number of guesses made. If there are m_i tables at level i , then the number of samples for each PAC-style equivalence query at level i is modified to be $\frac{1}{\epsilon}(\ln \frac{1}{\delta} + (i+1) \ln 2 + \ln m_i)$. We can also consider pruning the search tree by removing table candidates that appear to be less promising, for example with lower passing rate at the current iteration (which is similar to the genetic programming method in [26,29]). With such a pruning method, we obtain a procedure that is sound but not necessarily complete or terminating.

5 Implementation and Experimental Results

In order to further investigate the efficiency and scalability of the proposed methods, we implemented a prototype in PYTHON and evaluated it on the functional specification of the TCP protocol and a set of randomly generated DOTAs. All of the experiments were carried out on a MacBook Pro with 8GB RAM, Intel Core i5 with 2.7 GHz and running macOS Catalina 10.15.3. The tool and experiments are available in the tool page https://github.com/MrEnvision/learning_OTTA_by_testing.

5.1 TCP protocol

We refer to [5] for a state diagram specifying the state changes in the functional specification of the TCP protocol. It can be represented as a DOTA \mathcal{A} (see Appendix D of [4]) with $|Q| = 11$ locations, $|\Sigma| = 10$ untimed actions, $|F| = 2$ final locations, and $|\Delta| = 19$ transitions with appropriately specified timing constraints including guards and resets. With our sampling method, comparator, and counterexample minimization, we run the prototype on this example 30 times with the error parameter $\epsilon = 0.001$ and confidence parameter $\delta = 0.001$ in the smart teacher setting. Our tool learned out 30 PAC(ϵ, δ)-correct DOTAs in which 28 models are exactly correct. In theory, the remaining two models should have at least 0.999 accuracy with confidence 0.999. In order to further check the quality of the remaining two models, we test them on 20000 more samples generated from the same distribution in the learning process. Both models have a passing rate of at least 0.9999. The minimum, mean and maximum numbers for membership and PAC equivalence queries are 608, 717.3, 912 and 18, 18.7, 20, respectively. The minimum, mean and maximum numbers of tests in the PAC equivalence queries are 107925, 122565.2 and 143806. The average time of learning is 138.9 seconds.

5.2 Random experiments

We continue to use the random DOTAs in [5] to evaluate our PAC learning method. Additionally, we compare the performances with and without each of our three improvements, i.e. the specific sampling mechanism, the comparator and the counterexample minimization method.

Evaluation results on benchmark. With our specific sampling method, comparator, and counterexample minimization method, the experimental results on the benchmark are shown in Table 1. For each case, we run our tool 10 times and our tool learns all models in the corresponding PAC settings and sometimes generates a model which is exactly equivalent to the target model. The number of tests taken is also quite stable across the random trials, with minimum and maximum numbers usually within 50% of each other in each case.

With/without specific sampling method. We evaluated our tool on the TCP protocol case study and the random examples by replacing the specific sampling method (Section 3.3) with sampling from a naïve uniform distribution. As expected, the algorithm also returns with models which are PAC(ϵ, δ)-correct outputs according to the naïve uniform distribution. However, the learned models sometimes have big differences with the target model even when choosing high accuracy and confidence. For example, when

Table 1: Experimental results on random examples for the smart teacher situation.

Case ID	$ \Delta $	ϵ, δ	#Membership			#Equivalence			#Tests			n_{exact}	r_{pass}	t_{mean}
			N_{\min}	N_{mean}	N_{\max}	N_{\min}	N_{mean}	N_{\max}	N_{\min}	N_{mean}	N_{\max}			
4.4_20	18	0.001	167	173.1	181	28	29.3	30	418765	455798.8	487670	4	0.999945	90.2
4.4_20	18	0.01	140	161.5	178	23	26.7	29	26084	34647.1	41470	0	0.999460	34.5
7.2_10	18	0.001	471	561.1	781	28	31.6	39	333340	424286.5	592363	0	0.999964	88.6
7.2_10	18	0.01	375	487.4	717	23	26.5	31	21024	27676.6	36805	0	0.999695	24.8
7.4_10	26	0.001	746	766.5	796	46	47.7	51	533562	569403.8	618769	3	0.999995	133.8
7.4_10	26	0.01	729	778.0	853	46	48.0	50	49473	54829.8	63353	0	0.999960	59.8
7.6_10	32	0.001	676	832.7	1035	53	61.3	68	749773	1025486.9	1274129	4	1.0	341.5
7.6_10	32	0.01	678	807.7	998	54	58.2	63	72714	87215.3	101603	1	0.999870	137.3
7.4_20	26	0.001	419	442.2	480	41	42.4	43	599115	648487.5	690735	5	0.999986	160.7
7.4_20	26	0.01	382	451.6	733	36	38.8	42	44183	50828.7	62079	0	0.999860	101.0
10.4_20	36	0.001	682	935.5	1492	59	67.2	82	607149	816399.8	1233177	10	1.0	267.4
10.4_20	36	0.01	732	1029.4	1369	56	63.4	77	51185	69562.7	100264	2	0.999960	159.5

Case ID: $n.m.k$, consisting of the number of locations, the size of the alphabet and the maximum constant appearing in the clock constraints, respectively, of the corresponding model.

$|\Delta|$: the number of transitions in the corresponding model.

ϵ, δ : the error and confidence parameters in PAC learning. (Here choose a same value.)

#Membership & #Equivalence & #Tests: the number of conducted membership queries, PAC-style equivalence queries and tests utilized in equivalence queries, respectively. N_{\min} : the minimal, N_{mean} : the mean, N_{\max} : the maximum.

n_{exact} : the number of exactly learned model.

r_{pass} : the average passing rate of the learned model on extra 20000 test cases randomly generated from the same distribution in the learning process.

t_{mean} : the average wall-clock time in seconds, including that taken by the learner and the teacher.

we choose $\epsilon = \delta = 0.001$ and sample testing timed words from the uniform distribution $U(1, 2 \cdot |Q|)$ in the TCP protocol case, the tool learned out models without some transitions back to the initial state which is one of the accepting states.

With/without comparator. As introduced in Section 3.4, the comparator could reduce the number of equivalence queries, and hence the number of test cases needed in such queries. We evaluated our tool without the comparator, and the number of PAC-style equivalence queries and test cases increased by 10% on average.

With/without counterexample minimization. Fig. 1 shows the experimental results with and without counterexample minimization for some of the randomly generated examples. We find that the number of PAC-style equivalence queries and tests increased by around 150% and 400% respectively. Hence, counterexample minimization improves the learning performance significantly on the random examples.

Evaluation on random example in normal teacher setting. Finally, we evaluate the PAC-style learning method in the normal teacher situation. The results are shown in Table 2. As the method still depends on the quality of the provided counterexamples, a few of the cases can no longer be learned within the time limit of 2 minutes, compared to the case of exact equivalence query. Overall, the results still show that our method is effective in the normal teacher setting, which most importantly, provides a way to implement a teacher in practice.

6 Related work

Various attempts have been carried out in the literature on learning timed models, which can be divided into two directions. The first direction is about passive learning. An algo-

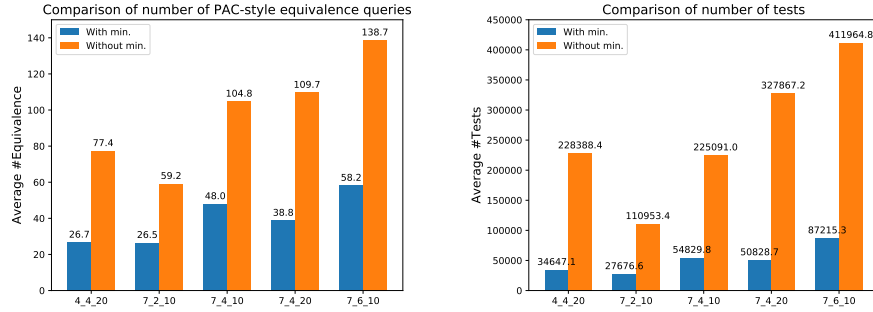


Fig. 1: Experimental results with and without counterexample minimization.

Table 2: Experimental results on random examples for the normal teacher situation.

Case ID	$ \Delta _{\text{mean}}$	ϵ, δ	#Membership			#Equivalence			t_{mean}	# $\mathbf{T}_{\text{explored}}$	#Learnt	n_{exact}
			N_{min}	N_{mean}	N_{max}	N_{min}	N_{mean}	N_{max}				
3_2_10	4.8	0.001	63	142.0	346	5	7.6	12	4.8	96.3	9/10	6
4_2_10	6.8	0.001	128	224.2	394	6	13.0	19	10.3	200.2	9/10	6
5_2_10	8.8	0.001	155	308.1	534	9	15.0	20	12.6	292.6	7/10	7
6_2_10	11.9	0.001	96	446.0	661	9	16.0	22	19.9	454.4	7/10	4

#Membership & #Equivalence: the number of conducted membership and equivalence queries with the cached methods, respectively. N_{min} : the minimal, N_{mean} : the mean, N_{max} : the maximum.

$|\Delta|_{\text{mean}}$: the average number of transitions in the corresponding group.

$\mathbf{T}_{\text{explored}}$: the average number of the explored table instances.

#Learnt: the number of the learnt DOTAs in the group (learnt/total).

rithm was proposed to learn deterministic real-time automata in [33]. A passive learning algorithm for timed automata with one clock was further proposed in [32]. We furthermore refer the readers to [22,25] for learning specialized forms of practical timed systems in a passive manner. A common weakness of passive learning is that the generated model merely accepts all positive traces and rejects all negative ones for the given set of traces, without guaranteeing that it is a correct model of the target system. As to active learning, a learning algorithm for event recording automata [3] is proposed in [16]. The underlying learning algorithm has double-exponential complexity. In [19], Lin et al. proposed an efficient learning method for the same model. Learning techniques for symbolic automata are introduced in [20,12] and An et al. applied the techniques to learning real-time automata [6].

Recently, applying the ideas of PAC learning [31] to model learning is receiving increasing attention. Angluin introduced a PAC learning algorithm of DFA in [7]. In [21], Maler et al. applied PAC learning to symbolic automata. In [10], using PAC learning to obtain an approximate regular model of the set of feasible paths in a program, Chen et al. introduced a novel technique for verification and model synthesis of sequential programs. Another way to replace exact equivalence queries is conformance testing [8,18] via a finite number of testing queries. Well-known methods for conformance testing include W-method [11,15], UIO-method [27], etc. These methods can also be modified to test timed models [13,14]. In [29], Aichernig et al. introduced an approach to learn timed automata based on genetic programming. In subsequent work [26], they combined genetic programming with conformance testing to improve its performance.

7 Conclusion

In this paper, we presented a PAC learning algorithm for black-box systems that can be specified by DOTAs. We relax the ideal setting of a teacher that maintains oracles for both membership queries and exact equivalence queries. In our new setting, both membership and equivalence queries are conducted via testing, with PAC-style equivalence query replacing exact equivalence query. In addition, to reduce the number of equivalence queries, we introduced comparator into our learning framework. We also discussed the sampling approach, and a heuristic method to minimize counterexamples. A prototype is implemented in PYTHON, and is evaluated on the functional specification of the TCP protocol as well as a set of randomly generated examples. The experiments shows the positive effects of each of the improvements on realistic examples. Possible future work include extension to timed automata with multiple clocks.

References

1. Aichernig, B.K., Tappler, M.: Efficient active automata learning via mutation testing. *Journal of Automated Reasoning* **63**(4), 1103–1134 (2019)
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), 253–274 (1999)
4. An, J., Chen, M., Zhan, B., Zhan, N., Zhang, M.: Learning one-clock timed automata (full version). arXiv:1910.10680 (2019), <https://arxiv.org/abs/1910.10680>
5. An, J., Chen, M., Zhan, B., Zhan, N., Zhang, M.: Learning one-clock timed automata. In: Biere, A., Parker, D. (eds.) TACAS 2020. LNCS, vol. 12078, pp. 444–462. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-45190-5_25
6. An, J., Wang, L., Zhan, B., Zhan, N., Zhang, M.: Learning real-time automata. *SCIENCE CHINA Information Sciences* (In press). <https://doi.org/10.1007/s11432-019-2767-4>
7. Angluin, D.: Learning regular sets from queries and counterexamples. *Information and computation* **75**(2), 87–106 (1987)
8. Berg, T., Grinchtein, O., Jonsson, B., Leucker, M., Raffelt, H., Steffen, B.: On the correspondence between conformance testing and regular inference. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, pp. 175–189. Springer (2005). https://doi.org/10.1007/978-3-540-31984-9_14
9. van den Bos, P., Smetsers, R., Vaandrager, F.W.: Enhancing automata learning by log-based metrics. In: Ábrahám, E., Huisman, M. (eds.) IFM 2016. LNCS, vol. 9681, pp. 295–310. Springer (2016). https://doi.org/10.1007/978-3-319-33693-0_19
10. Chen, Y.F., Hsieh, C., Lengál, O., Lii, T.J., Tsai, M.H., Wang, B.Y., Wang, F.: PAC learning-based verification and model synthesis. In: ICSE 2016. pp. 714–724. IEEE (2016)
11. Chow, T.S.: Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering* **4**(3), 178–187 (1978)
12. Drews, S., D’Antoni, L.: Learning symbolic automata. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 173–189. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54577-5_10
13. En-Nouaary, A., Dssouli, R., Khendek, F.: Timed Wp-Method: Testing real-time systems. *IEEE Transactions on Software Engineering* **28**(11), 1023–1038 (2002)
14. En-Nouaary, A., Dssouli, R., Khendek, F., Elqortobi, A.: Timed test cases generation based on state characterization technique. In: RTSS 1998. pp. 220–229. IEEE (1998)

15. Fujiwara, S., Bochmann, G.v., Khendek, F., Amalou, M., Ghedamsi, A.: Test selection based on finite state models. *IEEE Transactions on Software Engineering* **17**(6), 591–603 (1991)
16. Grinchtein, O., Jonsson, B., Leucker, M.: Learning of event-recording automata. *Theor. Comput. Sci.* **411**(47), 4029–4054 (2010)
17. Howar, F., Jonsson, B., Vaandrager, F.W.: Combining black-box and white-box techniques for learning register automata. In: Steffen, B., Woeginger, G.J. (eds.) *Computing and Software Science - State of the Art and Perspectives*, LNCS, vol. 10000, pp. 563–588. Springer (2019). https://doi.org/10.1007/978-3-319-91908-9_26
18. Lee, D., Yannakakis, M.: Principles and methods of testing finite state machines—a survey. *Proceedings of the IEEE* **84**(8), 1090–1123 (1996)
19. Lin, S., André, É., Dong, J.S., Sun, J., Liu, Y.: An efficient algorithm for learning event-recording automata. In: Bultan, T., Hsiung, P. (eds.) *ATVA 2011*. LNCS, vol. 6996, pp. 463–472. Springer (2011). https://doi.org/10.1007/978-3-642-24372-1_35
20. Maler, O., Mens, I.: Learning regular languages over large alphabets. In: Ábrahám, E., Havelund, K. (eds.) *TACAS 2014*. LNCS, vol. 8413, pp. 485–499. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_41
21. Maler, O., Mens, I.: A generic algorithm for learning symbolic automata from membership queries. In: Aceto, L., Bacci, G., Bacci, G., Ingólfssdóttir, A., Legay, A., Mardare, R. (eds.) *Models, Algorithms, Logics and Tools - Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday*. LNCS, vol. 10460, pp. 146–169. Springer (2017). https://doi.org/10.1007/978-3-319-63121-9_8
22. Mediouni, B.L., Nouri, A., Bozga, M., Bensalem, S.: Improved learning for stochastic timed models by state-merging algorithms. In: Barrett, C.W., Davies, M., Kahsai, T. (eds.) *NFM 2017*. LNCS, vol. 10227, pp. 178–193. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-57288-8_13
23. Mens, I.: Learning regular languages over large alphabets. (Apprentissage de langages réguliers sur des alphabets de grandes tailles). Ph.D. thesis, Grenoble Alpes University, France (2017), <https://tel.archives-ouvertes.fr/tel-01792635>
24. Ouaknine, J., Worrell, J.: On the language inclusion problem for timed automata: Closing a decidability gap. In: *Proceedings of the 19th IEEE Symposium on Logic in Computer Science, LICS 2004*. pp. 54–63. IEEE Computer Society (2004)
25. Pastore, F., Micucci, D., Mariani, L.: Timed k-Tail: Automatic inference of timed automata. In: *Proceedings of 10th IEEE International Conference on Software Testing, Verification and Validation, ICST 2017*. pp. 401–411. IEEE Computer Society (2017)
26. Pferscher, A., Aichernig, B., Tappler, M.: From passive to active: Learning timed automata efficiently. In: *NFM’20 (2020)*, <https://ti.arc.nasa.gov/events/nfm-2020/>
27. Shen, Y.N., Lombardi, F., Dahbura, A.T.: Protocol conformance testing using multiple uio sequences. *IEEE Transactions on Communications* **40**(8), 1282–1287 (1992)
28. Smetsers, R., Volpato, M., Vaandrager, F.W., Verwer, S.: Bigger is not always better: on the quality of hypotheses in active automata learning. In: *ICGI 2014*. pp. 167–181 (2014), <http://proceedings.mlr.press/v34/smetsters14a.html>
29. Tappler, M., Aichernig, B.K., Larsen, K.G., Lorber, F.: Time to learn - learning timed automata from tests. In: André, É., Stoelinga, M. (eds.) *FORMATS 2019*. LNCS, vol. 11750, pp. 216–235. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-29662-9_13
30. Vaandrager, F.: Model learning. *Commun. ACM* **60**(2), 86–95 (2017)
31. Valiant, L.G.: A theory of the learnable. *Commun. ACM* **27**(11), 1134–1142 (1984)
32. Verwer, S., De Weerd, M., Witteveen, C.: The efficiency of identifying timed automata and the power of clocks. *Information and Computation* **209**(3), 606–625 (2011)
33. Verwer, S., de Weerd, M., Witteveen, C.: Efficiently identifying deterministic real-time automata from labeled data. *Machine learning* **86**(3), 295–333 (2012)