One-clock deterministic timed automata are efficiently identifiable in the limit

Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen

Delft University of Technology
{S.E.Verwer, M.M.deWeerdt, C.Witteveen}@tudelft.nl

Abstract. We study the complexity of identifying (learning) timed automata in the limit from data. In previous work, we showed that in order for timed automata to be efficiently identifiable in the limit, it is necessary that they are deterministic and that they use at most one clock. In this paper, we show this is also sufficient: we provide an algorithm that identifies one-clock deterministic timed automata efficiently in the limit.

1 Introduction

Timed automata [1] (TAs) are finite state models that model timed events using an explicit notion of time. They can be used to model and reason about real-time systems [2]. In practice, however, the knowledge required to completely specify a TA model is often insufficient. An alternative is to try to induce the specification of a TA from observations. This approach is also known as *inductive inference*. The idea behind inductive inference is that it is often easier to find examples of the behavior of a real-time system than to specify the system in a direct way. Inductive inference then provides a way to find a TA model that characterizes the (behavior of the) real-time system that produced these examples.

In our case, we can monitor the occurrences of the events in a real-time system. This results in a set of labeled (positive and negative) time stamped event sequences. The exact problem we want to solve is to find the TA model that (most likely) produced this data. This is called *TA identification*. Naturally, we want to solve this problem in an efficient way, i.e., without requiring exponential amounts of space or time.

Unfortunately, identifying a TA efficiently is difficult due to the fact that the identification problem for non-timed deterministic finite state automata (DFAs) from a finite data set is already NP-complete [3]. This property easily generalizes to the problem of identifying a TA (by setting all time values to 0). Thus, unless $P = NP$, a TA cannot be identified efficiently from finite data. Even more troublesome is the fact that the DFA identification problem from finite data cannot even be approximated within any polynomial [4]. Hence (since this also generalizes), the TA identification problem from finite data is also inapproximable. However, both of these results require that the input for the identification problem is finite. While in normal decision problems this is very natural, in an identification problem the amount of input data is somewhat arbitrary: more data can be sampled if necessary. Therefore, it makes sense to study the behavior of an identification process when it is given more and more data. The framework that studies this behavior is called *identification in the limit* [5].

The identifiability of many interesting models (or languages) have been studied within the framework of learning in the limit. For instance, DFAs have been shown to be efficiently identifiable in the limit from labeled examples [6], but non-deterministic finite state automata (NFAs) have been shown not to be efficiently identifiable in this way [7]. Because of this, we restrict our attention to deterministic timed automata (DTAs). Unfortunately, in previous work we have shown that DTAs in general cannot be identified efficiently in the limit [8]. The argument we used was based on the existence of DTAs such that languages of these DTAs only contain examples of length exponential in the size of the DTA. Therefore, if we intend to identify the full class of DTAs, we will sometimes require an exponential amount of data in order to converge to the correct DTA. However, this argument no longer holds if the DTAs contain at most one timed component, known as a *clock*. In fact, in the same work, we even proved that the length of the shortest string in the symmetric difference of two one-clock DTAs (1-DTAs) can always be bounded by a polynomial. DTAs with this property are called *polynomially distinguishable*. This is a very important property for identification purposes because to identify a model is basically to distinguish models from each other based on examples. We have shown that this is a necessary requirement for efficient identification in the limit [8].

In this paper, we provide an algorithm that identifies 1-DTAs efficiently in the limit. The proof of convergence of this algorithm is based on the polynomial distinguishability of 1-DTAs. The algorithm is an important step in the direction of being able to identify real-time systems efficiently. To the best of our knowledge, our result is the first positive efficiency result for identifying timed automata. Moreover, we do not know of any other algorithm that identifies the complete structure of a timed automaton, including the clock resets.

The paper is organized as follows. We start with a brief introduction to 1-DTAs (Section 2), and a formal explanation of efficient identifiability in the limit (Section 3). We then describe our algorithm for identifying 1-DTAs (Section 4), and give the sketch of a proof that it converges to the correct 1-DTA efficiently in the limit(Section 5). We end our paper with some conclusions (Section 6).

2 Deterministic One-clock Timed Automata

A *timed automaton* (TA) [1] is an automaton that accepts (or generates) strings with event-time value pairs, called timed strings. A *timed string* τ over a finite set of symbols Σ is a sequence $(a_1, t_1)(a_2, t_2) \dots (a_n, t_n)$ of symbol-time value pairs $(a_i, t_i) \in \Sigma \times \mathbb{N}$.[1] Each time value t_i in a timed string represents the time passed until the occurrence of symbol a_i since the occurrence of the previous symbol a_{i-1}. We use $\tau_i = (a_1, t_1) \dots (a_i, t_i)$ to denote the prefix of length i of τ.

[1] Sometimes $\mathbb{R}$ is used as a time domain for TAs. However, for identification of TAs $\mathbb{N}$ is sufficient since, in practice, we always measure time using finite precision.

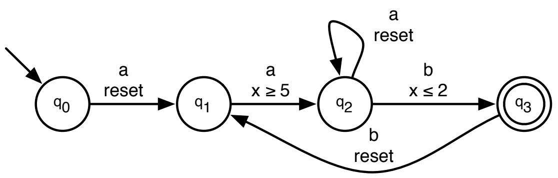

Fig. 1. A deterministic one-clock timed automaton. The start state q_0 is denoted by an arrow pointing to it from nowhere. The final state q_3 has two circles instead of one. The arrows represent transitions. The labels, clock guards, and clock resets are specified for every transition. When no guard is specified, the guard is always satisfied.

In TAs, timing conditions are added using a finite set X of *clocks* and one *clock guard* on every transition. For the purpose of this paper, i.e., identifying one-clock deterministic timed automata, we focus on the case where the TAs are *deterministic* and contain at most a *single clock*. Such a TA is called a *deterministic one-clock timed automata* (1-DTA). In TAs, *valuation* mappings $v : X \to \mathbb{N}$ are used to obtain the value of a clock $x \in X$. Since we use only a single clock x, we use v instead of $v(x)$ to denote the value of x. Every transition δ in a 1-TA contains a boolean value known as a *reset value*. When a transition δ fires (or occurs), and the reset value of δ is set to true, then the value of x is set to 0, i.e., $v := 0$. In this way, x is used to record the time since the occurrence of some specific event. Clock guards are then used to change the behavior of the 1-DTA depending on the value of x. A clock guard $g = c \leq x \leq c'$ is a boolean constraint on the value of x, where $c \in \mathbb{N}$ and $c' \in \mathbb{N} \cup \{\infty\}$ are constants.[2] A valuation v is said to *satisfy* a clock guard g, if whenever each occurrence of x in g is replaced by v the resulting statement is true. A deterministic one-clock timed automaton is defined as follows:

Definition 1. *A* deterministic one-clock timed automaton *(1-DTA) is a tuple $\mathcal{A} = \langle Q, x, \Sigma, \Delta, q_0, F\rangle$, where Q is a finite set of states, x is a clock, Σ is a finite set of symbols, Δ is a finite set of transitions, q_0 is the start state, and $F \subseteq Q$ is a set of final states.*

A transition $\delta \in \Delta$ is a tuple $\langle q, q', a, g, r\rangle$, where $q, q' \in Q$ are the source and target states, $a \in \Sigma$ is a symbol, called the transition label, g is a clock guard, and $r \in \{true, false\}$ is a reset value. Since $\mathcal{A}$ is deterministic, for every source state $q \in Q$, every label $a \in \Sigma$, and every valuation $v \in \mathbb{N}$, there exists at most a single transition $\langle q, q', a, g, r\rangle \in \Delta$ such that v satisfies g.

Figure 1 shows an example of a 1-DTA. Like a DFA, a 1-DTA moves from state to state by firing *state transitions* $\delta \in \Delta$ that connect these states. However, in addition, a 1-DTA is capable of remaining in the same state q for a while. While doing so, the valuation v of its clock x increases. We call this action of staying in

[2] Since we use the natural numbers to represent time open ($x < c$) and closed ($x \leq c$) one-clock timed automata are equivalent.

the same state a time transition. A *time transition* of t time units increases the valuation v of x by t, i.e. $v := v + t$. One can view a time transition as moving from one *timed state* (q, v) to another timed state $(q, v + t)$ while remaining in the same untimed state q. The length t of time transitions corresponds to the time values in a timed string. More precisely, the occurrence of a timed symbol (a, t) in a timed string means that before firing a transition δ labeled with a, the 1-DTA makes a time transition of t time units. After such a time transition, δ can only fire if its guard g is satisfied by $v + t$. The exact behavior of a 1-DTA is defined by what is called a *run* of a 1-DTA:

Definition 2. *A (finite)* run *of a 1-DTA* $\mathcal{A} = \langle Q, x, \Sigma, \Delta, q_0, F\rangle$ *over a timed string* $\tau = (a_1, t_1) \dots (a_n, t_n)$ *is a sequence of timed states and transitions* $(q_0, v_0) \xrightarrow{t_1} (q_0, v_0 + t_1) \xrightarrow{a_1} (q_1, v_1) \dots (q_{n-1}, v_{n-1}) \xrightarrow{t_n} (q_{n-1}, v_{n-1} + t_n) \xrightarrow{a_n} (q_n, v_n)$*, such that for all* $1 \leq i \leq n$ *there exists a transition* $\delta_i = \langle q_{i-1}, q_i, a_i, g, r\rangle \in \Delta$ *such that* $v_{i-1} + t_i$ *satisfies* g*,* $v_0 = 0$*, and* $v_i = 0$ *if* $r = true$*,* $v_i = v_{i-1} + t_i$ *otherwise.*

In a run the subsequence $(q_i, v_i + t) \xrightarrow{a_{i+1}} (q_{i+1}, v_{i+1})$ represents a state transition like in a finite automaton without time. When this occurs, the timed string τ is said to *fire* a transition δ with valuation $v_i + t$ to another timed state (q_{i+1}, v_{i+1}). The time transitions of a 1-DTA are represented by $(q_i, v_i) \xrightarrow{t_{i+1}} (q_i, v_i + t_{i+1})$. We say that a timed string τ *reaches* a timed state (q, v) in a TA $\mathcal{A}$ if there exist two time values $t \leq t'$ such that $(q, v') \xrightarrow{t'} (q, v'+t')$ occurs somewhere in the run of $\mathcal{A}$ over τ and $v = v' + t$. If a timed string reaches a timed state (q, v) in $\mathcal{A}$ for some valuation v, it also reaches the untimed state q in $\mathcal{A}$. A timed string *ends* in the last (timed) state it reaches, i.e., (q_n, v_n) or q_n. Notice that timed states in 1-DTAs are similar to the (untimed) states in DFAs: at any point during the run of a timed string τ, the state τ ends in depends on the current timed state and the remaining suffix of τ. A timed string τ is *accepted* by a 1-DTA $\mathcal{A}$ if τ ends in a final state, i.e., if $q_n \in F$. The set of all strings τ that are accepted by $\mathcal{A}$ is called the *language* $L(\mathcal{A})$ of $\mathcal{A}$.

Example 1. Consider the TA $\mathcal{A}$ of Fig. 1. The run of $\mathcal{A}$ over the timed string $\tau = (a, 5)(a, 6)(a, 2)(b, 2)$ is given by: $(q_0, 0) \xrightarrow{5} (q_0, 5) \xrightarrow{a} (q_1, 0) \xrightarrow{6} (q_1, 6) \xrightarrow{a} (q_2, 6) \xrightarrow{2} (q_2, 8) \xrightarrow{a} (q_2, 0) \xrightarrow{2} (q_2, 2) \xrightarrow{b} (q_3, 2)$. Since q_3 is a final state, it holds that $\tau \in L(\mathcal{A})$. Note that q_3 cannot be reached directly after reaching q_2 from q_1: the clock guard to q_2 is satisfied by a valuation v greater than 4, while the guard of the transition to q_3 requires v to be less than 3.

3 Efficient identification in the limit

An identification process tries to find (learn) a model that explains a set of observations (data). The ultimate goal of such a process is to find a model equivalent to the actual concept that was responsible for producing the observations, called the *target concept*. In our case, we try to find a 1-DTA model $\mathcal{A}$ that is equivalent to a target language L_t, i.e., $L(\mathcal{A}) = L_t$. If this is the case, we say that L_t is identified correctly. We try to find this model using *labeled data*: an *input sample* S for L_t is a pair of finite sets of positive examples $S_+ \subseteq L_t$ and negative examples $S_- \subseteq L_t^{\mathrm{C}} = \{\tau \mid \tau \not\in L_t\}$. We modify the non-strict set-inclusion operators for input samples such that they operate on the positive and negative examples separately, for example if $S = (S_+, S_-)$ and $S' = (S'_+, S'_-)$ then $S \subseteq S'$ means $S_+ \subseteq S'_+$ and $S_- \subseteq S'_-$. In addition, by $\tau \in S$ we mean $\tau \in S_+ \cup S_-$.

The input of our 1-DTA identification problem is a pair of finite sets. Unfortunately, as we already mentioned in the introduction, the 1-DTA identification problem is inapproximable. Because of this, we study the behavior of a 1-DTA identification process that is given more and more data. The framework for studying such a process is called *identification in the limit* [5]. In this framework, we do not regard the complexity for any possible input sample S, but for any possible target language L_t. For every target language L_t, there exist many possible input samples S. An identification process A is called *efficient in the limit* (from polynomial time and data) if for any target language L_t, A requires time polynomial in the size of any input sample S for L_t, and if the smallest input sample S such that A converges to L_t can be bounded by a polynomial in the size of L_t. The size of a target language L_t is defined as the size of a smallest model for L_t. Efficient identifiability in the limit can be proved by showing the existence of polynomial *characteristic sets* [7].

Definition 3. *A* characteristic set S_{cs} *of a target language L_t for an identification algorithm A is an input sample (S_+, S_-) for L_t such that:*

- *given S_{cs} as input, algorithm A identifies L_t, i.e., A returns an automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L_t$, and*
- *given any input sample $S' \supseteq S_{cs}$ as input, algorithm A still identifies L_t.*

Definition 4. *A class of automata C is* efficiently identifiable in the limit *if there exist two polynomials p and q, and an algorithm A such that:*

- *given an input sample of size $n = \sum_{\tau \in S} |\tau|$, A runs in time bounded by $p(n)$,*
- *and for every target language $L_t = L(\mathcal{A})$, $\mathcal{A} \in C$, there exists a characteristic set S_{cs} of L_t for A of size bounded by $q(|\mathcal{A}|)$.*

In previous work [8], we showed that DTAs in general are not efficiently identifiable. The argument we used was based on the fact that there exists DTAs such that languages of these DTAs only contain examples (strings) of length exponential in the size of the DTA. This was used to prove that DTAs are not *polynomially distinguishable*:

Definition 5. *A class C of automata is* polynomially distinguishable *if there exists a polynomial function p, such that for any $\mathcal{A}, \mathcal{A}' \in C$ with $L(\mathcal{A}) \neq L(\mathcal{A}')$, there exists a $\tau \in (L(\mathcal{A}) \cup L(\mathcal{A}')) \setminus (L(\mathcal{A}) \cap L(\mathcal{A}'))$, such that $|\tau| \leq p(|\mathcal{A}| + |\mathcal{A}'|)$.*

We have also shown that polynomial distinguishability is a necessary requirement for efficient identification [8]. Because of this, we cannot identify DTAs efficiently. In addition, we have proved that *1-DTAs are polynomially distinguishable*. Based on this result, we conjectured that 1-DTAs might be efficiently identifiable in the limit. In the following sections, we prove this conjecture by first describing an algorithm for identifying 1-DTAs. We then prove the convergence of this algorithm, i.e., that it identifies 1-DTAs efficiently in the limit. This proof is based on the polynomial distinguishability of 1-DTAs.

4 Identifying 1-DTAs efficiently in the limit

In this section, we describe our ID_1DTA algorithm for identifying 1-DTAs from an input sample S. The main value of this algorithm is that:

– given any input sample S, ID_1DTA returns in polynomial time a 1-DTA $\mathcal{A}$ that is consistent with S, i.e., such that $S_+ \subseteq L(\mathcal{A})$ and $S_- \subseteq L(\mathcal{A})^{\mathrm{C}}$,
– and if S contains a characteristic subsample S_{cs} for some target language L_t, then ID_1DTA returns a correct 1-DTA $\mathcal{A}$, i.e., such that $L(\mathcal{A}) = L_t$.

In other words, ID_1DTA identifies 1-DTAs efficiently in the limit. Note that, in a 1-DTA identification problem, the size of the 1-DTA is not predetermined. Hence, our algorithm has to identify the complete structure of a 1-DTA, including states, transitions, clock guards, and resets. Our algorithm identifies this structure one transition at a time: it starts with an empty 1-DTA $\mathcal{A}$, and whenever an identified transition requires more states or additional transitions, these will be added to $\mathcal{A}$. In this way, ID_1DTA builds the structure of $\mathcal{A}$ piece by piece. Since we claim that ID_1DTA identifies 1-DTAs efficiently, i.e., from polynomial time and data, we require that, for any input sample S for any target language L_t, the following four properties hold for this identification process:

Property 1. Identifying a single transition δ requires time polynomial in the size of S *(polynomial time per δ)*.
Property 2. The number of such transition identifications is polynomial in the size of S *(convergence in polynomial time)*.
Property 3. For every transition δ, there exists an input sample S_{cs} of size polynomial in the size of the smallest 1-DTA for L_t such that when included in S, S_{cs} guarantees that δ is identified correctly *(polynomial data per δ)*.
Property 4. The number of such correct transition identifications that are required to return a 1-DTA $\mathcal{A}$ with $L(\mathcal{A}) = L_t$ is polynomial in the size of the smallest 1-DTA for L_t *(convergence from polynomial data)* .

With these four properties in mind, we develop our ID_1DTA algorithm for the efficient identification of 1-DTAs. Pseudo code of this algorithm is shown in Algorithm 1. In this section, we use an illustrative example to show how this algorithm identifies a single transition, and to give some intuition why the algorithm satisfies these four properties. In the next section, we prove that our algorithm indeed satisfies these four properties and thus prove that it identifies 1-DTAs efficiently in the limit.

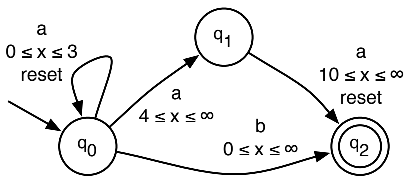

Fig. 2. A partially identified 1-DTA. The transitions from state q_0 have been completely identified. State q_1 only has one outgoing transition. State q_2 has none.

Example 2. Suppose that after having identified a few transitions, our algorithm has constructed the (incomplete) 1-DTA $\mathcal{A}$ from Figure 2. Furthermore, suppose that S contains the following timed strings: $\{(a, 4)(a, 6), (a, 5)(b, 6), (b, 3)(a, 2), (a, 4)(a, 1)(a, 3), (a, 4)(a, 2)(a, 2)(b, 3)\} \subseteq S_+$ and $\{(a, 3)(a, 10), (a, 4)(a, 2)(a, 2), (a, 4)(a, 3)(a, 2)(b, 3), (a, 5)(a, 3)\} \subseteq S_-$. Our algorithm has to identify a new transition δ of $\mathcal{A}$ using information from S. There are a few possible identifiable transitions: state q_1 does not yet contain any transitions with label b, or with label a and valuations smaller than 9, and state q_2 does not yet contain any transitions at all. Our algorithm first makes a choice which transition to identify, i.e., it selects the source state, label, and valuations for a new transition. Then our algorithm actually identifies the transition, i.e., it uses S in order to determine the target state, clock guard, and reset of the transition.

As can be seen from the example, the first problem our algorithm has to deal with is to determine which transition to identify. Our algorithm makes this decision using a fixed predetermined order (independent of the input sample). The order used by our algorithm is very straightforward: first a state q is selected in the order of identification (first identified first), second a transition label l is selected in alphabetic order, and third the highest possible upper bound c' for a clock guard in this state-label combination is chosen. This fixed order makes it easier to prove the existence of characteristic sets (satisfying property 3). In our example, our algorithm will try to identify a transition $\delta = \langle q, q', l, c \leq x \leq c', r\rangle$, where $q = q_1$, $l = a$, and $c' = 9$ (since there exists a transition with a clock guard that is satisfied by a valuation $v = 10$) are all fixed. Thus, our algorithm only needs to identify: (i) the target state q', (ii) the lower bound of the clock guard c, and (iii) the clock reset r.

Note that fixing q, a, and c' in this way does not influence which transitions will be identified by our algorithm. Since we need to identify a transition with these values anyway, it only influences the order in which these transitions are identified. We now show how our algorithm identifies c, r, and q'.

The lower bound c. Our algorithm first identifies the lower bound c of the clock guard g of δ. The smallest possible lower bound for g is the smallest reachable valuation $v_{\min}$ in q (q_1 in the example). This valuation $v_{\min}$ is equal to the smallest lower bound of a transition with q as target. In the example, $v_{\min}$ is

Algorithm 1 Efficiently learning 1-DTAs from polynomial data: ID_1DTA

```
Require: An input sample $S = (S_+, S_-)$ for a language $L_t$, with alphabet $\Sigma$
Ensure: $\mathcal{A}$ is a 1-DTA consistent with $S$, i.e. $S_+ \subseteq L(\mathcal{A})$ and $S_- \subseteq L(\mathcal{A})^C$, in addition,
  if it holds that $S_{cs} \subseteq S$, then $L(\mathcal{A}) = L_t$
  $\mathcal{A} := \langle Q = \{q_0\}, x, \Sigma, \Delta = \emptyset, q_0, F = \emptyset \rangle$
  if $S_+$ contains the empty timed string $\lambda$ then set $F := \{q_0\}$
  while there exist a reachable timed state $(q, v)$ and a symbol $a$ for which there
  exists no transition $\langle q, q', a, g, r \rangle \in \Delta$ such that $v$ satisfies $g$ do
    for all states $q \in Q$ and symbols $a \in \Sigma$ do
      $v_{\min} := \min\{v \mid (q, v) \text{ is reachable }\}$
      $c' := \max\{v \mid \neg\exists \langle q, q', a, g, r \rangle \in \Delta \text{ such that } v \text{ satisfies } g\}$
      while $v_{\min} \leq c'$ do
        create a new transition $\delta := \langle q, q' := 0, a, g := v_{\min} \leq x \leq c', r \rangle$, add $\delta$ to $\Delta$
        $V := \{v \mid \exists \tau \in S : \tau \text{ fires } \delta \text{ with valuation } v\}$
        $r :=$ true and $c_1 := \mathsf{lower\_bound}(\delta, V \cup \{v_{\min}\}, \mathcal{A}, S)$
        $r :=$ false and $c_2 := \mathsf{lower\_bound}(\delta, V \cup \{v_{\min}\}, \mathcal{A}, S)$
        if $c_1 \leq c_2$ then set $r :=$ true and $g := c_1 \leq x \leq c'$
        else set $r :=$ false and $g := c_2 \leq x \leq c'$
        for every state $q'' \in Q$ (first identified first) do
          $q' := q''$
          if $\mathsf{consistent}(\mathcal{A}, S)$ is true then break else $q' := 0$
        end for
        if $q' = 0$ then
          create a new state $q''$, set $q' := q''$, and add $q'$ to $Q$
          if $\exists \tau \in S_+$ such that $\tau$ ends in $q'$ then set $F := F \cup \{q'\}$
        end if
        $c' := \min\{v \mid v \text{ satisfies } g\} - 1$
      end while
    end for
  end while
```

4. Thus, the lower bound c has to be a value with the set $\{c \mid v_{\min} \leq c \leq c'\}$. One approach for finding c would be to try all possible values from this set and pick the best one. However, since time values are encoded in binary in the input sample S, iterating over such a set is exponential in the size of these time values, i.e., it is exponential in the size of S (contradicting property 1). This is why our algorithm only tries those time values that are actually used by timed strings from S. We determine these in the following way. We first set the lower bound of g to be $v_{\min}$. There are now examples in S that fire δ. The set of valuations V that these examples use to fire δ are all possible lower bounds for g, i.e., $V := \{v \mid \exists \tau \in S : \tau \text{ fires } \delta \text{ with valuation } v\}$. In our example, we have that $\{(a,4)(a,1)(a,3)\} \subseteq S_+$ and $\{(a,5)(a,3),(a,4)(a,2)(a,2)\} \subseteq S_-$. In this case, $V = \{4+1 = 5, 5+3 = 8, 4+2 = 6\}$. Since for every time value in V there exists at least one timed string in S for every such time value, iterating over this set is polynomial in the size of S (satisfying property 1).

From the set $V \cup \{v_{\min}\}$ our algorithm selects the smallest possible consistent lower bound. A lower bound is consistent if the 1-DTA resulting from identifying this bound is consistent with the input sample S. A 1-DTA $\mathcal{A}$ is called *consistent* if S contains no positive example that inevitably ends in the same state as a negative example, i.e., if the result $\mathcal{A}$ can still be such that $S_+ \in L(\mathcal{A})$ and $S_- \in L(\mathcal{A})^{\mathrm{C}}$ (satisfying property 4). Whether $\mathcal{A}$ is consistent with S is checked by testing whether there exist no two timed strings $\tau \in S_+$ and $\tau' \in S_-$ that reach the same timed state (possibly after making a partial time transition) and afterwards their suffixes are identical. We use consistent to denote this check. This check can clearly be done in polynomial time (satisfying property 1). Our algorithm finds the smallest consistent lower bound by trying every possible lower bound $c \in V \cup \{v_{\min}\}$, and testing whether the result is consistent. We use lower_bound to denote this routine. This routine ensures that at least one timed string from S will fire δ, and hence that our algorithm only identifies a polynomial amount of transitions (satisfying property 2). In our example, setting c to 5 makes $\mathcal{A}$ inconsistent since now both $(a,4)(a,1)(a,3)$ and $(a,4)(a,2)(a,2)$ reach $(q',6)$, where q' is any possible target for δ, and afterwards they have the same suffix $(a,2)$. However, setting c to 6 does not make $\mathcal{A}$ inconsistent. Since 6 is the smallest value in $V \cup \{v_{\min}\}$ greater than 5, $c = 6$ is the smallest consistent lower bound for g.

Our main reason for selecting the smallest consistent lower bound for g is that this selection can be used to force our algorithm to make the correct identification (required by property 3). Suppose that if our algorithm identifies c^*, and if all other identifications are correct, then the result $\mathcal{A}$ will be such that $L(\mathcal{A}) = L_t$. Hence, our algorithm should identify c^*. In this case, there always exist examples that result in an inconsistency when our algorithm selects any valuation smaller than c^*. The reason is that an example that fires δ with valuation $c^* - 1$ should actually fire a different transition, to a different state, or with a different reset value. Hence, the languages after firing these transitions are different. Therefore, there would exist two timed strings $\tau \in L_t$ and $\tau' \in L_t^{\mathrm{C}}$ (that can be included in S) that have identical suffixes after firing δ with valuations c^* and $c^* - 1$ respectively. Moreover, any pair of string that fire δ with valuations greater or equal to c^* cannot lead to an inconsistency since their languages after firing δ are the same.

The reset r. After having identified the lower bound c of the clock guard g of δ, our algorithm needs to identify the reset r of δ. One may notice that the identification of g depends on whether δ contains a clock reset or not: the value of r determines the valuations that are reached by timed strings after firing δ (the clock can be reset to 0), hence this value determines whether $\mathcal{A}$ is consistent after trying a particular lower bound for g. In our example, $(a,4)(a,1)(a,3)$ and $(a,4)(a,1)(a,2)$ reach $(q',1)$ and $(q',0)$ respectively before their suffixes are identical if $r = \text{true}$. Because of this, our algorithm identifies the clock reset r of δ at the same time it identifies its clock guard g. The method it uses to identify r is very simple: first set $r = \text{true}$ and then find the smallest consistent lower bound c_1 for g, then set $r = \text{false}$ and find another such lower bound c_2 for g. The value of r is set to true if and only if the lower bound found with this setting is smaller than the other one, i.e., iff $c_1 \leq c_2$. There always exist timed strings that ensure that the smallest consistent lower bound for g when the clock reset is set incorrectly is larger than when it is set correctly (satisfying property 3). In our example the timed strings that ensure this are $(a,4)(a,2)(a,2)(b,3) \in S_+$ and $(a,4)(a,3)(a,2)(b,3) \in S_-$. Because these examples reach the same valuations in state q' only if the clock is reset, they create an inconsistency when r is set to true. In general, such strings always exists since the difference of 1 time value is sufficient for such an inconsistency: a difference of 1 time value can always be the difference between later satisfying and not satisfying some clock guard.[3]

The target state q'. Having identified both the clock guard and the reset of δ, our algorithm still needs to identify the target state q' of δ. Since we need to make sure that our algorithm is capable of identifying any possible transition (required by property 3), we need to try all possible settings for q', and in order to make it easier to prove the existence of a characteristic set (required by property 3), we do so in a fixed order. The order our algorithm uses is the order in which our algorithm identified the states, i.e., first q_0, then the first additional identified state, then the second, and so on. The target state for δ is set to be the first consistent target state in this order. In our example, we just try state q_0, then state q_1, and finally state q_3. When none of the currently identified states result in a consistent 1-DTA $\mathcal{A}$, the target is set to be a new state. This new state is set to be a final state only if there exists a timed string in S_+ that ends in it. It should be clear that since the languages after reaching different states are different, there always exist timed strings that ensure that our algorithm identifies the correct target (satisfying property 3). In our example, there exist no timed strings that make $\mathcal{A}$ inconsistent when our algorithm tries the first state (state q_0), and hence our algorithm identifies a transition $\langle q_1, q_0, a, 6 \leq x \leq 9, \text{false}\rangle$.

This completes the identification of δ and (possibly) q'. This identification of a single transition δ essentially describes the main part of our algorithm (see Algorithm 1). However, we still have to explain how our algorithm iterates over the transitions it identifies. The algorithm consists of a main loop that iterates in a fixed order over the possible source states and labels for new transitions. For every combination of a source state q and a label a, our algorithm first sets two values: $v_{\min}$ and c'. The first is the smallest reachable valuation in q. The second is the fixed upper bound of the delay guard of a new transition. Because our model is deterministic, this is set to be the largest reachable valuation for which there exists no transition with q as source state and a as label. After identifying a transition δ with these values, our algorithm updates c' to be one less than the lower bound of the clock guard of δ. If c is still greater than $v_{\min}$, there are still transitions to identify for state q and label a. Thus, our algorithm iterates and continues this iteration until c' is strictly less than $v_{\min}$. Our main reason for adding this additional iteration is that it makes it easier to prove the

[3] This holds unless the clock guard can only be satisfied by a single valuation v, i.e., unless $g = c \leq x \leq c$. However, in this case any setting for r is correct since both can lead to results such that $L(\mathcal{A}) - L_t$.

convergence of our algorithm (property 4). The main loop of our algorithm continuously identifies new transitions and possibly new target states until there are no more new transitions to identify, i.e., until there exists a transition for every reachable timed state in $\mathcal{A}$. This is necessary because identifying a transition δ can create new identifiable transitions. This happens when the smallest reachable valuation $v_{\min}$ in some state is decreased, or when a new state is identified, by the identification of δ.

5 Properties of the algorithm

We described an algorithm for the identification of 1-DTAs. We claim now, and argued in the previous section, that the algorithm converges efficiently to a correct 1-DTA, i.e., that it identifies 1-DTAs efficiently in the limit. In this section, we show that the four properties mentioned in the previous section hold. The combination of these properties satisfies all the constraints required for efficient identification in the limit (Definition 4), and hence shows that 1-DTAs are efficiently identifiable (Theorem 1). We only give sketches of proofs due to space restrictions.

Proposition 1. *ID_1DTA is a polynomial time algorithm (properties* 1 *and* 2*).*

Proof. (sketch) Identifying a single transition can be done in polynomial time as argued in the previous section. In addition, every transition is guaranteed to be fired by at least one timed string from S. Hence, our algorithm will stop after identifying a polynomial amount of transitions. The proposition follows.

Lemma 1. *There exist polynomial characteristic sets of the transitions of 1-DTAs for ID_1DTA (property* 3*).*

Proof. (sketch) As shown by the example in the previous section, we can always find a polynomial amount of timed strings such that our algorithm identifies the correct transition δ. In addition, we can bound each of these strings by a polynomial since 1-DTAs are polynomially distinguishable. Moreover, because the order in which our algorithm identifies transitions is independent of S, it is impossible to add additional examples to S such that our algorithm will no longer identify δ. This proves the lemma.

Lemma 2. *ID_1DTA converges after identifying a polynomial amount of transitions (property* 4*).*

Proof. (sketch) Since 1-DTAs are polynomially distinguishable, any state can be reached by timed strings τ of polynomial length. Hence, the main loop will be run at most $|\tau|$ times before the smallest reachable valuation in any state of $\mathcal{A}_t$ can be reached. Once the smallest reachable valuation can be reached, every transition can be identified. In a single run of the main loop, at most $|\mathcal{A}_t|$ new correct transitions can be identified. Thus, our algorithm identifies at most $|\tau| \times |\mathcal{A}_t|$ transitions before every transition can be identified. By Lemma 1, our algorithm is capable of identifying any possible transition correctly. Hence, every transition of $\mathcal{A}_t$ can be identified correctly after identifying $|\tau| \times |\mathcal{A}_t|$ transitions.

Theorem 1. *1-DTAs are efficiently identifiable in the limit.*

Proof. By Proposition 1 and Lemma 2, if all the examples from Lemma 1 are included in S, our algorithm returns a 1-DTA $\mathcal{A}$ such that $L(A) = L_t$ in polynomial time and from polynomial data. We conclude that Algorithm 1 identifies 1-DTAs efficiently in the limit.

6 Conclusions

In this paper we described an algorithm that identifies 1-DTAs efficient in the limit. This algorithm is an important step in the direction of being able to identify timed systems efficiently. To the best of our knowledge, our result is the first positive efficiency result for identifying timed automata. The fact that 1-DTAs can be identified efficiently has important consequences for anyone interested in identifying timed systems. Most importantly, it is a reason to model timed systems with 1-DTAs. However, when 1-DTAs are too restrictive, our result is still useful because identification algorithms for other DTAs could identify the class of 1-DTAs efficiently. This is a desirable property since 1-DTAs are efficiently identifiable. For instance, in related work, a query learning algorithm is described for identifying event recording automata (ERAs) [9]. However, the used query learning algorithm requires an exponential amount of queries (data). It would be interesting to either adapt the timed query learning algorithm to the class of 1-DTAs, or to show that the algorithm does in fact identify 1-DTAs efficiently. This could result in an efficient query learning algorithm for timed systems.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126** (1994) 183–235
2. Larsen, K.G., Petterson, P., Yi, W.: Uppaal in a nutshell. International journal on software tools for technology transfer **1**(1-2) (1997) 134–152
3. Gold, E.M.: Complexity of automaton identification from given data. Information and Control **37**(3) (1978) 302–320
4. Pitt, L., Warmuth, M.: The minimum consistent DFA problem cannot be approximated within any polynomial. In: Annual ACM Symposium on Theory of Computing, ACM (1989) 421–432
5. Gold, E.M.: Language identification in the limit. Information and Control **10**(5) (1967) 447–474
6. Oncina, J., Garcia, P.: Inferring regular languages in polynomial update time. In: Pattern Recognition and Image Analysis. Volume 1 of Series in Machine Perception and Artificial Intelligence. World Scientific (1992) 49–61
7. de la Higuera, C.: Characteristic sets for polynomial grammatical inference. Machine Learning **27**(2) (1997) 125–138
8. Verwer, S., de Weerdt, M., Witteveen, C.: Polynomial distinguishability of timed automata. In: ICGI. Volume 5278 of LNAI, Springer (2008) 238–251
9. Grinchtein, O., Jonsson, B., Petterson, P.: Inference of event-recording automata using timed decision trees. In CONCUR. Volume 4137 of LNCS, Springer (2006) 435–449