# TIMED AUTOMATA FOR BEHAVIORAL PATTERN RECOGNITION

Sicco Verwer[1] Mathijs de Weerdt Cees Witteveen

Delft University of Technology, P.O.Box 5031 2600GA Delft

**Abstract**

We argue that timed models are a suitable framework for the detection of behavior in real-world event systems. A timed model which detects behavior is constructible by a domain expert. The inference of these timed models from data is a hard problem. We prove the inference of a class of timed automata (event recording automata) to be harder than the inference of finite automata.

## 1 Introduction

Real-world systems can often be modelled by systems consisting of states and transitions between these states, triggered by certain events. These systems are called *discrete event systems* [4]. We are interested in the creation of such an event system for the detection of different types of behavior within a real-world system. The detection takes place using *time series data* from different kinds of sensors. Usually a set of pre-specified patterns is given, partially identifying the events which should be detected in the time series data. Given this set, it is often unknown or hard to specify which pattern exactly identifies an event. Therefore it is common to construct a *classifier* from labelled data-sets, which given new data determines what label is most likely to belong to the data. In our project we want to infer behavioral patterns (in particular driving behavior) using low-level sensor data. Given a partial description of a pattern and some labelled low-level data we want to infer an event system which identifies the pattern.

A common model of an event system is a *finite automaton* (FA). An advantage of this model is the intuitive framework, i.e. the model can be interpreted by domain experts. A problem, however, is that FAs fail to model an important part of many real-world event systems, namely the timed relations between events. Real-world system behavior can often be described by a sequence of specific events, which are related to each other by the time at which they occur. For example, in our project the time between vehicle speedups and slowdowns is significant. A sequence of fast changes from slowing down to speeding up and vice versa indicates driving in a city, while a sequence of slow changes indicates driving on a freeway. The model of a FA which includes the notion of time is called a *timed automaton* (TA), first introduced by Alur and Dill (see [1]). In this model, each symbol of a word occurs at a certain point in time and the transitions between system states contain *guards*, i.e. constraints on these time points.

The inference of FAs is a well-studied problem in learning theory. There are, however, almost no studies of the inference of TAs (or timed systems in general) from data. One approach to the inference of timed systems is a transformation method proposed by Grinchtein et al. [7]. They show that an *event recording automaton* (ERA) is learnable by transforming it into a *deterministic finite automaton* (DFA). The resulting DFA can be learned using a modified version of a learning algorithm. Unfortunately, however, in the

[1]This research has been supported and funded by the Dutch Ministry of Economical Affairs under the SENTER program.

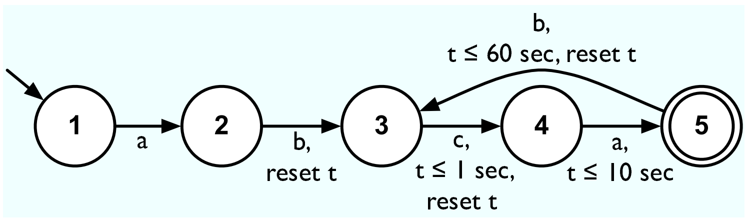

**Figure 1:** *The harmonica automaton. The alphabet consists of* $\{a, b, c\}$, *with* $a$ *representing constant speed,* $b$ *a slowdown and* $c$ *a speedup. The automaton uses just one clock which is reset when the automaton fires transitions which contain the reset command. A transition which includes a clock guard can only be fired when the guard is satisfied. For example when the automaton enters state* 3, *a* $c$ *symbol should occur within 1 second in order to make the transition to state* 4.

worst case, the size of the resulting DFA is exponential in the size of the alphabet. In this paper we show that unless P = NP this exponential blow-up cannot be avoided: While in some cases the inference of FA's is tractable, we prove the inference of ERAs to be a lot harder because of the expressiveness of timed constraints.

We will start this paper with a short overview of TAs. Subsequently we will briefly address the problem of FA inference. Next, we prove that the inference of ERAs is NP-hard under favorable circumstances. We conclude with some remarks for future work.

## 2 Timed Automata

The framework of TAs is a simple yet powerful way to describe the behavior of a real-time system. The basic idea behind these automata is to add a notion of time to the FA model, in the form of timed words. A *timed word* $w_t$ over an alphabet $\Sigma$ is a finite sequence of pairs $(\sigma_1, \tau_1)(\sigma_2, \tau_2) \ldots (\sigma_n, \tau_n)$, where $n \in \mathbb{N}$ and for all $1 \leq i \leq n$ $\sigma_i \in \Sigma$ is a symbol occurring at time $\tau_i \in \mathbb{R}$, such that the sequence of time points is strictly non-decreasing $\tau_1 \leq \tau_2 \leq \ldots \leq \tau_n$. A *timed language* is a set of timed words over $\Sigma$. TAs are FAs whose transitions are constrained with timing requirements so that they accept a timed language. The timing requirements are added to a FA model in the form of clocks and clock guards. A *clock* $c \in \mathbb{R} \cup \perp$ is a real-valued variable which can be undefined. The value of a clock always increases synchronously with the real-world clock. When a transition in a TA fires it can reset the value of any number of clocks to zero. The value of a clock is undefined when it has never been reset, otherwise it records the time elapsed since its last reset. A *clock guard* (or constraint) is an arithmetic constraint on clocks defined inductively as $g = c \leq n | n \leq c | c_1 \leq c_2 | g_1 \vee g_2 | g_1 \wedge g_2$, where $c$, $c_1$, and $c_2$ are clocks, $n \in \mathbb{Q}$, and $g_1$ and $g_2$ are clock constraints. A clock guard is *satisfied* by a set of clocks when the guard evaluates to *true*.

Formally, a *timed automaton* (TA) is a tuple $\mathcal{A} =< Q, C, \Sigma, T, q_0, F >$, where $Q$ is a finite set of states, $C$ is a finite set of clocks, $\Sigma$ is a finite set of symbols, $T$ is a finite set of transitions, $q_0$ is the start state, and $F \subseteq Q$ is a subset of final states. A transition $t \in T$ is a tuple $< q, q', \sigma, \phi, R >$, where $q, q' \in Q$ are the source and target states, $\sigma \in \Sigma$ is a symbol, $\phi$ is a clock guard, and $R \subseteq C$ is a finite set of clocks. Such a transition is interpreted as follows: whenever the machine is in state $q$, reading $\sigma$, and the clock guard $\phi$ is satisfied, the machine can move to state $q'$, resetting all clocks in $R$. A *run* (or computation) of a TA $\mathcal{A}$ over a timed word $w_t = (\sigma_1, \tau_1) \ldots (\sigma_n, \tau_n)$ is a finite sequence of states and transitions $q_0 t_1 q_1 t_2 \ldots q_{n-1} t_n q_n$, such that for all $1 \leq i \leq n : t_i =< q_{i-1}, q_i, \sigma_i, \phi, R >$, where $\phi$ is satisfied by $C$ at time $\tau_i$ and each $c \in R$ is reset at time $\tau_i$. An run of a TA $\mathcal{A}$ over a timed word $w_t$ of length $n$ such that $q_n \in F$ is called an *accepting run.* The language $L(\mathcal{A})$ of a TA $\mathcal{A}$ is the set of timed words for which there exists an accepting run.

That TAs are suitable for modelling the driving behavior patterns we are interested in can be illustrated by a simple example. In our project we try to detect different types of driver behavior in transport vehicles. An example of driving behavior is the so called 'Harmonica effect'. This is a common effect when the transport vehicle driver is driving behind another vehicle. An expert description of this effect is:

> *The driver is driving on the freeway with constant speed when suddenly a slowdown occurs, immediately followed by a speedup, and then within 10 seconds the speed becomes constant again. Within a minute the slowdown occurs again etc.*[2]

This behavior can be described by a TA with symbols for the slowdown, constant speed, and speedup events. This automaton is depicted in Figure 1. We tested this automaton on real sensor data from a transport vehicle. The automaton did detect instances of the harmonica effect, but there was no perfect match between the detected instances and the actual instances. We therefore want to use learning techniques to fine-tune such TAs constructed from partial description. These learning techniques can also be used to infer relations which were previously unknown to experts, or to refine a previously learned model.

## 3 Inference of Automata

In many real-world applications, the knowledge from experts will not be sufficient to build the complete FA model because certain properties of the system are simply not known in detail. In these cases, data can be gathered from the system to be modelled, and can subsequently be used to infer the FA model. In real-world applications one usually has a set of examples of a language, known as a *sample.* Such a sample consists of words and for each word an indication whether it belongs to the language (positive) or not (negative). From a sample we try to infer the minimal automaton which accepts all the positive and none of the negative examples. Gold proved the correct inference of a minimal DFA from incomplete data is known to be NP-complete [6]. Approximating the minimal DFA from a (sparse) sample, however, can in the average case be done in polynomial time [5]. Most of these approximation algorithms use a technique called *state merging.* This technique first creates a complete prefix tree from all the positive examples, and continuously merges states from this tree until it has constructed a small DFA.

Another approach than to learn from a sparse sample, is to learn from a dense sample by using Angluin's *query learning* [3]. This requires the use of an oracle which answers *membership* and *equivalence queries.* A membership query is a question which asks whether one specific word is included in a language. An equivalence query asks whether a possible hypothesis (automaton) is the correct one, the oracle returns a counterexample if this is not the case. Angluin's fundamental algorithm learns a correct DFA using a polynomial amount of these queries. In the real world we do not have access to such an oracle, but there are ways to replace this component, for example by a dense data-set and a data-mining tool which answers queries using this data-set. Such a data-set would allow us to answer even more powerful queries such as *subset* and *superset* queries. It is however far from trivial to efficiently answer queries from large data-sets. The result of the use of a

[2] Joost van der Luijt from, Van der Luyt Transport, personal communication.

data-set for query answering is an approximation when the language is infinite, because a data-set will never (at least not provably) be able to include all possible examples.

## 4 Inference of Timed Automata

We will show that, in contrast to learning FAs from sample data, the problem of learning even a simple class of TAs is intractable.[3] For TAs in general the problem of determinization (i.e. finding a deterministic TA equivalent to a non-deterministic TA) is undecidable, which implies that the problem of language inclusion is also undecidable. Therefore, different subclasses of TAs have been investigated, such as the *event recording automaton* (ERA) [2]. In an ERA, there is exactly one clock for each symbol in the alphabet with each clock recording the time since the last occurrence of the corresponding symbol. Given a timed word $w_t = (\sigma_1, \tau_1) \dots (\sigma_n, \tau_n)$ the value of the clock $c_a$ for symbol $a$ at the $i - th$ position of $w_t$ is equal to $\tau_i - \tau_j$, where $\tau_j$ is the largest position preceding $i$ such that $\sigma_j$ equals $a$. Apart from this ERAs are identical to TAs. Note that the reassignment of clocks in an ERA is not controlled by the automaton itself: at each point in time the value of each clock is determined solely by the input word. This property allows for the determinization of ERAs.

But for the ERA-subclass, the problem of inferring the correct automaton from data under quite favourable circumstances is still a hard one: *even when the underlying FA for the timed language to be inferred is known, the problem to infer the corresponding ERA is an NP-hard problem.* This essentially implies that even if the only problem is to identify the simple clock guards for a given FA this constitutes an NP-hard problem. This is the message of the following result:

**Proposition 4.1** *Given a non-deterministic finite automaton $\mathcal{A}$ and a finite sample $S$ of positive and negative examples, the problem whether there exist clock guards which can be added to the automaton $\mathcal{A}$ such that the resulting event-recording automaton accepts all the positive and none of the negative timed examples in $S$ is an NP-hard problem.*

**Proof** *(sketch)* We use a reduction from the NP-complete 3-SAT problem. Let $(U = \{u_1, \dots, u_n\}, C = \{c_1, \dots, c_m\})$ be a given 3-SAT instance. Each clause $c = \{x_1, x_2, x_3\}$ is transformed into a positive sample string $s_c \in S$ of the form

$$\sigma_1(a_1, t_{1,1})(a_1, t_{1,2})\sigma_2(a_2, t_{2,1})(a_2, t_{2,2})\sigma_3(a_3, t_{3,1})(a_3, t_{3,2}),$$

where, $a_i = atom(x_i)$ for all $i$, and in some fixed enumeration of $U$, $\sigma_1 = (u_1, 0) \dots (u_n, 0)$ and $\sigma_j = (u_1, t_{j-1,2}) \dots (u_n, t_{j-1,2})$ for $j = 2, 3$ and the time values $t_{i,j}$ are defined as follows: $t_{1,1} = 1$ if $x_1$ is a positive literal and 0 else, and for $j = 2, 3$ $t_{j,1} = t_{j-1,2} + 1$ and $t_{j,2} = t_{j,1} + 1$ if $x_j$ is a positive literal and $t_{j,1} = t_{j-1,2}$ and $t_{j,2} = t_{j,1}$ if $x_j$ is a negative literal.[4]

The set of negative examples consists of strings of the form $\sigma_1(u, 1)(u, 1)$ and $\sigma_1(u, 0), (u, 1)$ for each atom $u \in U$.

Figure 2 shows the automaton which is used in the clock guard finding problem. The idea behind this automaton is that each positive string has 3 possibilities of making it to the final state, one for each literal in the 3-SAT clause. Whether it reaches the final state is

[3]Assuming P$\neq NP$.

[4]$atom(x_i)$ is simply the atomic symbol in $U$ resulting from stripping any occurrence of the negation operator $\neg$ in $x_i$.

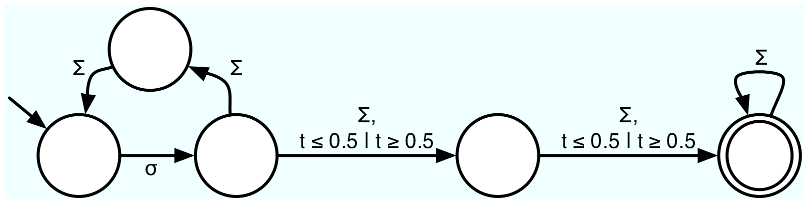

**Figure 2:** *The automaton used in the reduction from 3-SAT. The arc labelled with $\sigma$ is actually a link to a small sub-automaton in which there is one transition for each literal, and it connects back to the main automaton. This sub-automaton is used to reset all the clock values of the different literals. An arc labelled with $\Sigma$ is actually a set of arcs, one arc for each literal. The two arcs leading to the final state are the* selecting arcs. *These determine which literals (timed symbols) are needed to reach a final state. The two arcs leading back to the start state make sure that if a literal of a clause occurring in a positive example is not selected for the final state the next literal will be tried.*

determined by the clock guards on the selecting arcs. For each atom $u \in U$ a clock guard can either accept it by setting the guard to include 1 or refuse it by setting the guard to include 0. Note that the negative examples are constructed in such a way that the clock guards cannot include both 1 and 0. Hence, for each atom $u \in U$ its corresponding guard acts as a truth setting element, thus enabling all positive sample strings to be accepted while refusing every negative sample string iff the original 3-SAT instance is satisfiable. Hence, finding such guards is as hard as solving the satisfiability problem. □

This proof shows that adding the notion of time to a FA model significantly increases the difficulty of the problem, even when the resulting class of TAs is determinizable. Note that, when the resulting ERA is deterministic (as it is in the proof), we can verify the solution in polynomial time by walking the path (only one is possible) of each example. Hence the problem is NP-complete when we require the resulting ERA to be deterministic.

For the problem of learning a TA it is important to know where this hardness comes from. Knowing this would allow us to learn a restricted form of a TA (or to use a restricted form of learning) such that the problem becomes tractable. Using the same type of construction as the one in the proof above we can prove the following proposition:

**Proposition 4.2** *The problem of whether there exist clock guards which can be added to a non-deterministic finite automaton $\mathcal{A}$ such that the resulting (deterministic) event-recording automaton accepts all the positive and none of the negative timed examples remains NP-hard even when:*

- *$\mathcal{A}$ contains no loops.*
- *The alphabet of the language $L(\mathcal{A})$ is a unary language and Boolean combinations of guards are allowed.*
- *The timed symbols do not occur at exactly the same point in time and no Boolean combinations of guards are allowed.*

These results show us where the computational hardness does not come from. The problem is already hard when there is just a linear amount of examples, which are of linear length, and which use an alphabet of just one symbol. Also a constant number of clock guards already guarantees the problem to be hard if Boolean combinations of guards are allowed. The ERA does not have to contain loops. One of the properties the reduction does rely on is the fact that the given FA is non-deterministic. It is an interesting open question whether the adding of clock constraints to a DFA results in the same (or another) hardness result.

## 5 Conclusions and Future Work

We have described how the theory of TAs can be used in a real-time pattern detection problem. The model of a TA is intuitive and very powerful, and can therefore be learned from both data and from expert knowledge. Learning a TA from data, however, is a hard problem. An algorithm capable of solving the subproblem of learning just the clock guards is capable of solving any problem in NP. Finding classes of TAs of which the learning problem is of the same difficulty as learning FAs is subject to future work.

The way in which we intend to learn TAs is by using the learning by refinement framework, the foundations of which were by Laird [8]. Contrary to most FA learning algorithms, this framework allows us to include expert knowledge. The main idea is to use small refinements of the FA model in order to obtain subsets and supersets of the target language. In such a model, expert knowledge can be incorporated by constructing the starting FA model. By using small refinements, the idea is that the resulting FA will not deviate too much from this starting model, enabling experts to give a guideline for the learning algorithm.

## References

[1] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

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

[3] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computing*, 75:87–106, 1987.

[4] Christoc G. Cassandras and Stephane Lafortune. *Introduction to Discrete Event Systems.* Springer Verlag, 1999.

[5] Orlando Cicchello and Stefan C. Kremer. Inducing grammars from sparse data sets: A survey of algorithms and results. *Journal of Machine Learning Research*, 2003(4):603–632, 2003.

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

[7] Olga Grinchtein, Bengt Jonsson, and Martin Leucker. Inference of timed transition systems. *Electronic Notes in Theoretical Computer Science*, 2005.

[8] Philip D. Laird. *Learning from good and bad data.* Kluwer Academic Publishers, 1988.