The efficiency of identifying timed automata and the power of clocks

Sicco Verwer[a,b,1,*], Mathijs de Weerdt[b], Cees Witteveen[b]

[a] *Eindhoven University of Technology, Department of Mathematics and Computer Science (EWI), P.O. 513, 5600 MB, Eindhoven, the Netherlands*
[b] *Delft University of Technology, Department of Engineering, Mathematics and Computer Science, Mekelweg 4, 2628 CD, Delft, the Netherlands*

Abstract

We develop theory on the efficiency of identifying (learning) timed automata. In particular, we show that: i) deterministic timed automata cannot be identified efficiently in the limit from labeled data, and ii) that one-clock deterministic timed automata can be identified efficiently in the limit from labeled data. We prove these results based on the distinguishability of these classes of timed automata. More specifically, we prove that the languages of deterministic timed automata cannot, and that one-clock deterministic timed automata can be distinguished from each other using strings in length bounded by a polynomial. In addition, we provide an algorithm that identifies one-clock deterministic timed automata efficiently from labeled data.

Our results have interesting consequences for the power of clocks that are interesting also out of the scope of the identification problem.

Keywords: deterministic timed automata, identification in the limit, complexity

1. Introduction

Timed automata [1] (TAs) are finite state models that represent timed events using an *explicit* notion of time, i.e., using numbers. They can be used to model and reason about real-time systems [14]. In practice, however, the knowledge required to completely specify a TA model is often insufficient. An alternative is to try to identify (induce) the specification of a TA from observations. This approach is also known as *inductive inference* (or *identification of models*). The idea behind identification is that it is often easier to find examples of the behavior of a real-time system than to specify the system in a

*Corresponding author

Email addresses: s.verwer@tue.nl (Sicco Verwer), m.m.deweerdt@tudelft.nl (Mathijs de Weerdt), c.witteveen@tudelft.nl (Cees Witteveen)

URL: http://wwwis.win.tue.nl/∼sverwer (Sicco Verwer)

[1] The main part of this research was performed when the first author was a PhD student at Delft University of Technology. It has been supported and funded by the Dutch Ministry of Economical Affairs under the SENTER program.

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.

Often these examples are obtained using sensors. This results in a time series of system states: every millisecond the state of (or event occurring in) the system is measured and recorded. Every such time series is then given a label, specifying for instance whether it displays 'good' or 'bad' system behavior. Our goal is to then to identify (learn) a TA model from these labeled examples (also called from informants). From this timed data, we could have opted to identify a model that models time *implicitly*. Examples of such models are the *deterministic finite state automaton* (DFA), see e.g. [18], and the *hidden Markov model* (HMM) [17]. A common way to identify such models is to first transform the input by *sampling* the timed data, e.g., by generating a new symbol every second. When a good sampling frequency is used, a DFA or HMM can be identified that characterizes the real-time system in the same way that a TA model would.

The main problem of modeling time implicitly is that it results in an exponential blow-up of the model size: numbers use a binary representation of time while states use a unary representation of time. Hence both the transformed input and the resulting DFA or HMM are exponentially larger than their timed counterparts. Because of this, we believe that if the data contains timed properties, i.e., if it can be modeled efficiently using a TA, then it is less efficient and much more difficult to identify a non-timed model correctly from this data. In previous work [19], we have experimentally compared the performance of a simple TA identification algorithm with a sampling approach combined with the evidence driven state merging method (EDSM) [12] for identifying DFAs. While EDSM has been the most successful method for identifying DFAs from untimed data, on timed data it performed worse than our algorithm.

In contrast to DFAs and HMMs, however, until now the identification problem for TAs has not received a lot of attention from the research community. We are only aware of studies on the related problem of the identification of event-recording automata (ERAs) [2]. It has for example been shown that ERAs are identifiable in the query learning framework [9]. However, the proposed query learning algorithm requires an exponential amount of queries, and hence is data inefficient. We would like our identification process to be efficient. This is difficult because the identification problem for DFAs is already NP-complete [7]. This property easily generalizes to the problem of identifying a TA (by setting all time values to 0). Thus, unless $\mathsf{P} = \mathsf{NP}$, a TA cannot be identified efficiently. Even more troublesome is the fact that the DFA identification problem cannot even be approximated within any polynomial [16]. Hence (since this also generalizes), the TA identification problem is also inapproximable.

These two facts make the prospects of finding an efficient identification process for TAs look very bleak. However, both of these results rely on there being a fixed input for the identification problem (encoding a hard problem). 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 (no longer encoding the hard problem). The framework that studies this behavior is *identification in the limit* [6]. This framework can be summarized as follows. A class of languages (for example the languages, modeled by DFAs) is called *identifiable in the limit* if there exists an identification algorithm that at some point (in the limit) converges to any language from this class when given an increasing amount of examples from this language. If this

identification algorithm requires polynomial time in the size of these examples, and if a polynomial amount of polynomially sized examples in the size of the smallest model for these languages is sufficient for convergence, this class of languages is said to be *identifiable in the limit from polynomial time and data*, or simply *efficiently identifiable* (see [4]).

The identifiability of many interesting models (or languages) have been studied within the framework of learning in the limit. In particular, the class of all DFAs have been shown to be efficiently identifiable using a state merging method [15]. Also, it has been shown that the class of *non-deterministic finite automata* (NFAs) are not efficiently identifiable in the limit [4]. This again generalizes to the problem of identifying a non-deterministic TA (by setting all time values to 0). Therefore, we consider the identification problem for *deterministic timed automata* (DTAs). Our goal is to determine exactly when and how DTAs are efficiently identifiable in the limit.

Our proofs mainly build upon the property of *polynomial distinguishability*. This property states that the length of a shortest string in the symmetric difference of any two languages should be bounded by a polynomial in the sizes of the shortest representations (automata) for these languages. For a language class without this property, it is impossible to distinguish all languages from each other using a polynomial amount of data. Hence, an identification algorithm cannot always converge when given a polynomial amount of data as input. In other words, the language class cannot be identified efficiently. We prove many important results using this property. Our main results are:

1. Polynomial distinguishability is a necessary condition for efficient identification in the limit (Lemma 9).
2. DTAs with two or more clocks are not polynomially distinguishable (Theorem 1 and 2).
3. DTAs with one clock (1-DTAs) are polynomially distinguishable (Theorem 5).
4. 1-DTAs are efficiently identifiable from labeled examples using our ID_1-DTA algorithm (Algorithm 1 and Theorem 6).

We prove the first two results in Section 4. These results show that DTAs cannot be identified efficiently in general. Although these results are negative, they do show they importance of polynomial distinguishability. In order to find DTAs that can be identified efficiently, we thus looked for a class of DTAs that is polynomially distinguishable. In Section 5 we prove that the class of 1-DTAs is such a class. Our proof is based on an important lemma regarding the modeling power of 1-DTAs (Lemma 18). This has interesting consequences outside the scope of the DTA identification problem. For instance, it proves membership in coNP for the equivalence problem for 1-DTAs (Corollary 21). In addition, it has consequences for the power of clocks in general. We give an overview of these consequences in Section 7.

In Section 6.1, we describe our algorithm for identifying 1-DTAs efficiently from labeled data. In Section 6.2, we use the results of Section 4 in order to prove our fourth main result: the fact that 1-DTAs can be identified efficiently. This fact is surprising because the standard methods of transforming a DTA into a DFA (sampling or the region construction [1]) results in a DFA that is exponentially larger than an original 1-DTA. This blow-up is due to the fact that time is represented in binary in a 1-DTA, and in unary (using states) in a DFA. In other words, 1-DTAs are exponentially more compact than DFAs, but still efficiently identifiable.

We end this paper with a summary and discussion regarding the obtained results (Section 8). In general, our results show that identifying a 1-DTA from timed data is more efficient than identifying an equivalent DFA. Furthermore, the results show that anyone who needs to identify a DTA with two or more clocks should either be satisfied with sometimes requiring an exponential amount of data, or he or she has to find some other method to deal with this problem, for instance by identifying a subclass of DTAs (such as 1-DTAs). Before providing our results and proofs, we first briefly review timed automata and efficient identification in the limit (Sections 2 and Section 3).

2. Timed automata

We assume the reader to be familiar with the theory of languages and automata, see, e.g., [18]. A *timed automaton* [1] is an automaton that accepts (or generates) strings of symbols (events) paired with time values, known as timed strings. In the setting of TA identification, we always measure time using finite precision, e.g., milliseconds. Therefore, we represent time values using the natural numbers $\mathbb{N}$ (instead of the nonnegative reals $\mathbb{R}_+$).[2] We define a finite *timed string* τ over a finite set of symbols Σ as a sequence $(a_1, t_1)(a_2, t_2)\dots(a_n, t_n)$ of symbol-time value pairs $(a_i, t_i) \in \Sigma \times \mathbb{N}$. We use τ_i to denote the length i prefix of τ, i.e., $\tau_i = (a_1, t_1)\dots(a_i, t_i)$. A time value t_i in τ represents the time until the occurrence of symbol a_i as measured from the occurrence of the previous symbol a_{i-1}. We define the length of a timed string τ, denoted $|\tau|$, as the number symbol occurrences in τ, i.e., $|\tau| = n$. Thus, length of a timed string is defined as the length in *bits*, not the length in *time*.[3]

A TA models a language over timed strings. It can model all of the non-timed language conditions that are models using non-timed automata (such as DFAs). In addition, however, a TA can model *timing conditions* that constrain the amount of time that can elapse between two event occurrences. For instance, a TA can specify that it only accepts timed strings that consist of events that occur within ten milliseconds of each other. In this case, it accepts $(a, 5)(a, 5)$ and rejects $(a, 5)(a, 11)$, because of the time that elapses between the first and second events. These time constraints can exist between any pair of events. For example, a TA can accept only those timed strings such that the time that elapses between the fifth and the tenth a event is greater than 1 second. In this way, TAs can for instance be used to model deadlines in real-time systems. The time constraints in TAs can be combined (using $\vee$ and $\wedge$ operators), resulting in complicated timed language conditions. These timing conditions are modeled using a finite set of clocks X, and for every transition a finite set of clock resets R and a clock guard g. We now explain these three notions in turn.

A *clock* $x \in X$ is an object associated with a value that increases over time, *synchronously* with all other clocks. A *valuation* v is a mapping from X to $\mathbb{N}$, returning the value of a clock $x \in X$. We can add or subtract constants or other valuations to or

[2] All the results in this paper also apply to TAs that use the nonnegative reals with finite precision.

[3] This makes sense because our main goal is to obtain an efficient identification algorithm for TAs. Such an algorithm should be efficient in the size of an efficient encoding of the input, i.e., using binary notation for time values. We do not consider the actual time values (in binary notation) because their influence on the length of a timed string is negligible compared to the influence of the number of symbols (in unary notation).

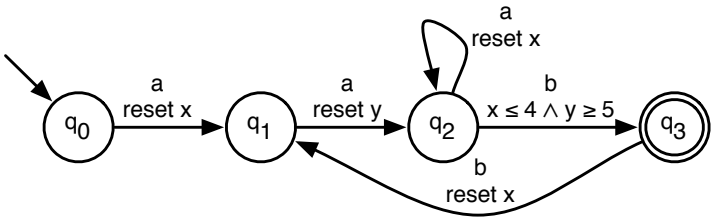

Figure 1: A timed automaton. The labels, clock guards, and clock resets are specified for every transition. When no guard is specified it means that the guard is always satisfied.

from a valuation: if $v = v' + t$ then $\forall x \in X : v(x) = v'(x) + t$, and if $v = v' + v''$ then $\forall x \in X : v(x) = v'(x) + v''(x)$.

Every transition δ in a TA is associated with a set of clocks R, called the *clock resets*. When such a transition δ occurs (fires), the values of all the clocks in R are set to 0, i.e., $\forall x \in R : v(x) := 0$. The values of all other clocks remain the same. We say that δ *resets* x if $x \in R$. In this way, clocks are used to record the time since the occurrence of some specific event. The clock guards are then used to change the behavior of the TA depending on the value of clocks.

A *clock guard* g is a boolean constraint defined by the grammar $g := x \leq c \mid x \geq c \mid g \wedge g$, where $x \in X$ is a clock and $c \in \mathbb{N}$ is a constant.[4] A valuation v is said to *satisfy* a clock guard g, denoted $v \in g$, if for each clock $x \in X$, each occurrence of x in g is replaced by $v(x)$, and the resulting constraint is satisfied.

A timed automaton is defined as follows:

Definition 1. *(TA) A* timed automaton *(TA) is a 6-tuple* $\mathcal{A} = (Q, X, \Sigma, \Delta, q_0, F)$*, where* Q *is a finite set of states,* X *is a finite set of clocks,* 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* $R \subseteq X$ *is the set of clock resets.*

A transition $\langle q, q', a, gR \rangle$ in a TA is interpreted as follows: whenever the TA is in state q, the next symbol is a, and the clock guard g is satisfied, then the TA can move to state q', resetting the clock associated with R. We give a small example:

Example 1. *The TA of Figure 1 accepts* $(a,1)(a,2)(a,3)(b,4)$ *and rejects* $(a,1)(a,2)(a,1)(b,2)$*. The computation of the first timed string starts in the start state* q_0*. There it waits 1 time step before generating* $(a,1)$*, it fires the transition to state* q_1*. By firing this transition, clock* x *has been reset to* 0*. In state* q_1*, it waits* 2 *time steps before generating* $(a,2)$*. It fires the transition to state* q_2 *and resets clock* y*. The values of* x *and* y *are now* 2 *and* 0*, respectively. It then waits* 3 *time steps before generating* $(a,3)$*, and firing the transition to state* q_2*, resetting* x *again. The values of* x *and* y *are now* 0 *and* 3*, respectively. After*

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

waiting 4 *time steps, it generates* $(b, 4)$*. The value of* x *now equals* 4*, and the value of* y *equals* 7*. These values satisfy the clock guard* $x \leq 4 \wedge y \geq 5$*, and hence the transition to state* q_3 *is fired, ending in a final state.*

The computation of the second timed string does not end in a final state because the value of x *equals* 2*, and the value of* y *equals* 3*, which does not satisfy the clock guard on the transition to state* q_3*. Notice that the clock guard of the transition to state* q_3 *cannot be satisfied directly after entering state* q_2 *from state* q_1*: the value of* x *is greater or equal to the value of* y *and the guard requires it to be less then the value of* y*. The fact that TAs can model such behavior is the main reason why identifying TAs can be very difficult.*

The computation of a TA is defined formally as follows:

Definition 2. *(TA computation) A finite* computation *of a TA* $\mathcal{A} = (Q, X, \Sigma, \Delta, q_0, F)$ *over a (finite) timed string* $\tau = (a_1, t_1) \dots (a_n, t_n)$ *is a finite sequence*

$$(q_0, v_0) \xrightarrow{t_1} (q_0, v_0 + t_1) \xrightarrow{a_1} (q_1, v_1) \xrightarrow{t_2} \dots$$
$$\dots \xrightarrow{a_{n-1}} (q_{n-1}, v_{n-1}) \xrightarrow{t_n} (q_{n-1}, v_{n-1} + t_n) \xrightarrow{a_n} (q_n, v_n)$$

such that for all $1 \leq i \leq n : q_i \in Q$*, a transition* $\delta = \langle q_{i-1}, q_i, a_i, g_i, R_i \rangle \in \Delta$ *exists such that* $v_{i-1} + t_i \in g_i$*, and for all* $x \in X : v_0(x) = 0$*, and* $v_i(x) := 0$ *if* $x \in R_i$*,* $v_i(x) := v_{i-1}(x) + t_i$ *otherwise.*

We call a pair (q, v) of a state q and a valuation v a *timed state*. In a computation the subsequence $(q_i, v_i + t) \xrightarrow{a_{i+1}} (q_{i+1}, v_{i+1})$ represents a state transition analogous to a transition in a conventional non-timed automaton. In addition to these, a TA performs time transitions represented by $(q_i, v_i) \xrightarrow{t_{i+1}} (q_i, v_i + t_{i+1})$. A *time transition* of t time units increases the value of all clocks of the TA by t. One can view such a transition as moving from one timed state (q, v) to another timed state $(q, v + t)$ while remaining in the same untimed state q. 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 computation 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). A timed string τ is *accepted* by a TA $\mathcal{A}$ if τ ends in a final state $q_f \in F$. The set of all strings τ that are accepted by $\mathcal{A}$ is called the *language* $L(\mathcal{A})$ of $\mathcal{A}$.

In this paper we only consider deterministic timed automata. A TA $\mathcal{A}$ is called *deterministic* (DTA) if for each possible timed string τ there exists at most one computation of $\mathcal{A}$ over τ. We only consider DTAs because the class of non-timed non-deterministic automata are already not efficiently identifiable in the limit [4]. In addition, without loss of generality, we assume these DTAs to be *complete*, i.e., for every state q, every symbol a, and every valuation v, there exists a transition $\delta = \langle q, q', a, g, R \rangle$ such that g is satisfied by v. Any non-complete DTA can be transformed into a complete DTA by adding a garbage state.

Although DTAs are very powerful models, any DTA $\mathcal{A}$ can be transformed into an equivalent DFA, recognizing a non-timed language that is equivalent to $L(\mathcal{A})$. This can for example be done by sampling[5] the timed strings: every timed symbol (a, t) is replaced

[5] When time is modeled using $\mathbb{R}$, the *region construction method* [1] can be used.

by an a, followed by t special symbols that denote a time increase. For every DTA $\mathcal{A}$, there exists a DFA that accepts the language created by sampling all of the timed string in $L(\mathcal{A})$. However, because this DFA uses a unary representation of time, while the original DTA uses a binary representation of time, such a sampling transformation results in an exponential blow-up of the model size.

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 language that was used to produce the observations, called the *target language*. In our case, we try to find a DTA model $\mathcal{A}$ that is equivalent to a timed 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* (also called *supervised learning*): an *input sample* S is a pair of finite sets of positive examples $S_+ \subseteq L_t$ and negative examples $S_- \subseteq L_t^{\mathrm{C}} = \{\tau \mid \tau \notin L_t\}$. With slight abuse of notation, 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 $S_- \subseteq S'_-$.

An identification process can be called *efficient in the limit* (from polynomial time and data) if the time and data it requires for convergence are both polynomial in the size of the target concept. Efficient identifiability in the limit can be shown by proving the existence of polynomial *characteristic sets* [4]. In the case of DTAs:

Definition 3. *(characteristic sets for DTAs) A* characteristic set S_{cs} *of a target DTA language L_t for an identification algorithm A is an input sample $\{S_+ \in L_t, S_- \in L_t^c\}$ such that:*

1. *given S_{cs} as input, algorithm A identifies L_t correctly, i.e., A returns a DTA $\mathcal{A}$ such that $L(\mathcal{A}) = L_t$;*
2. *given any input sample $S' \supseteq S_{cs}$ as input, algorithm A still identifies L_t correctly.*

Definition 4. *(efficient identification in the limit of DTAs) A class DTAs C is* efficiently identifiable in the limit *(or simply* efficiently identifiable*) from labeled examples if there exist two polynomials p and q and an algorithm A such that:*

1. *given an input sample S of size $n = \sum_{\tau \in S} |\tau|$, A runs in time bounded by $p(n)$;*
2. *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 $q(|\mathcal{A}|)$.*

We also say that algorithm A identifies C efficiently in the limit. We now show that, in general, the class of DTAs cannot be identified efficiently in the limit. The reason for this is that there exists no polynomial q that bounds the size of a characteristic set for every DTA language.

In learning theory and in the grammatical inference literature there exist other definitions of efficient identification in the limit. It has for example been suggested to allow the examples in a characteristic set to be of length unbounded by the size of $\mathcal{A}$ [21]. The algorithm is then allowed to run in polynomial time in the sum of the lengths of these strings and the amount of examples in S. We use Definitions 3 and 4 because these are

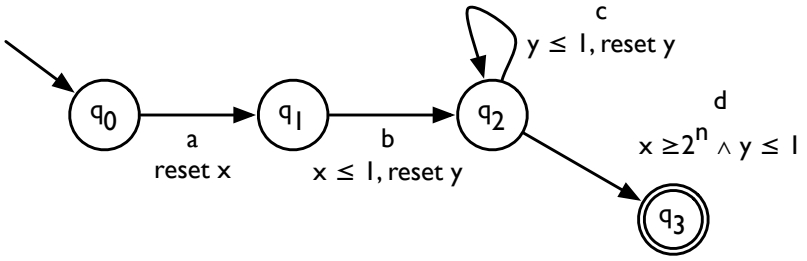

Figure 2: In order to reach state q_3, we require a string of exponential length (2^n). However, due to the binary encoding of clock guards, the DTA is of size polynomial in n.

the most restrictive form of efficient identification in the limit, and moreover, because allowing the lengths of strings to be exponential in the size of $\mathcal{A}$ results in an exponential (inefficient) identification procedure.

4. Polynomial distinguishability of DTAs

In this section, we study the length of timed strings in DTA languages. In particular, we focus on the properties of polynomial reachability (Definition 5) and polynomial distinguishability (Definition 7). These two properties are of key importance to identification problems because they can be used to determine whether there exist polynomial bounds on the lengths of the strings that are necessary to converge to a correct model.

4.1. Not all DTAs are efficiently identifiable in the limit

The class of DTAs is not efficiently identifiable in the limit from labeled data. The reason is that in order to reach some parts of a DTA, one may need a timed string of exponential length. We give an example of this in Figure 2. Formally, this example can be used to show that in general DTAs are not *polynomially reachable*:

Definition 5. *(polynomial reachability) We call a class of automata C* polynomially reachable *if there exists a polynomial function p, such that for any reachable state q from any automaton $\mathcal{A} \in C$, there exists a string τ, with $|\tau| \leq p(|\mathcal{A}|)$, such that τ reaches q in $\mathcal{A}$.*

Proposition 6. *The class of DTAs is not polynomially reachable.*

Proof. Let $C^* = \{\mathcal{A}_n \mid n \geq 1\}$ denote the (infinite) class of DTAs defined by Fig. 2. In any DTA $\mathcal{A}_n \in C^*$, state q_3 can be reached only if both $x \geq 2^n$ and $y \leq 1$ are satisfied. Moreover, $x \leq 1$ is satisfied when y is reset for the first time, and later y can be reset only if $y \leq 1$ is satisfied. Therefore, in order to satisfy both $y \leq 1$ and $x \geq 2^n$, y has to be reset 2^n times. Hence, the shortest string τ that reaches state q_3 is of length 2^n. However, since the clock guards are encoded in binary, the size of $\mathcal{A}_n$ is only polynomial in n. Thus, there exists no polynomial function p such that $\tau \leq p(|\mathcal{A}_n|)$. Since every $\mathcal{A}_n \in C^*$ is a DTA, DTAs are not polynomially reachable. □

The non-polynomial reachability of DTAs implies non-polynomial distinguishability of DTAs:

Definition 7. *(polynomial distinguishability) We call a class of automata C* polynomially distinguishable *if there exists a polynomial function p, such that for any two automata $\mathcal{A}, \mathcal{A}' \in C$ such that $L(\mathcal{A}) \neq L(\mathcal{A}')$, there exists a string $\tau \in L(\mathcal{A}) \triangle L(\mathcal{A}')$, such that $|\tau| \leq p(|\mathcal{A}| + |\mathcal{A}'|)$.*

Proposition 8. *The class of DTAs is not polynomially distinguishable.*

Proof. DTAs are not polynomially reachable, hence there exists no polynomial function p such that for every state q of any DTA $\mathcal{A}$, the length of a shortest timed string τ that reaches q in $\mathcal{A}$ is bounded by $p(|\mathcal{A}|)$. Thus, there is a DTA $\mathcal{A}$ with a state q for which the length of τ cannot be polynomially bounded by $p(|\mathcal{A}|)$. Given this DTA $\mathcal{A} = \langle Q, X, \Sigma, \Delta, q_0, F\rangle$, construct two DTAs $\mathcal{A}_1 = \langle Q, X, \Sigma, \Delta, q_0, \{q\}\rangle$ and $\mathcal{A}_2 = \langle Q, X, \Sigma, \Delta, q_0, \emptyset\rangle$. By construction, τ is the shortest string in $L(\mathcal{A}_1)$, and $\mathcal{A}_2$ accepts the empty language. Therefore, τ is the shortest string such that $\tau \in L(\mathcal{A}_1) \triangle L(\mathcal{A}_2)$. Since $|\mathcal{A}_1| + |\mathcal{A}_2| \leq 2 \times |\mathcal{A}|$, there exists no polynomial function p such that the length of τ is bounded by $p(|\mathcal{A}_1| + |\mathcal{A}_2|)$. Hence the class of DTAs is not polynomially distinguishable. □

It is fairly straightforward to show that polynomial distinguishability is a necessary requirement for efficient identifiability:

Lemma 9. *If a class of automata C is efficiently identifiable, then C is polynomially distinguishable.*

Proof. Suppose a class of automata C is efficiently identifiable, but not polynomially distinguishable. Thus, there exists no polynomial function p such that for any two automata $\mathcal{A}, \mathcal{A}' \in C$ (with $L(\mathcal{A}) \neq L(\mathcal{A}')$) the length of the shortest timed string $\tau \in L(\mathcal{A}) \triangle L(\mathcal{A}')$ is bounded by $p(|\mathcal{A}| + |\mathcal{A}'|)$. Let $\mathcal{A}$ and $\mathcal{A}'$ be two automata for which such a function p does not exist and let S_{cs} and S'_{cs} be their polynomial characteristic sets. Let $S = S_{cs} \cup S'_{cs}$ be the input sample for the identification algorithm A for C from Definition 4. Since C is not polynomially distinguishable, neither S_{cs} nor S'_{cs} contains a timed string τ such that $\tau \in L(\mathcal{A})$ and $\tau \notin L(\mathcal{A}')$, or vice versa (because no distinguishing string is of polynomial length). Hence, $S = (S_+, S_-)$ is such that $S_+ \subseteq L(\mathcal{A})$, $S_+ \subseteq L(\mathcal{A}')$, $S_- \subseteq L(\mathcal{A})^c$, and $S_- \subseteq L(\mathcal{A}')^c$. The second requirement of Definition 3 now requires that A returns both $\mathcal{A}$ and $\mathcal{A}'$, a contradiction. □

Thus, in order to efficiently identify a DTA, we need to be able to distinguish it from every other DTA or vice versa, using a timed string of polynomial length. Combining this with the fact that DTAs are not polynomially distinguishable leads to the main result of this section:

Theorem 1. *The class of DTAs cannot be identified efficiently.*

Proof. By Proposition 8 and Lemma 9. □

Or more specifically:

Theorem 2. *The class of DTAs with two or more clocks cannot be identified efficiently.*

Proof. The theorem follows from the fact that the argument of Proposition 6 requires a DTA with at least two clocks. □

An interesting alternative way of proving a slightly weaker non-efficient identifiability result for DTAs is by linking the efficient identifiability property to the *equivalence problem* for automata:

Definition 10. *(equivalence problem) The* equivalence problem *for a class of automata C is to find the answer to the following question: Given two automata $\mathcal{A}, \mathcal{A}' \in C$, is it the case that $L(\mathcal{A}) = L(\mathcal{A}')$?*

Lemma 11. *If a class of automata C is identifiable from polynomial data, and if membership of a C-language is decidable in polynomial time, then the equivalence problem for C is in* coNP*.*

Proof. The proof follows from the definition of polynomial distinguishability. Suppose that C is polynomially distinguishable. Thus, there exists a polynomial function p such that for any two automata $\mathcal{A}, \mathcal{A}' \in C$ (with $L(\mathcal{A}) \neq L(\mathcal{A}')$) the length of a shortest timed string $\tau \in L(\mathcal{A}) \triangle L(\mathcal{A}')$ is bounded by $p(|\mathcal{A}| + |\mathcal{A}'|)$. This example τ can be used as a certificate for a polynomial time falsification algorithm for the equivalence problem: On input $((\mathcal{A}, \mathcal{A}'))$:

1. Guess a certificate τ such that $|\tau| \leq p(|\mathcal{A}| + |\mathcal{A}'|)$.
2. Test whether τ is a positive example of $\mathcal{A}$.
3. Test whether τ is a positive example of $\mathcal{A}'$.
4. If the tests return different values, return reject.

The algorithm rejects if there exists a polynomial length certificate $\tau \in L(\mathcal{A}) \triangle L(\mathcal{A}')$. Since C is polynomially distinguishable, this implies that the equivalence problem for C is in coNP. □

The equivalence problem has been well-studied for many systems, including DTAs. For DTAs it has been proven to be PSPACE-complete [1], hence:

Theorem 3. *If* coNP $\neq$ PSPACE*, then DTAs cannot be identified efficiently.*

Proof. By Lemma 11 and the fact that equivalence is PSPACE-complete for DTAs [1]. □

Using a similar argument we can also show a link with the *reachability problem*:

Definition 12. *(reachability problem) The* reachability problem *for a class of automata C is to find the answer to the following question: Given an automaton $\mathcal{A} \in C$ and a state q of $\mathcal{A}$, does there exist a string that reaches q in $\mathcal{A}$?*

Lemma 13. *If a class of models C is identifiable from polynomial data then the reachability problem for C is in* NP*.*

Proof. Similar to the proof of Lemma 11. But now we know that for each state there has to be an example of polynomial length in the size of the target automaton. This example can be used as a certificate by a polynomial-time algorithm for the reachability problem. □

Theorem 4. *If* NP $\neq$ PSPACE*, then DTAs cannot be identified efficiently.*

Proof. By Lemma 13 and the fact that reachability is PSPACE-complete for DTAs [1]. □

These results seem to shatter all hope of ever finding an efficient algorithm for identifying DTAs. Instead of identifying general DTAs, we therefore focus on subclasses of DTAs that could be efficiently identifiable. In particular, we are interested in DTAs with equivalence and reachability problems in coNP and NP respectively. For instance, for DTAs that have at most one clock, the reachability problem is known to be in NLOGSPACE [13]. The main result of this paper is that these DTAs are also efficiently identifiable.

5. DTAs with a single clock are polynomially distinguishable

In the previous section we showed that DTAs are not efficiently identifiable in general. The proof for this is based on the fact that DTAs are not polynomially distinguishable. Since polynomial distinguishability is a necessary requirement for efficient identifiability, we are interested in classes of DTAs that *are* polynomially distinguishable. In this section, we show that DTAs with a single clock are polynomially distinguishable.

A one-clock DTA (1-DTA) is a DTA that contains exactly one clock, i.e., $|X| = 1$. Our proof that 1-DTAs are polynomially distinguishable is based on the following observation:

- If a timed string τ reaches some timed state (q, v) in a 1-DTA $\mathcal{A}$, then for all v' such that $v'(x) \geq v(x)$, the timed state (q, v') can be reached in $\mathcal{A}$.

This holds because when a timed string reaches (q, v) it could have made a larger time transition to reach all larger valuations. This property is specific to 1-DTAs: a DTA with multiple clocks can wait in q, but only bigger valuations can be reached where the difference between the clocks remains the same. It is this property of 1-DTAs that allows us to polynomially bound the length of a timed string that distinguishes between two 1-DTAs. We first use this property to show that 1-DTAs are polynomially reachable. We then use a similar argument to show the polynomial distinguishability of 1-DTAs.

Proposition 14. *1-DTAs are polynomially reachable.*

Proof. Given a 1-DTA $\mathcal{A} = \langle Q, \{x\}, \Sigma, \Delta, q_0, F\rangle$, let $\tau = (a_1, t_1) \dots (a_n, t_n)$ be a shortest timed string such that τ reaches some state $q_n \in Q$. Suppose that some prefix $\tau_i = (a_1, t_1) \dots (a_i, t_i)$ of τ ends in some timed state (q, v). Then for any $j > i$, τ_j cannot end in (q, v') if $v(x) \leq v'(x)$. If this were the case, τ_i instead of τ_j could be used to reach (q, v'), and hence a shorter timed string could be used to reach q_n, resulting in a contradiction. Thus, for some index $j > i$, if τ_j also ends in q, then there exists some index $i < k \leq j$ and a state $q' \neq q$ such that τ_k ends in (q', v_0), where $v_0(x) = 0$. In other words, x has to be reset between index i and j in τ. In particular, if x is reset at index i (τ_i ends in (q, v_0)), there cannot exist any index $j > i$ such that τ_j ends in q. Hence:

- For every state $q \in Q$, the number of prefixes of τ that end in q is bounded by the amount of times x is reset by τ.
- For every state $q' \in Q$, there exists at most one index i such that τ_i ends in (q', v_0). In other words, x is reset by τ at most $|Q|$ times.

Consequently, each state is visited at most $|Q|$ times by the computation of $\mathcal{A}$ on τ. Thus, the length of τ is bounded by $|Q| * |Q|$, which is polynomial in the size of $\mathcal{A}$. $\square$

Given that 1-DTAs are polynomially reachable, one would guess that it should be easy to prove the polynomial distinguishability of 1-DTAs. But this turns out to be a lot more complicated. The main problem is that when considering the difference between two 1-DTAs, we effectively have access to two clocks instead of one. Note that, although we have access to two clocks, there are no clock guards that bound both clock values. Because of this, we cannot construct DTAs such as the one in Figure 2. Our proof for the polynomial distinguishability of 1-DTAs follows the same line of reasoning as our proof of Proposition 14, although it is much more complicated to bound the amount of times that x is reset. We have split the proof of this bound into several proofs of smaller propositions and lemmas, the main theorem follows from combining these.

For the remainder of this section, let $\mathcal{A}_1 = \langle Q_1, \{x_1\}, \Sigma_1, \Delta_1, q_{1,0}, F_1\rangle$ and $\mathcal{A}_2 = \langle Q_2, \{x_2\}, \Sigma_2, \Delta_2, q_{2,0}, F_2\rangle$ be two 1-DTAs. Let $\tau = (a_1, t_1)\dots(a_n, t_n)$ be a shortest string that distinguishes between these 1-DTAs, formally:

Definition 15. *(shortest distinguishing string) A* shortest distinguishing string τ *of two DTAs $\mathcal{A}_1$ and $\mathcal{A}_2$ is a minimal length timed string such that $\tau \in L(\mathcal{A}_1)$ and $\tau \notin L(\mathcal{A}_2)$, or vice versa.*

We combine the computations of $\mathcal{A}_1$ and $\mathcal{A}_2$ on this string τ into a single computation sequence:

Definition 16. *(combined computation) The* combined computation *of $\mathcal{A}_1$ and $\mathcal{A}_2$ over τ is the sequence:*

$$\begin{aligned} \langle q_{1,0}, q_{2,0}, v_0\rangle &\xrightarrow{t_1} \langle q_{1,0}, q_{2,0}, v_0 + t_1\rangle \dots \\ \dots \langle q_{1,n-1}, q_{2,n-1}, v_{n-1} + t_n\rangle &\xrightarrow{a_n} \langle q_{1,n}, q_{2,n}, v_n\rangle \end{aligned}$$

where for all $0 \le i \le n$, v_i is a valuation function for both x_1 and x_2, and

$$(q_{1,0}, v_0) \xrightarrow{t_1} (q_{1,0}, v_0 + t_1)\dots(q_{1,n-1}, v_{n-1} + t_n) \xrightarrow{a_n} (q_{1,n}, v_n)$$

is the computation of $\mathcal{A}_1$ over τ, and

$$(q_{2,0}, v_0) \xrightarrow{t_1} (q_{2,0}, v_0 + t_1)\dots(q_{2,n-1}, v_{n-1} + t_n) \xrightarrow{a_n} (q_{2,n}, v_n)$$

is the computation of $\mathcal{A}_2$ over τ.

All the definitions of properties of computations are easily adapted to properties of combined computations. For instance, (q_1, q_2) is called a *combined state* and $\langle q_1, q_2, v\rangle$ is called a *combined timed state*. By using the properties of τ its combined computation, we now show the following:

Proposition 17. *The length of τ is bounded by a polynomial in the size of $\mathcal{A}_1$, the size of $\mathcal{A}_2$, and the sum of the amount of times x_1 and x_2 are reset by τ.*

Proof. Suppose that for some index $1 \leq i \leq n$, τ_i ends in $\langle q_1, q_2, v\rangle$. Using the same argument used in the proof of Proposition 14, one can show that for every $j > i$ and for some v', if τ_j ends in $\langle q_1, q_2, v'\rangle$, then there exists an index $i < k \leq j$ such that τ_k ends in (q_1', v_0) in $\mathcal{A}_1$ for some $q_1' \in Q_1$, or in (q_2', v_0) in $\mathcal{A}_2$ for some $q_2' \in Q_2$. Thus, for every combined state $(q_1, q_2) \in Q_1 \times Q_2$, the number of prefixes of τ that end in (q_1, q_2) is bounded by the sum of the amount of times r that x_1 and x_2 have been reset by τ. Hence the length of τ is bounded by $|Q_1| * |Q_2| * r$, which is polynomial in r and in the sizes of $\mathcal{A}_1$ and $\mathcal{A}_2$. □

We want to bound the number of clock resets in the combined computation of a shortest distinguishing string τ. In order to do so, we first prove a restriction on the possible clock valuations in a combined state (q_1, q_2) that is reached directly after one clock x_1 has been reset. In Proposition 14, there was exactly one possible valuation, namely $v(x) = 0$. But since we now have an additional clock x_2, this restriction no longer holds. We can show, however, that after the second time (q_1, q_2) is reached by τ directly after resetting x_1, the valuation of x_2 has to be smaller than all but one of the previous times τ reached (q_1, q_2):

Lemma 18. *If there exist three indexes $1 \leq i < j < k \leq n$ such that τ_i, τ_j, and τ_k all end in (q_1, q_2) such that $v_i(x_1) = v_j(x_1) = v_k(x_1)$, then the valuation of x_2 at index k has to be smaller than one of the previous valuations of x_2 at indexes i and j, i.e., $v_i(x_2) > v_k(x_2)$ or $v_j(x_2) > v_k(x_2)$.*

Proof. Without loss of generality we assume that $\tau \in L(\mathcal{A}_1)$, and consequently $\tau \notin L(\mathcal{A}_2)$. Let $\tau_{-k} = (a_{k+2}, t_{k+2}) \dots (a_n, t_n)$ denote the suffix of τ starting at index $k+2$. Let $a = a_{k+1}$ and $t = t_{k+1}$. Thus, $\tau = \tau_k(a, t)\tau_{-k}$.

We prove the lemma by contradiction. Assume that the valuations v_i, v_j, and v_k are such that $v_i(x_2) < v_j(x_2) < v_k(x_2)$. The argument below can be repeated for the case when $v_j(x_2) < v_i(x_2) < v_k(x_2)$. Let d_1 and d_2 denote the differences in clock values of x_2 between the first and second, and second and third time that (q_1, q_2) is reached by τ, i.e., $d_1 = v_j(x_2) - v_i(x_2)$ and $d_2 = v_k(x_2) - v_j(x_2)$.

We are now going to make some observations about the acceptance property of the computations of $\mathcal{A}_1$ and $\mathcal{A}_2$ over τ. However, instead of following the path specified by τ, we are going to perform a time transition in (q_1, q_2) and then only compute the final part τ_{-k} of τ. Because we assume that (q_1, q_2) is reached (at least) three times, and each time the valuation of x_2 is larger, in this way we can reach the timed state (q_2, v_k) in $\mathcal{A}_2$. Because we reach the same timed state (q_2, v_k) as τ, and because the subsequent timed string is identical to τ_{-k}, the acceptance property has to remain the same. We know that $\tau = \tau_k(a, t)\tau_{-k} \notin L(\mathcal{A}_2)$. Hence, it has to hold that $\tau_j(a, t + d_2)\tau_{-k} \notin L(\mathcal{A}_2)$ and that $\tau_i(a, t + d_1 + d_2)\tau_{-k} \notin L(\mathcal{A}_2)$. Similarly, since $\tau \in L(\mathcal{A}_1)$, and since τ_i, τ_j, and τ_k all end in the same timed state (q_1, v_k) in $\mathcal{A}_1$, it holds that $\tau_i(a, t)\tau_{-k} \in L(\mathcal{A}_1)$ and that $\tau_j(a, t)\tau_{-k} \in L(\mathcal{A}_1)$. Below we summarize this information in two tables (+ denotes true, and − denotes false):

value of d	0	d_1	d_2	$(d_1 + d_2)$
$\tau_i(a, t+d)\tau_{-k} \in L(\mathcal{A}_1)$	+			
$\tau_j(a, t+d)\tau_{-k} \in L(\mathcal{A}_1)$	+			
$\tau_i(a, t+d)\tau_{-k} \in L(\mathcal{A}_2)$				−
$\tau_j(a, t+d)\tau_{-k} \in L(\mathcal{A}_2)$			−	

Since τ is a shortest distinguishing string, it cannot be the case that both $\tau_i(a, t+d)\tau_{-k} \in L(\mathcal{A}_1)$ and $\tau_i(a, t+d)\tau_{-k} \notin L(\mathcal{A}_2)$ (or vice versa) hold for any $d \in \mathbb{N}$. Otherwise, $\tau_i(a, t+d)\tau_{-k}$ would be a shorter distinguishing string for $\mathcal{A}_1$ and $\mathcal{A}_2$. This also holds if we replace i by j. Hence, we obtain the following table:

value of d	0	d_1	d_2	$(d_1 + d_2)$
$\tau_i(a, t+d)\tau_{-k} \in L(\mathcal{A}_1)$	$+$			$-$
$\tau_j(a, t+d)\tau_{-k} \in L(\mathcal{A}_1)$	$+$		$-$	
$\tau_i(a, t+d)\tau_{-k} \in L(\mathcal{A}_2)$	$+$			$-$
$\tau_j(a, t+d)\tau_{-k} \in L(\mathcal{A}_2)$	$+$		$-$	

Furthermore, since τ_i ends in the same timed state as τ_j in $\mathcal{A}_1$ (directly after a reset of x_1), it holds that for all $d \in \mathbb{N}$: $\tau_i(a, t+d)\tau_{-k} \in L(\mathcal{A}_1)$ if and only if $\tau_j(a, t+d)\tau_{-k} \in L(\mathcal{A}_1)$. The table thus becomes:

value of d	0	d_1	d_2	$(d_1 + d_2)$
$\tau_i(a, t+d)\tau_{-k} \in L(\mathcal{A}_1)$	$+$		$-$	$-$
$\tau_j(a, t+d)\tau_{-k} \in L(\mathcal{A}_1)$	$+$		$-$	$-$
$\tau_i(a, t+d)\tau_{-k} \in L(\mathcal{A}_2)$	$+$			$-$
$\tau_j(a, t+d)\tau_{-k} \in L(\mathcal{A}_2)$	$+$		$-$	

By performing the previous step again, we obtain:

value of d	0	d_1	d_2	$(d_1 + d_2)$
$\tau_i(a, t+d)\tau_{-k} \in L(\mathcal{A}_1)$	$+$		$-$	$-$
$\tau_j(a, t+d)\tau_{-k} \in L(\mathcal{A}_1)$	$+$		$-$	$-$
$\tau_i(a, t+d)\tau_{-k} \in L(\mathcal{A}_2)$	$+$		$-$	$-$
$\tau_j(a, t+d)\tau_{-k} \in L(\mathcal{A}_2)$	$+$		$-$	$-$

Now, since $\tau_i(a, t+d_1)$ ends in the same timed state as $\tau_j(a, t)$ in $\mathcal{A}_2$, it holds that $\tau_i(a, t+d_1)\tau_{-k} \in L(\mathcal{A}_2)$ if and only if $\tau_j(a, t)\tau_{-k} \in L(\mathcal{A}_2)$ (since they reach the same timed state and then their subsequent computations are identical). Combining this with the previous steps results in:

value of d	0	d_1	d_2	$(d_1 + d_2)$
$\tau_i(a, t+d)\tau_{-k} \in L(\mathcal{A}_1)$	$+$	$+$	$-$	$-$
$\tau_j(a, t+d)\tau_{-k} \in L(\mathcal{A}_1)$	$+$	$+$	$-$	$-$
$\tau_i(a, t+d)\tau_{-k} \in L(\mathcal{A}_2)$	$+$	$+$	$-$	$-$
$\tau_j(a, t+d)\tau_{-k} \in L(\mathcal{A}_2)$	$+$	$+$	$-$	$-$

More generally, it holds that for any time value $d \in \mathbb{N}$, $\tau_i(a, t+d_1+d)\tau_{-k} \in L(\mathcal{A}_2)$ if and only if $\tau_j(a, t+d)\tau_{-k} \in L(\mathcal{A}_2)$. Hence, we can extend the table in the following way:

value of d	$\ldots$	$(d_1 + d_2)$	$2d_1$	$(2d_1 + d_2)$
$\tau_i(a, t+d)\tau_{-k} \in L(\mathcal{A}_1)$	$\ldots$	$-$	$+$	$-$
$\tau_j(a, t+d)\tau_{-k} \in L(\mathcal{A}_1)$	$\ldots$	$-$	$+$	$-$
$\tau_i(a, t+d)\tau_{-k} \in L(\mathcal{A}_2)$	$\ldots$	$-$	$+$	$-$
$\tau_j(a, t+d)\tau_{-k} \in L(\mathcal{A}_2)$	$\ldots$	$-$	$+$	$-$

Proceeding inductively, it is easy to see that for any value $m \in \mathbb{N}$ it holds that $\tau_i(a, t + m * d_1)\tau_{-k} \in L(\mathcal{A}_2)$ and $\tau_i(a, t + m * d_1 + d_2)\tau_{-k} \notin L(\mathcal{A}_2)$. This can only be the case if a different transition is fired for each of these $|\mathbb{N}|$ different values for m. Consequently, $\mathcal{A}_2$ contains an infinite amount of transitions, and hence $\mathcal{A}_2$ is not a 1-DTA, a contradiction. □

We have just shown that if a shortest distinguishing string τ reaches some combined timed state (q_1, q_2) with identical valuations for x_1 at least three times, then the valuation for x_2 at these times forms an *almost decreasing* sequence. In other words, in at most one of these previous times a valuation can be reached that is smaller than the last time. Since a clock reset gives a clock the constant value of 0, this lemma implies that after the second time (q_1, q_2) is reached by τ directly after resetting x_1, the valuation of x_2 will be almost decreasing. In the remainder of this section we use Lemma 18 to show that the number of clock resets of x_1 in a shortest distinguishing string τ can be polynomially bounded by the number of clock resets of x_2. Also we show that it follows that the number of times a combined state is reached by τ directly after resetting x_1 is polynomially bounded by the size of the automata. From this we then conclude that the number of resets is polynomial, and with Proposition 17 that τ is of polynomial length. We start by showing that if τ reaches (q_1, q_2) at least twice directly after a reset of x_1, then x_2 is reset before again reaching (q_1, q_2) and resetting x_1.

Proposition 19. *If for some indexes i and j, τ_i and τ_j end in (q_1, q_2) directly after a reset of x_1, and if there exists another index $k > j$ such that τ_k ends in (q_1, q_2) directly after a reset of x_1, then there exists an index $i < r \leq k$ such that τ_r ends in $(q_{2,k}, v_{2,0})$ in $\mathcal{A}_2$.*

Proof. By Lemma 18, it has to hold that $v_{2,i}(x_2) > v_{2,k}(x_2)$. The value of x_2 can only decrease if it has been reset. Hence there has to exist an index $i < r \leq k$ at which x_2 is reset. □

Thus, the number of different clock resets of x_1 in (q_1, q_2) by a shortest distinguishing string τ is bounded by the number of different ways that τ can reset x_2 before reaching (q_1, q_2). Let us consider these different resets of x_2. Clearly, after resetting x_2, some combined state (q_1', q_2') will be reached by τ. The possible paths that τ can take from (q_1', q_2') to (q_1, q_2) can be bounded by assuming τ to be quick. We call a distinguishing string *quick* if it never makes unnecessary long time transitions. In other words, all the time transitions in a computation of τ either makes the clock valuation equal to the lower bound of a transition in $\mathcal{A}_1$ or $\mathcal{A}_2$, or is of length 0. Clearly, we can transform any distinguishing string into a quick distinguishing string by modifying its time values such that no time transition is unnecessary. Because this transformation does not modify the transitions fired during its computation by $\mathcal{A}_1$ and $\mathcal{A}_2$, and hence does not influence its membership in $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$, we can assume τ to be quick without loss of generality.

A quick shortest distinguishing string has a very useful property, namely that it either has to fire at least one transition δ using the lower bound valuation of the clock guard of δ on its path from (q_1', q_2') to (q_1, q_2), or al the time tranition on this path are of length 0. This follows directly from the definition of quick. We use this property to polynomially bound the number of different ways in which x_2 can be reset by τ before reaching (q_1, q_2):

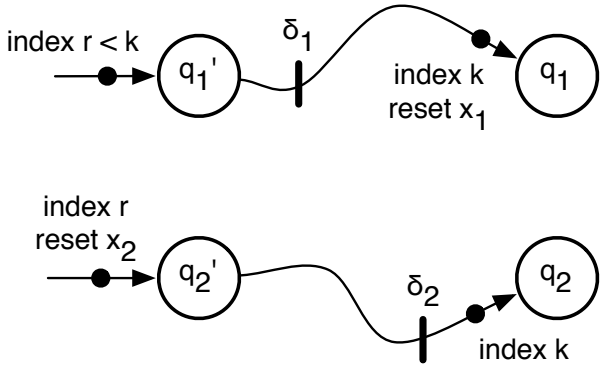

Figure 3: Bounding the number of resets of x_2 before a reset of x_1. The figures represent possible computation paths from the combined state (q_1', q_2') to the combined state (q_1, q_2). Clock x_1 is reset at index k, directly before entering (q_1, q_2). Clock x_2 is reset at index $r < k$, directly before entering (q_1', q_2'). We use this to polynomially bound the amount of times a transition δ_1 or δ_2 on the computation path from (q_1', q_2') to (q_1, q_2) can be fired.

Lemma 20. *The number of times x_2 is reset by τ before reaching a combined state (q_1, q_2) directly after a reset of x_1 is bounded by a polynomial in the sizes of $\mathcal{A}_1$ and $\mathcal{A}_2$.*

Proof. Suppose x_1 is reset at index k just before reaching (q_1, q_2), i.e., τ_k ends in $\langle q_1, q_2, v_k \rangle$, with $v_k(x_1) = 0$. The proof starts after x_1 is reset at least twice just before reaching (q_1, q_2). This clearly does not influence a possible polynomial bound. Let $i < j < k$ be two indexes where x_1 is also reset just before reaching (q_1, q_2). Thus, by Lemma 18, $v_k(x_2)$ is almost decreasing with respect to the previous indexes i and j. By Proposition 19, we know that x_2 is reset before index k. Let $r < k$ be the largest index before k where x_2 is reset. Let (q_1', q_2') be the combined state that is reached directly after this reset. By Lemma 18, $v_r(x_1)$ is also almost decreasing with respect to previous indexes. In addition, since τ is quick, either there exists at least one transition δ_1 or δ_2 that is fired using the lower bound valuation of its clock guard in either $\mathcal{A}_1$ or $\mathcal{A}_2$, or the computation from (q_1', q_2') to (q_1, q_2) contains only length 0 time transitions. This situation is depicted in Figure 3. We consider the possible cases in turn.

Suppose there exists such a transition in $\mathcal{A}_2$. Let δ_2 be the *last* such transition. Hence, $v_k(x_2)$ is the lower bound of the guard of δ_2. Since $v_k(x_2)$ is almost decreasing, it can at most occur once that τ reaches a higher value for x_2 than $v_k(x_2)$ in (q_1, q_2) just after a reset of x_1. Thus, δ_2 can be the last such transition at most twice. In total, there are therefore at most $2 * |\Delta_2|$ ways such a transition can exist in $\mathcal{A}_2$.

Suppose there exists no such a transition in $\mathcal{A}_2$, but at least one such transition δ_1 in $\mathcal{A}_1$. Suppose for the sake of contradiction that δ_1 is the *first* such transition *four* times at indexes $l4 > l3 > l2 > l > r$. Let $r4 > r3 > r2 > r$ be the corresponding indexes where τ reaches $(q_1,' q_2')$ just after a reset of x_2. The sequence $v_r(x_1), v_{r2}(x_1), v_{r3}(x_1), v_{r4}(x_1)$ is almost decreasing. Moreover, $v_l(x_1) = v_{l2}(x_1) = v_{l3}(x_1) = v_{l4}(x_1)$ because they are equal to the lower bound of the clock guard of δ_2. Consequently, $v_l(x_1) - v_r(x_1), v_{l2}(x_1) - v_{r2}(x_1), v_{l3}(x_1) - v_{r3}(x_1), v_{l4}(x_1) - v_{r4}(x_1)$ is *almost increasing*. Since there is no reset of x_1 between these paired indexes, the values

in this sequence represents the total time elapsed between resetting x_2 and reaching δ_1. Thus, $v_l(x_2) - v_r(x_2), v_{l2}(x_2) - v_{r2}(x_2), v_{l3}(x_2) - v_{r3}(x_2), v_{l4}(x_2) - v_{r4}(x_2)$ is also almost increasing because x_2 is also not reset between these paired indexes. Therefore, since $v_r(x_2) = v_{r2}(x_2) = v_{r3}(x_2) = v_{r4}(x_2) = 0$ due to the reset of x_2, this implies that the sequence $v_l(x_2), v_{l2}(x_2), v_{l3}(x_2), v_{l4}(x_2)$ is *almost increasing.*

However, since it also holds that $v_r(x_2) = v_{r2}(x_2) = v_{r3}(x_2) = v_{r4}(x_2)$, by Lemma 18, $v_r(x_2) = v_{r2}(x_2) = v_{r3}(x_2) = v_{r4}(x_2)$ has to form an almost decreasing sequence. Since it is impossible to have an almost increasing sequence of size 4 that is also an almost decreasing sequence, this leads to a contradiction. Thus, δ_1 can be the last such transition at most four times. In total, there are therefore at most $4 * |\Delta_1|$ ways such a transition can exist in $\mathcal{A}_2$.

Finally, suppose there exists no such a transition in both $\mathcal{A}_1$ and $\mathcal{A}_2$. In this case, since τ is quick, the time transitions in the computation path from (q_1', q_2') to (q_1, q_2) all have length 0. Therefore $v_k(x_2)$ will still be 0 when τ reaches (q_1, q_2), i.e., $v_k(x_2) = v_k(x_1) = 0$. This can occur only once since τ is a shortest distinguishing string.

In conclusion, the number of times that x_2 can be reset by τ before reaching (q_1, q_2) directly after a reset of x_1 is bounded by $|Q_1| * |Q_2| * (1 + 4 * |\Delta_1| + 2 * |\Delta_2|)$, which is polynomial in the sizes of $\mathcal{A}_1$ and $\mathcal{A}_2$. □

We are now ready to show the main result of this section:

Theorem 5. *1-DTAs are polynomially distinguishable.*

Proof. By Proposition 19, after the second time a combined state (q_1, q_2) is reached by τ directly after resetting x_1, it can only be reached again if x_2 is reset. By Lemma 20, the total number of different ways in which x_2 can be reset before reaching (q_1, q_2) and resetting x_1 is bounded by a polynomial p in $|\mathcal{A}_1| + |\mathcal{A}_2|$. Hence the total number of times a combined state (q_1, q_2) can be reached by τ directly after resetting x_1 is bounded by $|Q_1| * |Q_2| * p(|\mathcal{A}_1| + |\mathcal{A}_2|)$. This is polynomial in $|\mathcal{A}_1|$ and $|\mathcal{A}_2|$. By symmetry, this also holds for combined states that are reached directly after resetting x_2. Hence, the total number of resets of x_1 and x_2 by τ is bounded by a polynomial in $|\mathcal{A}_1| + |\mathcal{A}_2|$. Since, by Proposition 17, the length of τ is bounded by this number, 1-DTAs are polynomially distinguishable. □

As a bonus we get the following corollary:

Corollary 21. *The equivalence problem for 1-DTAs is in* coNP.

Proof. By Theorem 5 and the same argument as Lemma 11. □

6. DTAs with a single clock are efficiently identifiable

In the previous section, we proved that DTAs in general can not be identified efficiently because they are not polynomially distinguishable. In addition, we showed that 1-DTAs are polynomially distinguishable, and hence that they might be efficiently identifiable. In this section we show this indeed to be the case: 1-DTAs are efficiently identifiable. We prove this by providing an algorithm that identifies 1-DTAs efficiently in the limit. In other words, we describe a polynomial time algorithm ID_1-DTA (Algorithm 1) for the identification of 1-DTAs and we show that there exists polynomial

characteristic sets for this algorithm. An algorithm for identifying 1-DTAs is given an input set $S = (S_+, S_-)$ for some target 1-DTA language L_t, i.e., S is such that $S_+ \in L_t$ and $S_- \in L_t^c$. Given such an input sample, our algorithm works by identifying transitions δ of a 1-DTA $\mathcal{A}$ one by one. We prove the following in order to show that our algorithm identifies 1-DTAs efficiently:

- We show that for any possible δ there exists a polynomial amount of timed strings that can force our algorithm to identify any transition δ correctly (Lemma 23). The union of all of these timed strings form a characteristic set S_{cs}.
- We prove that a polynomial amount of these transitions is always sufficient for identifying any 1-DTA language (Proposition 24).
- In order to bound the length of these timed strings (and the size of S_{cs}) we make use of the facts that 1-DTAs are both polynomially reachable (Proposition 14) and polynomially distinguishable (Theorem 5).

The combination of these results satisfies all the constraints required for efficient identification in the limit (Definition 4), and hence shows that 1-DTAs are efficiently identifiable (Theorem 6).

6.1. An algorithm for identifying 1-DTAs efficiently

In this section, we describe our ID_1-DTA algorithm for identifying 1-DTAs efficiently from an input sample S. 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_1-DTA builds the structure of $\mathcal{A}$ piece by piece. Since we claim that ID_1-DTA 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_1-DTA algorithm for the efficient identification of 1-DTAs. 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.

Algorithm 1 Efficiently learning 1-DTAs from polynomial data: ID_1-DTA

Require: An input sample $S = (S_+, S_-)$ for a language L_t, with alphabet Σ
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 λ **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**
      $v_{\min} := \min\{v \mid (q, v)$ is reachable $\}$
      $c' := \max\{v \mid \neg\exists \langle q, q', a, g, r \rangle \in \Delta$ such that v 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 δ to Δ
        $V := \{v \mid \exists \tau \in S : \tau$ fires δ with valuation $v\}$
        $r :=$ *true* and $c_1 :=$ lower_bound$(\delta, V \cup \{v_{\min}\}, \mathcal{A}, S)$ (see Algorithm 3)
        $r :=$ *false* and $c_2 :=$ lower_bound$(\delta, V \cup \{v_{\min}\}, \mathcal{A}, S)$ (see Algorithm 3)
        if $c_1 \leq c_2$ **then**
          set $r :=$ *true* and $g := c_1 \leq x \leq c'$
        else
          set $r :=$ false and $g := c_2 \leq x \leq c'$
        end if
        for every state $q'' \in Q$ (first identified first) **do**
          $q' := q''$
          if consistent$(\mathcal{A}, S)$ is *true* (see Algorithm 2) **then**
            break
          else
            $q' := 0$
          end if
        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 τ ends in q' **then** set $F := F \cup \{q'\}$
        end if
        $c' := \min\{v \mid v$ satisfies $g\} - 1$
      end while
    end for
  end while

Example 2. *Suppose that after having identified a few transitions, our algorithm has constructed the (incomplete) 1-DTA $\mathcal{A}$ from Figure 4. 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 for b,*

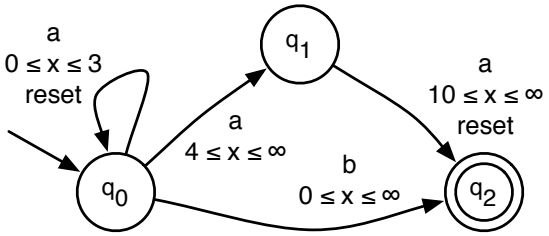

Figure 4: A partially identified 1-DTA. The transitions from state q_0 have been completely identified. State q_1 only has one outgoing transition. State q_2 has none.

or for a and valuations smaller than 9*, and state* q_2 *does not yet contain any transitions at all. Our algorithm first chooses 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 according to an 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 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

Algorithm 2 Checking for consistency: consistent

Require: An 1-DTA $\mathcal{A}$ and an input sample S
Ensure: Returns true if $\mathcal{A}$ is consistent with S
  for every positive example $\tau = (a_1, t_1) \dots (a_n, t_n)$ from S_+ **and**
  every negative example $\tau' = (a'_1, t'_1) \dots (a'_m, t'_m)$ from S_- **and**
  every pair of indices $1 \leq i \leq n$ and $1 \leq j \leq m$ **do**
    if τ_i ends in (q, v) and τ'_j ends in (q, v') **and**
    $\tau_{i+1} = \tau_i(a, t)$, $\tau'_{j+1} = \tau'_j(a, t + v - v')$ **and**
    $(a_{i+2}, t_{i+2}) \dots (a_n, t_n)$ equals $(a'_{j+2}, t'_{j+2}) \dots (a'_m, t'_m)$ **then**
      return *false*
    end if
  end for
  return *true*

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 final result $\mathcal{A}$ can be such that $S_+ \in L(\mathcal{A})$ and $S_- \in L(\mathcal{A})^c$. 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. The algorithm for checking this is shown in Algorithm 2. 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. This lower_bound routine is shown in Algorithm 3. 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 $(a,2)$. Note that $(a,4)(a,1)(a,3)$ reaches $(q',6)$ after a part (one time value) of the time transition taken by $(a,3)$. This time transition changes the suffix of $(a,4)(a,1)(a,3)$ from $(a,3)$ to $(a,2)$. 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

Algorithm 3 Obtaining the lower bound: lower_bound

Require: A new transition $\delta = \langle q, q', a, g, R \rangle$

Require: A set of possible lower bounds $V \cup \{v_{\min}\}$ for the clock guard of a transition $\delta = \langle q, q', a, g = v_{\min} \leq x \leq c', r \rangle$, and a 1-DTA $\mathcal{A}$ that is consistent with an input sample S

Ensure: Returns the smallest consistent lower bound $v \geq v_{\min}$ for δ

```
v := c'
for all valuations v' ∈ V ∪ {v_min} do
    g := v' ≤ x ≤ c'
    if consistent(A, S) is true and v' < v then set v := v'
end for
Set g to its original value.
return v
```

different. Therefore, there would exist two timed strings $\tau \in L_t$ and $\tau' \in L_t^c$ (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 $r = \mathit{true}$. Because of this, our algorithm identifies the clock reset r of δ at the same time it identifies the clock guard g. The method it uses to identify r is very simple: first set $r = \mathit{true}$ and then find the smallest consistent lower bound c_1 for g, then set $r = \mathit{false}$ 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 such that when the clock reset is set incorrectly, it 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, these 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. This holds unless the clock guard can only be satisfied by a unique valuation, i.e., unless $g = c \leq x \leq c$. However, in this case any setting for r is correct since all computations that fire δ will afterwards have the same clock valuation, independent of whether the clock is reset or not. Hence, the two settings for r do not influence whether computations reach the same timed state or not. Our algorithm can identify a 1-DTA for any possible assignment of accepting and rejecting labels to timed strings that reach different timed states. Thus, both settings can lead to 1-DTAs $\mathcal{A}$ such that $L(\mathcal{A}) = L_t$.

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 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.

6.2. Polynomial characteristic sets for 1-DTAs

We described an algorithm for the identification of 1-DTAs. The algorithm is consistent, i.e., given an input sample $S = (S_+, S_-)$, it always returns a 1-DTA $\mathcal{A}$ such that $S_+ \subset L(\mathcal{A})$ and $S_- \subset L(\mathcal{A})^c$. But it this result also desired? There are infinitely many consistent 1-DTAs. Given an input sample obtained from a 1-DTA language L_t (the target language), the desired result is a 1-DTA $\mathcal{A}$ such that $L(\mathcal{A}) = L_t$. In this section, we show that our algorithm returns the desired result *in the limit*. Moreover, it does so *efficiently*, i.e., it only requires a polynomial amount of examples in the size of the smallest 1-DTA model for the target language L_t. These are the two properties required for efficient identification in the limit of 1-DTAs and we prove this by showing

the existence of polynomial characteristic sets (Definition 3). We show this by proving that the four properties mentioned in the previous section hold for our algorithm:

- First, we prove that the ID_1-DTA algorithm is a polynomial-time algorithm. This satisfies properties 1 and 2.
- Second, we prove that in every iteration of the ID_1-DTA algorithm, there exists a polynomial amount of timed strings that can force our algorithm to identify the correct δ (Lemma 23). The union of all of these timed strings form a characteristic set S_{cs} for the ID_1-DTA algorithm.
- Third, we prove that the ID_1-DTA algorithm converges efficiently, i.e., that only a polynomial amount of (correctly identified) transitions are required to construct a 1-DTA $\mathcal{A}$ such that $L(\mathcal{A}) = L_t$ (Proposition 24)

In order to bound the length of the timed strings in S_{cs} (and hence the size of S_{cs}) we make use of the facts that 1-DTAs are both polynomially reachable (Proposition 14) and polynomially distinguishable (Theorem 5). The combination of these results satisfies all the constraints required for efficient identification in the limit (Definition 4), and hence shows that 1-DTAs are efficiently identifiable (Theorem 6).

Proposition 22. *ID_1-DTA is a polynomial-time algorithm (properties* 1 *and* 2*).*

Proof. In Algorithm 1, the main loop stops when the inner while loop cannot iterate anymore. Therefore, the running time of Algorithm 1 is bounded by the running time of the iterations on the inner while loop (plus a constant factor for determining $v_{\min}$ and c'). We have to show that the running time of the inner while loop can be bounded by a polynomial in the size of the input sample S.

In every iteration of the inner while loop of Algorithm 1, the algorithm identifies (constructs) one new transition. In the lower_bound subroutine, this transition is identified using a set V of valuations that is constructed using timed strings from S. It can be the case that V is empty but this case can be neglected since it occurs rarely and does not cause any additional iterations of the inner while loop. Since V is non-empty, the new transition is fired by at least one timed string τ from the input sample S. Every timed string fires a number of transitions equal to or less than its length. Hence, every example $\tau \in S$ can create at most $|\tau|$ iterations of the inner while loop in the worst case. Thus, in total there are at most $\sum_{\tau \in S} |\tau|$ iterations of the inner while loop. This is polynomial in the input size.

The lower_bound subroutine iterates $|V|$ times, which is bounded by $|S|$, and hence is bounded by a polynomial of the input size. Constructing V boils down to checking for every prefix τ_i of every timed string $\tau \in S$, whether τ_i fires δ. Thus, we need to make $|S|$ checks. These checks can be performed by running $\mathcal{A}$ over τ_i while increasing the value of i. This way, we can perform each check in $O(1)$ time. Constructing V thus requires $O(|S|)$ running time.

The loop for identifying the target state iterates $|Q|$ times. Since the algorithm can in the worst case identify a new state in every iteration of the inner while loop, this number is also bounded by a polynomially of the input size. The consistency check (consistent) can be implemented by trying all combinations of indexes of positive and

negative examples. Therefore its worst-case complexity is $\sum_{\tau \in S_+, \tau' \in S_-} |\tau| * |\tau'|$, which is polynomially bounded in the input size $|S|$.

Since polynomials are closed under composition, the running time of the inner while loop can be bounded by a polynomial in the size of the input sample. Hence, Algorithm 1 is a polynomial-time algorithm. □

The above proposition shows that our algorithm is time-efficient. More specifically, given any input sample S, ID_1-DTA returns in polynomial time a 1-DTA $\mathcal{A}$ that is consistent with S, i.e., such that $S_+ \subseteq L(\mathcal{A})$ and $S_- \subseteq L(\mathcal{A})^c$. We now show that the ID_1-DTA algorithm is also data-efficient. We first show that it requires a polynomial amount of data for a single transitions. Then we show that it converges after a polynomial amount of transitions.

Lemma 23. *There exist polynomial characteristic sets of the transitions of 1-DTAs for ID_1-DTA (property 3).*

Proof. First, our algorithm identifies whether q_0 is a final state. The example that ensures correct identification is the empty timed string λ: $\lambda \in S_+$ if $q_0 \in F$ and $\lambda \in S_-$ otherwise. Including this example in S makes our algorithm identify q_0 as a final state only when it should in fact be a final state. In a similar way, we now show that there exists a polynomial amount of polynomial-sized timed strings that ensure the correct identification of the transitions of $\mathcal{A}$. We prove this by showing that there exists a polynomial characteristic set for every transition δ such that our algorithm will identify the lower bound c, the reset r, and the target state q' of δ correctly.

The lower bound c. We need to ensure that our algorithm identifies the correct lower bound c. Without loss of generality, we assume the correct bound c to be minimal (as small as possible) and consistent with the target language L_t. More specifically, every smaller bound can only lead to a 1-DTA $\mathcal{A}'$ such that $L(\mathcal{A}') \neq L_t$. The lower_bound subroutine selects a valuation v that is the smallest valuation from V that leads to a consistent 1-DTA $\mathcal{A}$. Thus, we need to find examples that guarantee that the correct valuation is an element of V, that $\mathcal{A}$ is consistent when this valuation is selected as a lower bound, and that $\mathcal{A}$ is inconsistent when any smaller valuation is selected.

We can guarantee the correct valuation to be an element of V using a single example $\tau(a, c - v_{\min})$, where τ is a timed string that ends in $(q, v_{\min})$. Since $(q, v_{\min})$ is reachable, this example is guaranteed to exist. Moreover, when the algorithm constructs V, this example will end in (q, c). This ensures that c is an element of V. Naturally, it should be the case that $\tau(a, c - v_{\min}) \in S_+$ if and only if $\tau(a, c - v_{\min}) \in L_t$.

In order to ensure that $\mathcal{A}$ is consistent when c is selected we do not require any examples. This consistency is guaranteed by definition since our algorithm should identify c, and since the initial target of δ is a new state. Since this new state can be reached by no transition other than δ, the fact that S is an input sample for L_t guarantees that there can be no pair of timed strings in S that lead to an inconsistency when the correct lower bound c is selected.

Both a positive and a negative example are required to ensure that $\mathcal{A}$ is inconsistent when any smaller valuation is selected. These examples $\tau \in L_t$ and $\tau' \notin L_t$ should be such that if a smaller valuation is selected, then after some prefixes τ_i and τ'_j of τ and τ', they both reach the same timed state, and their subsequent computations are identical.

The valuations of the timed states in which τ_i and τ'_j end (after firing δ) depend on whether δ contains a reset or not. We later show the existence of examples which ensure that δ contains a reset only if δ_t contains a reset. Now, we therefore only need to show the examples that are required, depending on whether $r = \mathit{true}$ or $r = \mathit{false}$.

In the case that $r = \mathit{true}$, the two examples we require are $\tau(a, c - v_{\min})\tau'$, and $\tau(a, c - v_{\min} - 1)\tau'$, where τ is a timed string that ends in $(q, v_{\min})$, and τ' is a timed string such that $\tau(a, c - v_{\min})\tau' \in L_t$ and $\tau(a, c - v_{\min} - 1)\tau' \notin L_t$, or vice versa. Because $(q, v_{\min})$ is reachable, and because c is minimal and consistent with L_t, these examples are guaranteed to exist. Essentially, τ' is a string that distinguishes between the two languages $L_1 = \{\tau_1 \mid \tau(a, c - v_{\min})\tau_1 \in L_t\}$ and $L_2 = \{\tau_2 \mid \tau(a, c - v_{\min} - 1)\tau_2 \in L_t\}$. Since our algorithm should select c, it holds that $L_1 \neq L_2$. Otherwise, our algorithm might as well select $c - 1$, if all other identifications are performed correctly the result will still be such that $L(\mathcal{A}) = L_t$. This would contradict the fact that our algorithm should select c.

In the case that $r = \mathit{false}$, the two examples are $\tau(a, c - v_{\min})(b, t)\tau'$ and $\tau(a, c - v_{\min} - 1)(b, t + 1)\tau'$, where τ is a timed string, b is a symbol, t is a time value, and τ' is a timed string, such that τ ends in $(q, v_{\min})$ and $\tau(a, c - v_{\min})(b, t)\tau' \in L_t$ and $\tau(a, c - v_{\min} - 1)(b, t + 1)\tau' \notin L_t$, or vice versa. These examples are similar to the ones we require when $r = \mathit{true}$. The only difference being that in order to reach the same valuation in q' (and hence being capable of creating an inconsistency), the second example has to wait one additional time value. The examples are guaranteed to exists because c is minimal for L_t.

The reset r. The correct identification of the clock reset r of δ is ensured by similar examples. In the case that $r = \mathit{false}$, the two examples we require are: $\tau(a, c - v_{\min})\tau'$ and $\tau(a, c - v_{\min} + 1)\tau'$. In the case that $r = \mathit{true}$, we require: $\tau(a, c - v_{\min})(b, t)\tau'$ and $\tau(a, c - v_{\min} + 1)(b, t - 1)\tau'$. Because these examples reach the same valuations in q' only if x is reset while it should not be (or the other way around), these examples can be used to create a inconsistency. The difference of 1 time value is sufficient for such an inconsistency since 1 time value can always be the difference between satisfying and not satisfying a clock guard. Hence, these inconsistencies are guaranteed to exist unless the clock guard can only be satisfied by a single clock valuation. However, in this case it does not matter for the identified language whether the clock is reset or not, see Section 6.1.

In the case that r is set incorrectly, the timed strings that guarantee the correct identification of r ensure that there is an inconsistency when our algorithm selects the correct lower bound c. Hence, these examples ensure that setting r incorrectly results in a higher lower bound than setting r correctly. Thus, with these examples our algorithm is guaranteed to identify the correct reset.

The target state q'. We still need to ensure the identification of the correct target state q' of δ. This is achieved by ensuring inconsistencies for every incorrect target state $q'' \neq q'$: $\tau(a, c - v_{\min})(b, t)\tau'$ and $\tau''(b, t')\tau'$, where τ'' ends in q'', and t and t' are such that $\tau(a, c - v_{\min})(b, t)$ and $\tau''(b, t')$ both end in the same valuation (but not the same state). Naturally, these examples are guaranteed to exist, otherwise q' is identical to q''.

The states Q of $\mathcal{A}$ are identified correctly since our algorithm only adds new states when none of the old ones is a consistent target. We ensure the correct identification of the final states F by requiring for every state an example that ends in it when the

state is identified. This completes the specification of all the examples we require for the correct identification of δ by our algorithm.

The amount of examples required for one transition is clearly polynomial. Moreover, since all of the examples consist of a prefix that reaches some specific timed state and a suffix that is a distinguishing string, the fact that 1-DTAs are polynomially distinguishable guarantees that all of the examples are of polynomial length. 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 no longer returns $\mathcal{A}$. This proves the lemma. □

We have just shown that our algorithm is capable of returning a correct transition efficiently. We still have to show that it in fact will return a correct 1-DTA efficiently, i.e., that it converges after identifying a polynomial amount of transitions.

Lemma 24. *ID_1-DTA converges after identifying a polynomial amount of transitions (property 4).*

Proof. By the previous lemma, our algorithm is capable of making only correct identifications, i.e., there exists a characteristic set such that our algorithm only identifies those transitions that lead to $L(\mathcal{A}) = L_t$. Notice that, since a 1-DTA is a finite model, only a finite number of such identifications are necessary to make until our algorithm converge a correct 1-DTA, i.e., one such that $L(\mathcal{A}) = L_t$. We will show that the number of these identifications is polynomial in the size of the smallest correct 1-DTA.

Let $\mathcal{A}_t = \langle Q, x, \Sigma, \Delta, q_0, F\rangle$ be a the smallest 1-DTA such that $L_t = L(\mathcal{A}_t)$. Clearly, only the reachable parts of $\mathcal{A}_t$ matter for the acceptance of timed strings. Since 1-DTAs are polynomially reachable, there exists some polynomial p such that these parts can be reached by timed strings of length $p(|\mathcal{A}_t|)$. Without loss of generality, we assume one such timed string τ_q to be included the input sample for every state $q \in Q$, where τ_q reaches the smallest reachable valuation in q.

In a single complete run (over all states and symbols) of the main loop, our algorithm identifies new transitions for every newly reachable valuation in any timed state. Hence, every τ_q fires at least one new transition, advancing the computation of τ_q by at least one step. At most $p(|\mathcal{A}_t|)$ of these steps are required before τ_q reaches the smallest reachable valuation in q. Thus, the main loop is run at most $p(|\mathcal{A}_t|)$ times before the smallest reachable valuation in any state of $\mathcal{A}_t$ can be identified by our algorithm.

In one iteration of the main loop, new transitions are identified for every newly reachable region of valuations in any timed state. If all transitions (and states) are identified correctly so far, then these regions correspond to parts of the transitions Δ of $\mathcal{A}_t$. Thus, if our algorithm identifies the transitions for these parts correctly, then at most $|\Delta|$ new correct transitions can be identified.

In conclusion, at most $p(|\mathcal{A}_t|) * |\Delta|$ transitions are identified in total before the transition for the smallest reachable valuation can be identified in any state of $\mathcal{A}_t$. This is clearly polynomial in $|\mathcal{A}_t|$. Once the transitions for the smallest reachable valuation can be identified in every state, every transition can be identified. Hence, by the previous proposition, every transition can be identified correctly. Thus, our algorithm can return a 1-DTA $\mathcal{A}$ such that $L(\mathcal{A}) = L_t$ by identifying a number of transitions polynomial in $|\mathcal{A}_t|$. □

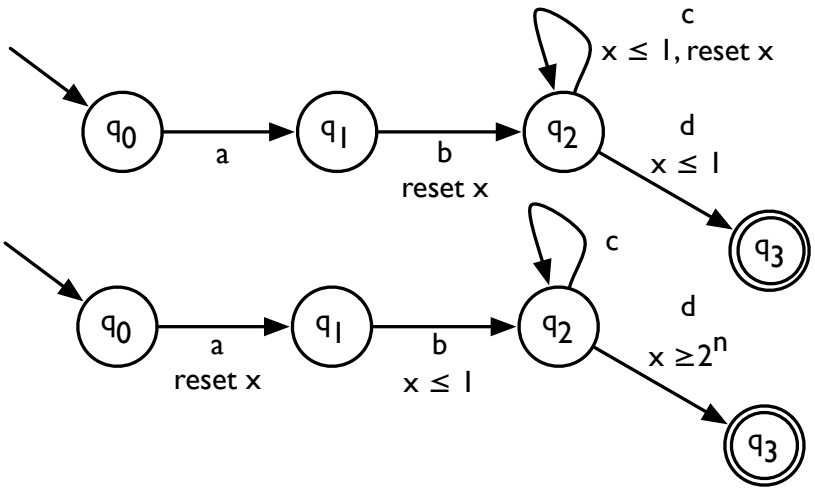

Figure 5: Two 1-DTAs. Taking the intersection of the languages of these 1-DTAs results in a language equivalent to the language of the 2-DTA of Figure 2.

The two lemmas above show that our algorithm converges from polynomial data. In other words, if S contains a characteristic subsample S_{cs} for some target language L_t, then ID_1-DTA returns a correct 1-DTA $\mathcal{A}$, i.e., such that $L(\mathcal{A}) = L_t$. Combined with the time efficiency, this is sufficient to prove the efficient identifiability of 1-DTAs:

Theorem 6. *1-DTAs are efficiently identifiable in the limit from labeled examples.*

Proof. By Proposition 22 and Lemma 24, if all the examples from Lemma 23 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. □

7. The power of one-clock and multi-clock DTAs

In general, our lemmas and theorems in the previous sections are statements about the expressive power of one-clock DTAs (1-DTAs) and multi-clock DTAs (n-DTAs). These statements are important by themselves, i.e., not necessarily restricted to just the identification problem. We believe that there can be other problems (such as reachability analysis) that may benefit from our results. In this section we give an overview of the consequences of our results in general.

First of all, Theorem 5 tells us that:

> The length of the shortest string in the *symmetric difference* $L(\mathcal{A}_1) \triangle L(\mathcal{A}_2)$ between the languages of any two 1-DTAs $\mathcal{A}_1$ and $\mathcal{A}_2$ is of length bounded by a polynomial p in the sizes of $\mathcal{A}_1$ and $\mathcal{A}_2$.

In the timed automata (formal methods) field, it is well-known that there exists a language-preserving transformation from any DTA with n clocks to n 1-DTAs [3]. In

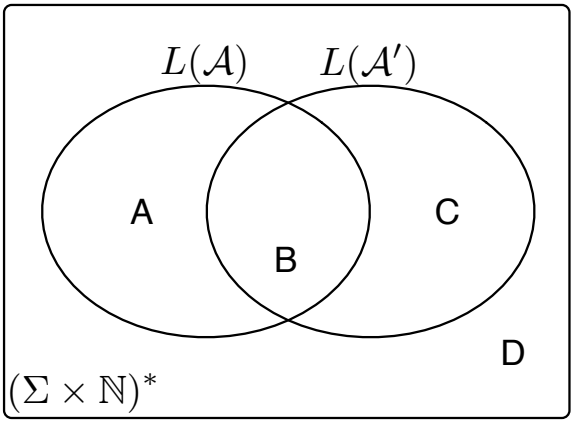

Figure 6: The power of two 1-DTAs depicted in a Venn diagram. $\mathcal{A}$ and $\mathcal{A}'$ are two 1-DTAs. In the Venn diagram there are 4 spaces. The language $L(\mathcal{A})$ of $\mathcal{A}$ is equal to $A \cup B$. The language $L(\mathcal{A}')$ of $\mathcal{A}'$ is equal to $C \cup B$. B is the intersection of $L(\mathcal{A})$ and $L(\mathcal{A}')$. Thus, B is a 2-DTA language, and hence we cannot polynomially bound the size of the smallest timed string in B (Proposition 8). Using complementation and again intersection, the same holds for A, C, and D. However, we can polynomially bound the size of the smallest timed string in $X \cup Y$ where $X, Y \in \{A, B, C, D\}$ (Proposition 14 and Theorem 5).

fact, this transformation is quite straightforward: given an n-DTA, create a 1-DTA copy for every clock and set all occurrences of different clocks in clock guard to *true*. The intersection of the languages of these n 1-DTAs is (with some additional transformations) equal to the language of the original DTA. An example of two such 1-DTAs is shown in Figure 5. Figure 2 is the intersected n-DTA version of Figure 5.

The combination of this fact with Proposition 8 tells us that:

> The shortest string in the *intersection* $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$ between the languages of two 1-DTAs $\mathcal{A}_1$ and $\mathcal{A}_2$ *cannot* be bounded by a polynomial p in the sizes of $\mathcal{A}_1$ and $\mathcal{A}_2$.

The two statements above tell us something important regarding the expressive power of 1-DTAs and n-DTAs. We also know that the complement $L(\mathcal{A})^{\mathrm{C}}$ of the language of a 1-DTA $\mathcal{A}$ can be easily computed (polynomially) by changing the final and non-final states of $\mathcal{A}$. Since 1-DTA are polynomially reachable (Proposition 14), this implies that:

> The shortest string in the (non-symmetric) *difference* $L(\mathcal{A}_1) \backslash L(\mathcal{A}_2) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)^{\mathrm{C}}$ between the languages of any two 1-DTAs $\mathcal{A}_1$ and $\mathcal{A}_2$ *cannot* be bounded by a polynomial p in the sizes of $\mathcal{A}_1$ and $\mathcal{A}_2$.

In fact, using the basic set operations, the above statements, and some algebra, we can show many of such statements. These statements are summarized in Figure 6. In general, they tell us that the intersection of the languages of two 1-DTAs is strictly more expressive than either of these 1-DTAs: it can be represented by a 1-DTA (by applying the region construction [1] to all but one clock), but at the cost of exponential blowup. Since this tells us something about the power of clocks in DTAs in general, we believe

that it should be possible to apply our theorems to for example reachability analysis in timed automata. Perhaps they can be used to reduce the complexity of reachability analysis depending on which type of combination (using set operations) of the languages of 1-DTAs has to be analyzed.

8. Summary and future work

In this paper we have shown the following main results:

1. Polynomial distinguishability (Definition 7) is a necessary condition for efficient identification in the limit (Lemma 9).
2. DTAs with two or more clocks are not polynomially distinguishable (Theorem 1 and 2).
3. DTAs with one clock (1-DTAs) are polynomially distinguishable (Theorem 5).
4. 1-DTAs are efficiently identifiable using our ID_1-DTA algorithm (Algorithm 1 and Theorem 6).

These results are of importance for anyone interested in identifying timed systems (and DTAs in particular). Most importantly, the efficiency results tell us that identifying a 1-DTA from timed data is more efficient than identifying an equivalent DFA. Furthermore, the results show that anyone who needs to identify a DTA with two or more clocks should either be satisfied with sometimes requiring an exponential amount of data, or he or she has to find some other method to deal with this problem, for instance by identifying a subclass of DTAs (such as 1-DTAs).

The reason for our inefficiency results is due to the explicit representation of time (using numbers) that is used in DTAs. We know that for any DTA, there exists a DFA that models the exact same language (is language equivalent) but with an implicit representation of time (using states). The standard method of creating such a DFA is the region construction (see [1]). This construction results in a DFA of size exponential in both the binary encoding of time values used in the DTA and the amount of clocks of the DTA. Therefore, it is not unexpected that DTAs cannot be identified efficiently.

It comes as a surprise that 1-DTAs can be identified efficiently. This is surprising since the region construction still results in an exponentially larger DFA when applied to a 1-DTA: time is still represented in binary (instead of unary). The main reason that 1-DTAs can be identified efficiently is an important lemma regarding their modeling power (Lemma 18). This lemma states a restriction on the shortest timed strings in the symmetric difference of two 1-DTA languages. This restriction is then used to prove that 1-DTAs can be distinguished using a timed string of polynomial length. We believe however, that this restriction and the results that come from this restriction have consequences that are of importance outside the scope of the DTA identification problem (see Section 7).

Future work. Our results give a first general view on the complexity of identifying DTAs. However, there are of course still many interesting questions left that can be answered by future work.

A fundamental question is whether an n-DTA identification algorithm can be used to identify 1-DTAs efficiently, while representing them using n-DTAs. A similar result holds for deterministic and non-deterministic finite state automata (DFAs and NFAs):

while NFAs cannot be identified efficiently, an NFA identification algorithm can be used to identify DFAs efficiently [20].[6] This is possible because NFAs and DFAs are language equivalent. The other way around, we could also identify NFAs using an DFA identification algorithm, and then returning the smallest NFA that is language equivalent to the identified DFA. Such an approach is not efficient however, because the finding this minimal NFA is a very difficult problem, i.e., to be precise it is PSPACE-complete [11]. Moreover, in order to converge to an NFA-language, a DFA identification algorithm requires an exponential amount of data. In contrast, the NFA identification algorithm identifies some of these NFA-languages more efficiently. Whether this is also possible for 1-DTAs and n-DTAs is an open problem.

Another interesting question is whether 1-DTAs are really the largest class of efficiently identifiable DTAs. To the best of our knowledge, the identification of DERAs in [9] is the only other work that deals with the identification of (subclasses of) DTAs. It would be interesting to search for other subclasses of DTAs besides 1-DTAs that can be identified efficiently. A good first step in this search is to look for classes of DTAs for which the proof of Proposition 6 does not hold. An example of such DTAs are DTAs with the restriction that clock guards can only compare the valuations of a single clock to constants. These are more expressive than 1-DTAs (1-DTAs are language equivalent but with exponential blowup), but they are not expressive enough to construct the DTA used in the proof of Proposition 6. We would like to determine whether this class of DTAs is efficiently identifiable in future work.

An interesting observation based on the power of one-clock and multi-clock DTAs is that it is possible to identify an n-DTA by representing it using n 1-DTAs and taking their intersection. This can be viewed as a type of ensemble method (see, e.g., [5]). Perhaps these 1-DTAs can all be learned efficiently, and perhaps some form of teamwork can then be used to give some performance guarantees for the n-DTA identification problem. A similar idea, based on the non-closure under union of sets identifiable from text, was one of the main motivations for team-learning (see, e.g., [10]). It would be interesting to investigate such an approach for the identification of n-DTAs.

References

[1] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[2] R. Alur, L. Fix, and T. A. Henzinger. Event-clock automata: a determinizable class of timed automata. *Theoretical Computer Science*, 211(1):253–273, 1999.

[3] E. Asarin, P. Caspi, and O. Maler. Timed regular expressions. *Journal of the Association for Computing Machinery*, 49(2):172–206, 2001.

[4] C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27(2):125–138, 1997.

[5] T. G. Dietterich. Ensemble methods in machine learning. In *First International Workshop on Multiple Classifier Systems*, volume 1857 of *LNCS*, pages 1–15. Springer, 2000.

[6] E. M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[7] E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.

[6] The proposed NFA identification algorithm is a query learning algorithm. However, any efficient query learning algorithm can be transformed into an algorithm that is efficient in the limit from labeled data [8].

[8] S. A. Goldman and H. D. Mathias. Teaching a smarter learner. *Journal of Computer and System Sciences*, 52(2):255–267, 1996.
[9] O. Grinchtein, B. Jonsson, and P. Petterson. Inference of event-recording automata using timed decision trees. In *CONCUR*, volume 4137 of *LNCS*, pages 435–449. Springer, 2006.
[10] S. Jain, D. Osherson, J. S. Royer, and A. Sharma. *Systems that learn*. MIT Press, 1999.
[11] T. Jiang and B. Ravikumar. Minimal NFA problems are hard. In *SIAM Journal of Computation*, volume 22, pages 1117–1141, 1993.
[12] K. J. Lang, B. A. Pearlmutter, and R. A. Price. Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In *Grammatical Inference*, volume 1433 of *LNCS*. Springer, 1998.
[13] F. Laroussinie, N. Markey, and P. Schnoebelen. Model checking timed automata with one or two clocks. In *CONCUR*, volume 3170 of *LNCS*, pages 387–401. Springer, 2004.
[14] K. G. Larsen, P. Petterson, and W. Yi. Uppaal in a nutschell. *International journal on software tools for technology transfer*, 1(1-2):134–152, 1997.
[15] J. Oncina and P. Garcia. Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis*, volume 1 of *Series in Machine Perception and Artificial Intelligence*, pages 49–61. World Scientific, 1992.
[16] L. Pitt and M. Warmuth. The minimum consistent DFA problem cannot be approximated within and polynomial. In *Annual ACM Symposium on Theory of Computing*, pages 421–432. ACM, 1989.
[17] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77, pages 267–296, 1989.
[18] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing, 1997.
[19] S. Verwer, M. de Weerdt, and C. Witteveen. An algorithm for learning real-time automata. In M. van Someren, S. Katrenko, and P. Adriaans, editors, *Proceedings of the Sixteenth Annual Machine Learning Conference of Belgium and the Netherlands*, pages 128–135, 2007.
[20] T. Yokomori. *Learning non-deterministic finite automata from queries and counterexamples*, volume 13 of *Machine Intelligence*, pages 196–189. Oxford University Press, 1993.
[21] T. Yokomori. On polynomial-time learnability in the limit of strictly deterministic automata. *Machine Learning*, 19(2):153–179, 1995.