Learning Mealy Machines with Timers

Bengt Jonsson
Department of Information Technology, Uppsala University

Frits Vaandrager
Department of Software Science, ICIS, Radboud University, Nijmegen

Abstract
We introduce a new model of Mealy machines with timers (MMTs), which is able to describe the timing behavior of a broad class of practical systems, and sufficiently restricted for active learning algorithms. We present a natural extension of Angluin’s active learning algorithm, which employs sequences of inputs with precise timing. Our algorithm is based on three key results: (i) an untimed semantics for MMTs, which is equivalent to the natural timed one (ii) a Nerode congruence based on the untimed semantics, and (iii) an active automata learning algorithm which is based on approximating this Nerode congruence. This algorithm allows to learn MMTs using a number of membership and equivalence queries, which is polynomial in the number of states of the resulting MMT, and doubly exponential in the maximal number of simultaneously active timers.

ACM Reference Format:

1 Introduction
Active automata learning aims to construct black-box state diagram models of software and hardware systems by providing inputs and observing outputs. In 1987, Angluin [1] published a seminal paper in which she showed that finite automata can be learned using so-called membership queries and equivalence queries. Many (if not most) efficient active learning algorithms used today are designed following Angluin’s approach of a minimally adequate teacher (MAT). In this approach, learning is viewed as a game in which a learner has to infer the behavior of an unknown state diagram by asking queries to a teacher. Following pioneering work by [1, 15, 18, 24, 25], active automata learning is emerging as an effective bug finding technique. Using active automata learning, for instance, standard violations have been found in many implementations of major network and security protocols such as TLS [7], TCP [10, 11] and SSH [12].

Timing often plays a crucial role in these applications. A TCP implementation, for instance, may retransmit packets if they are not acknowledged within a specified time. Also, a timeout may occur if the implementation does not receive an acknowledgment after a number of retransmissions, or if it remains in certain states too long. Timing behavior cannot be captured using existing learning tools, which only support learning of deterministic Mealy machines and related untimed models. In the case of TCP, previous work only succeeded to learn Mealy machine models by having the network adaptor ignore all retransmissions, and by completing learning queries before the occurrence of certain timeouts [11]. All timing issues had to be artificially suppressed.

There has been some work on algorithms for learning timed systems, e.g., [4, 13, 14, 28]. The modeling frameworks of [4, 28] are very restrictive however: they essentially allow only a single timer, which is reset on every transition, thus allowing to represent only constraints on delays between successive transitions. In contrast, the event recording automata studied by [13, 14] appear to have too many degrees of freedom, leading to prohibitively complex algorithms. In the literature, there is no tractable algorithm which can learn models that capture timing behavior of common network protocols.

In this paper, we address this challenge by presenting a framework for learning timed extensions of automata models that appear to be sufficiently expressive to describe the real-time behavior of network protocols such as TLS, TCP and SSH. Our work is inspired by the results of [4] on time delay Mealy machines, but we focus on a significantly richer class of automata models. We introduce the class of Mealy machines with timers (MMTs) that is able to model the timing behavior of a wide variety of communication protocols. Timers are set to integer values in transitions, and may be stopped or time out in later transitions. MMTs can be viewed as a formalization of the finite state models with countdown timers that are used in the textbook of Kurose and Ross [19] to explain transport layer protocols. Figure 1 presents an MMT model of the sender from the alternating-bit protocol, adapted from [19, Figure 3.15]. In the diagram $x := 3$ denotes that a transition starts a timer $x$ with value 3, and $\text{stop}(x)$ denotes that timer $x$ is stopped. For readability we have omitted trivial self-loops. A denotes the absence of an observable output. The approaches of [4, 28] cannot model this protocol.

We present a natural timed adaptation of the MAT framework for learning MMTs. We base it on the set of timed words of an MMT, which record possible sequences of inputs and outputs and their precise timing. In an MMT, each timeout immediately triggers an observable output, hence we can observe the occurrence of a timeout indirectly. However, we cannot observe which timer times out.

Figure 1. MMT model for alternating-bit protocol sender
out. In a membership query, the learner supplies a sequence of inputs with precise timing. In response, the teacher specifies outputs occur in response to these inputs, as well as their precise timing. Via an equivalence query, the learner asks whether a hypothesis MMT that it has constructed accepts the same timed words as the (unknown) MMT of the teacher. If this is the case, the teacher’s answer is ‘yes’; otherwise it is ‘no’ coupled with a timed word showing that the hypothesis is incorrect.

The timed MAT framework naturally corresponds to a setting, where the behavior of a black-box protocol component is investigated by supplying inputs. However, it is not convenient for formulating model learning algorithms. We therefore develop an alternative semantics for MMTs, based on sets of untimed words, which do not record timing of inputs and outputs, but instead record when and how timers are set and when they expire. We show, perhaps surprisingly, that under certain mild restrictions (the most important being that each transition sets at most one timer) this untimed semantics is equivalent to the above timed semantics. The result shows that the set of timed words can be inferred from the set of untimed words and vice versa.

The correspondence between the timed and untimed semantics suggests an architecture for our active learning algorithm that is shown in Figure 2. The idea is to define an alternative (and simpler) MAT framework in which the learner uses membership queries (MQ) and equivalence queries (EQ) to obtain information about the untimed behaviors of an MMT. An adapter then implements each untimed membership query via a series of timed membership queries for the timed teacher, and each untimed equivalence query via a timed equivalence query. The timed setting (represented in red) is suitable to represent practical learning scenarios, whereas the untimed setting (represented in blue) is suitable for formulating automata learning algorithms.

In order to develop a learning algorithm for the untimed MAT framework, we first develop a Nerode equivalence, which generalizes the standard Nerode equivalence for regular languages, and induces the canonical MMTs that will be constructed by the learning algorithm. We also develop an approximation of this Nerode equivalence as a basis for learning algorithms. We also develop an approximation of this Nerode equivalence, which generalizes the standard Nerode equivalence for regular languages, and induces the canonical MMTs that will be constructed by the learning algorithm. We also develop an approximation of this Nerode equivalence as a basis for learning algorithms.

In Section 2, we present the definition of MMTs, their timed semantics, and a minimally adequate teacher for MMTs. Section 3 presents the untimed semantics, and the equivalence with the timed semantics. Section 4 describes the untimed MAT framework for MMTs and the adapter that implements an untimed teacher using a timed teacher. Sections 5 and 6 present the Nerode equivalence and its approximated version, and Section 7 presents our learning algorithm. Section 8 contains some concluding remarks and lists topics for future research. Proofs are in the appendix.

**Related Work** Previous work on learning timed automata models have either been very complex, and not suitable as a basis for practical implementation [13, 14], one reason being that they aim to learn rather general classes of models, or target rather restricted models, e.g., allowing to capture only delays between consecutive transitions [27, 28]. There are also algorithms for learning timed models from white-box components, whose internal “code” can be inspected [21] and whose internal state can be inspected during execution [22].

Other generalizations of classical automata learning include algorithms for learning of symbolic automata [9, 23] or register automata [2, 6, 16, 17]. These models can capture simple relations, such as equality and ordering, between data parameters of inputs and outputs, but cannot capture how timers operate. The techniques for learning them cannot be applied to MMTs, although our treatment of unknown timer names has been inspired by their handling of unknown registers names. The treatment of timer names in our Nerode equivalence has also been inspired by treatment of registers in corresponding equivalences for register automata [5].

**2 Mealy machines with timers**

**2.1 Definition and timed semantics**

We assume an infinite set \( \mathcal{X} = \{x_1, x_2, x_3, \ldots \} \) of *timers*. Let \( \text{TO}(\mathcal{X}) \) be the set of *timeout events* of the form \( \text{to}(x) \) for \( x \in \mathcal{X} \). For a set \( \mathcal{I} \), let \( \mathcal{I} \cup \text{TO}(\mathcal{X}) \). We view (partial) functions as sets of pairs. We write \( \mathcal{A} \equiv \mathcal{B} \) for the set of partial functions from \( \mathcal{A} \) to \( \mathcal{B} \), and \( f \upharpoonright \mathcal{A} \) for the restriction of function \( f \) to \( \text{dom}(f) \cap \mathcal{A} \). We write \( \mathcal{P}_{\text{fin}}(\mathcal{A}) \) for the set of finite subsets of \( \mathcal{A} \).

**Definition 2.1.** A *Mealy machine with timers* (MMT) is a tuple \( \mathcal{M} = (I, O, Q, q_0, \mathcal{X}, \delta, \lambda, \pi) \), where

- \( I \) and \( O \) are finite sets of input events and output events, respectively, with \( I \cap \text{TO}(\mathcal{X}) = \emptyset \),
- \( Q \) is a finite set of states, with \( q_0 \in Q \) the initial state,
- \( \mathcal{X} : Q \to \mathcal{P}_{\text{fin}}(\mathcal{X}) \), with \( \mathcal{X}(q_0) = \emptyset \),
- \( \delta : Q \times \mathcal{I} \to Q \) is a transition function,
- \( \lambda : Q \times \mathcal{I} \to O \) is an output function,
- \( \pi : Q \times \mathcal{I} \to (\mathcal{X} \cup \mathcal{N}^{>0}) \) is a timer update function.

![Figure 2. Learning architecture](image-url)
Let $q \in Q$, $i \in I$, $q' = \delta(q,i)$ and $\rho = \pi(q,i)$. We require that $\text{dom}(\rho) = X(q')$ and $\text{ran}(\rho) \subseteq X(q) \cap \mathbb{N}^0$. When a timer expires it is stopped: if $i = t\in X$, for some $x$, then $x \notin \text{ran}(\rho)$. We require that input events are always enabled and timeout events are enabled for timers that are active in the current state: $\delta(q,i),(\lambda,q,i)$ and $\pi(q,i)$ are defined iff either $i \in I$ or $i = t\in X$, for some $x \in X(q)$. We write $q \xrightarrow{i/o,\rho} q'$ if $\delta(q,i) = q'$, $\lambda(q,i) = o$ and $\pi(q,i) = \rho$.

Update function $\pi$ determines how timers are affected when an event occurs. Suppose $q \xrightarrow{i/o,\rho} q'$. If $x \in X(q) \setminus \text{ran}(\rho)$ then we say that input $i$ stops timer $x$. If $x \in X(q')$ and $\rho(x) \in \mathbb{N}^0$ then $i$ starts timer $x$ with value $\rho(x)$. Finally, if $x \in X(q')$ and $\rho(x) \in X(q)$ then we say that input $i$ renames timer $\rho(x)$ to $x$.

**Semantics.** We define the semantics of an MMT $M = (I, O, Q, q_0, X, \delta, \lambda, \pi, \text{Val})$ via an infinite state transition system that describes all possible configurations and transitions between them. A **valuation** is a partial function $\kappa : X \rightarrow \mathbb{R}^\geq 0$, defined on a finite subset of $X$, that assigns nonnegative real numbers as values to timers. We write $\text{Val}(Y)$ for the set of valuations with domain $Y \subseteq X$. A configuration of an MMT is a pair $(q, \kappa)$, where $q \in Q$ is a state and $\kappa \in \text{Val}(X(q))$ is a valuation. The initial configuration is the pair $(q_0, \kappa_0)$, where $\kappa_0$ is the empty function. Valuations and configurations can be modified by the occurrence of input and timeout events, and by the occurrence of delays. If $x$ is a valuation then all timers have a value of at least $d$, then $d$ units of time may pass. As a result of this delay the value of all the timers is decremented by $d$. Formally, for $\kappa, \kappa'$ valuations and $d \in \mathbb{R}^\geq 0$, we define a delay transition relation by:

\[
\kappa \xrightarrow{d} \kappa' \text{ iff } \quad \forall x \in \text{dom}(\kappa) : \kappa'(x) = \kappa(x) - d.
\]

If the current valuation is $\kappa$, then timeout event $t\in X$ may occur only if $\kappa(x) = 0$. If an input or a timeout event occurs, $\kappa$ is updated as specified by update function $\rho$. Let $i$ denote the embedding of $I \cap \mathbb{N}^0$ in $\mathbb{R}^\geq 0$. Then, for $\kappa, \kappa'$ valuations, $i \in I, o \in O$ and $\rho \in X(q_0, \mathbb{N}^0)$, we define a discrete transition relation by $\kappa \xrightarrow{i/o,\rho} \kappa'$ iff

\[
\text{dom}(\rho) = \text{dom}(\kappa') \land \text{ran}(\rho) \subseteq \text{dom}(\kappa) \cup \mathbb{N}^0 \land \\
\kappa' = (\kappa + \rho) \land \\
\forall x \in \text{ran}(\rho) : \kappa'(x) = (x - \rho(x))^+.
\]

**Transition relations $\rightarrow$ and $\xrightarrow{d}$** can be lifted to configurations. For all configurations $(q, \kappa), (q', \kappa')$ of MMT $M$,

\[
\begin{align*}
q \xrightarrow{d} (q', \kappa') \quad &\text{ iff } \\
(q, \kappa) \xrightarrow{d} (q', \kappa').
\end{align*}
\]

A **timed run** of $M$ over $w$ is a sequence $\alpha = C_0 \xrightarrow{d_1} i_{1/o} \xrightarrow{i_{1/o}} C_1 \xrightarrow{d_2} \cdots \xrightarrow{d_k} \cdots \xrightarrow{d_k} i_{k/o} \xrightarrow{i_{k/o}} C_k$ of transitions between configurations $C_j, C'_j$ of $M$, where $C_0$ is the initial configuration. A **timed word over inputs $I$ and outputs $O$** is a sequence $w = d_1 i_1 o_1 d_2 i_2 o_2 \cdots d_k i_k o_k$, where $d_j \in \mathbb{R}^\geq 0$, $i_j \in I \cup \{t\}$, and $o_j \in O$. To each timed run $\alpha$ we associate a **timed word** by forgetting the configurations and the timers in timeout events:

\[
tw(\alpha) = d_1 i_1' o_1 d_2 i_2' o_2 \cdots d_k i_k' o_k,
\]

where for all $j$,

\[
i'_j = \begin{cases} i_j & \text{if } i_j \in I, \\ t & \text{if } i_j \in T \cup X. \end{cases}
\]

The idea is that timeouts cannot be observed directly. However, when we observe an output that is not triggered by an input, we may conclude that a timeout occurred. But in general we do not know which timer expired.

We say $w$ is a timed word of $M$ if $M$ has a timed run $\alpha$ with $w = tw(\alpha)$. Two MMTs $M$ and $\tilde{M}$ with the same sets of inputs are **timed equivalent**, denoted $M \equiv_{\text{timed}} \tilde{M}$, if they have the same sets of timed words.

**Remark.** In our semantics we assume that discrete transitions occur instantaneously and take no time. In applications, however, $\beta$ interactions often take a significant amount of time, see e.g. [26]. If it is important to model such delays explicitly, this can be done within the MMT framework by splitting a transition with input $i$ and output $o$ into a pair of consecutive transitions: a first transition with input $i$ and some default output $A$ that starts a timer $x$, and a second transition in which $x$ times out and output $o$ is produced. If inputs arrive in the newly introduced intermediate state, these may either be ignored, buffered or forbidden (via a transition to some designated error state). Note that $A$ corresponds to the absence of an observable output. For inputs $i \in I$ this is fine: if such an input does not trigger an observable output then we just assume that the output event is $A$. We do not allow $A$ as output event in timeout transitions $t\in X$: since timeout events themselves are not observable, we can only observe their occurrence indirectly through the observable output that they trigger.

**Remark.** Note that we assume that in a timed run each discrete transition is preceded by a nonzero delay. The idea that multiple consecutive discrete transitions may occur in zero time is a useful abstraction in synchronous programming and in the theory of timed automata, but nonzero delays form a crucial requirement for the learning algorithm that we present in this paper.

Disallowing zero delays creates certain complications that we have to deal with. The simple MMT of Figure 3, for instance, may reach a timed following the timed word $1 \cdot i \cdot o \cdot 1 \cdot o$ at this point timer $x$ has value $0$, but no timeout is enabled since first a nonzero amount of time has to elapse, which is not possible since then $x$ would become negative. Our learning algorithm will only explore

\[
\text{start} \xrightarrow{1/o,x=1} q_0 \xrightarrow{t\in X} q_1 \xrightarrow{i/o} q_2 \xrightarrow{i/o} q_3.
\]

**Figure 3.** An MMT with a timelock

runs in which no “races” occur. This eliminates the above timeout, which is cause by a race between an $i$ event and a timeout.

**Experiments.** We may perform experiments on an MMT in which we provide a series of inputs at specific times, and observe the outputs that occur in response to these inputs. An experiment can be formally described by a **timed input word**: a sequence $u = d_1 i_1 \cdots d_k i_k d_{k+1}$, where $d_j \in \mathbb{R}^\geq 0$ and $i_j \in I$, for all $1 \leq j \leq k$, and $d_{k+1} \in \mathbb{R}^\geq 0$. We may associate a timed input word $t\in X$.
to each timed word \( w \) by removing the outputs events, removing the occurrences of \( to \) by replacing consecutive numbers by their sum, and possibly placing 0 at the end of the sequence. Thus, for instance, \( tiv(7 i 0 i 1 i 0 \to o') = 7 i 1 i 1 \) and \( tiv(3 i 1 \to o 1.1 i 2 \to o 2 \to o 3.2 \to o 4.3 i 4 \to o 0) = 3 i 1.1 i 2 \to 4.1 i 4 \to o 0 \). If \( u \) and \( u' \) are timed input words, then we write \( u \preceq u' \) if \( u \) and \( u' \) are equal, except that the final delay of \( u \) is less or equal than the final delay of \( u' \). For any timed input word \( u \) over \( I \), there exists a maximal timed word \( w \) of \( M \) such that \( tiv(w) \preceq u \). We call \( w \) an outcome of running experiment \( u \) on \( M \). For instance, if we perform the experiment \( 1 \in i 0 1 \to o' \) in the MMT of Figure 1, then the unique outcome is \( 1 \in send0 3 \to send0 3 \to send0 \).

**Non-determinism.** Since we cannot observe the identity of a timer in a timeout event, experiments do not always have a unique outcome and MMTs may exhibit non-deterministic behavior. For the MMT of Figure 4, for instance, experiment \( 1 \in i 0 1 \to o' \) and \( 1 \in i 0 1 \to o'' \) have outcomes \( 1 \in i 0 1 \to o' \) and \( 1 \in i 0 1 \to o'' \). This type of non-determinism is “uncontrollable” and will not occur if we carefully select the timing of inputs.

**Further restrictions on MMTs.** Although learning of non-deterministic systems has been studied in the literature [29], non-determinism clearly is a major complication for learning algorithms. For this reason, we impose two additional restrictions on the timer update functions in the remainder of this article: for each transition \( i/o, \rho \), \( q \to q' \) of an MMT (a) \( \rho \) is injective, and (b) at most one timer is started by \( \rho \). Condition (b) rules out the non-determinism of Figure 4 (top). Condition (a) is needed to rule out a variation of this MMT in which timer \( z \) is started and then copied to distinct timers \( x \) and \( y \).

We call timed input word \( u \) transparent if the fractional parts of the absolute times of occurrence of all the inputs from \( I \) are different. A timed word \( w \) is transparent iff \( tiv(w) \) is transparent. It is easy to check that for each MMT \( M \) that satisfies the above conditions, each transparent experiment \( u \) has a unique outcome: since each timer is started at a different fractional time, and each timer expires after an integer amount of time, it is not possible that two timers expire simultaneously.

### 2.2 A timed MAT framework for learning MMTs

We now propose an instance of Angluin’s MAT framework for Mealy machines with timers. In our setting, illustrated in Figure 5, the teacher knows an MMT \( M \). Initially, the learner only knows the set \( I \) of inputs of \( M \). The learner may perform experiments (membership queries, MQ) to learn about the timed words of \( M \), and pose equivalence queries (EQ) to find out whether a constructed hypothesis is correct.

![Figure 5. A timed MAT framework](image)

**Membership queries.** With a membership query, the learner asks what the output is in response to a timed input word \( u \) over \( I \). The teacher answers with a maximal timed word \( w \) of \( M \) such that \( tiv(w) \preceq u \).

**Equivalence queries.** With an equivalence query, the learner asks if a hypothesized MMT \( H \) is correct, that is, whether \( H \approx_{\text{timed}} M \). The teacher answers yes if this is the case. Otherwise she answers no and supplies a counterexample: a transparent timed word \( w \) of \( M \) that is not a timed word of \( H \). (Lemma 3.14 asserts that such a timed word always exists when \( H \approx_{\text{timed}} M \).)

The main result of this paper is an algorithm that allows the learner to learn an MMT \( H \) that is timed equivalent to \( M \) via a finite number of membership and equivalence queries.

### 3 Untimed semantics

In this section, we present an untimed semantics for MMTs and prove that this is equivalent with the timed semantics. In order to define the untimed semantics, we need to define a number of abstractions of timed runs. Technically, we need renamings of timers in transitions of an MMT in order to prove the Myhill-Nerode Theorem 5.3. However, many definitions and proofs are easier to understand when timers are not renamed. For this reason we only consider update functions \( \rho \) that satisfy, for all \( x \): \( p(x) \in \mathbb{N}^{\geq 0} \) or \( p(x) = x \). In fact, since an update starts at most one timer, updates can be represented as either the empty set or a singleton set \( \{(x, n)\} \). Our results generalize to arbitrary, injective renamings.

### 3.1 Timed and untimed runs and behaviors

Configurations consist of pairs of states and timer valuations. This means that there are two natural abstractions of timed runs: an abstraction \( untim \) that forgets all timing information and keeps the transitions of the MMT, and an abstraction \( beh \) that forgets
information on states and preserves the timing information. When we compose these abstractions we obtain untimed behaviors in which only information about inputs, outputs, updates and active timers is preserved. The abstractions commute, beh untimed(α) = untimed(beh(α)), and play a key role in the technical development of this paper. Formally, an untimed behavior over inputs I and outputs O is a sequence

$$\beta = \gamma = X_0 \overset{i_0/o_0/p_0}{\longrightarrow} X_1 \overset{\ldots}{\longrightarrow} X_k,$$

where $$X_0 \subseteq X$$ and, for each $$j > 0$$, $$i_j \in I$$, $$o_j \in O$$, $$p_j \in X \leftrightarrow \mathbb{N}^+$$, and $$X_j \setminus X_{j-1} \subseteq \text{dom}(p_j) \subseteq X_j \subseteq X$$. Moreover, if $$i_j = to(x)$$, for some $$j > 0$$, then $$x \in X_{j-1}$$ and $$x \not\in X_j \setminus \text{dom}(p_j)$$. An untimed run of an MMT $$M$$ is a sequence

$$\gamma = q_0 \overset{i_0/o_0/p_0}{\longrightarrow} q_1 \overset{\ldots}{\longrightarrow} q_k,$$

of transitions of $$M$$ that starts with the initial state $$q_0$$. To each untimed run $$\gamma$$ we associate an untimed behavior by replacing all states by their sets of timers:

$$\text{beh}(\gamma) = X(q_0) \overset{i_0/o_0/p_0}{\longrightarrow} X(q_1) \overset{\ldots}{\longrightarrow} X(q_k).$$

We say that $$\beta$$ is an untimed behavior of $$M$$ if $$M$$ has an untimed run $$\gamma$$ with $$\text{beh}(\gamma) = \beta$$. Note that the initial timer set of an untimed behavior of $$M$$ is empty. A timed behavior over inputs I and outputs O is an alternating sequence

$$\sigma = \kappa_0 \overset{d_0}{\longrightarrow} \kappa_0 \overset{i_0/o_0/p_0}{\longrightarrow} \kappa_1 \overset{d_1}{\longrightarrow} \kappa_1 \overset{\ldots}{\longrightarrow} \kappa_{k-1} \overset{i_k/o_k/p_k}{\longrightarrow} \kappa_k$$

(1) of delay transitions and discrete transitions with, for each $$j$$, $$\kappa_j, \kappa_j'$$, values and, for each $$j > 0$$, $$d_j \in \mathbb{R}^+, i_j \in I$$, $$o_j \in O$$, and $$p_j \in X \leftrightarrow \mathbb{N}^+$$. To each timed behavior $$\sigma$$ we associate an untimed behavior by forgetting the delay transitions and by replacing values of transitions by their domain:

$$\text{untimed}(\sigma) = \text{dom}(\kappa_0) \overset{i_0/o_0/p_0}{\longrightarrow} \text{dom}(\kappa_1) \overset{\ldots}{\longrightarrow} \text{dom}(\kappa_k).$$

We say that untimed behavior $$\beta$$ is feasible if there exists a timed behavior $$\sigma$$ such that $$\text{untimed}(\sigma) = \beta$$.

We also associate a timed word to timed behavior $$\sigma$$ by forgetting values, timers, and update functions:

$$\text{tw}(\sigma) = d_1 i_1' o_1' t_1' d_2 o_2' \ldots d_k i_k' o_k',$$

where for each $$j$$,

$$i_j' = \begin{cases} i_j & \text{if } i_j \in I, \\ o_j & \text{if } i_j \in O. \end{cases}$$

Let $$\alpha$$ be a timed run of an MMT $$M$$:

$$\alpha = (q_0, \kappa_0) \overset{d_1}{\longrightarrow} (q_1, \kappa_1) \overset{i_1/o_1/p_1}{\longrightarrow} (q_1, \kappa_1) \overset{\ldots}{\longrightarrow} (q_k, \kappa_k).$$

Then $$\alpha$$ can be prject both to an untimed run of $$M$$

$$\text{untimed}(\alpha) = q_0 \overset{i_0/o_0/p_0}{\longrightarrow} q_1 \overset{\ldots}{\longrightarrow} q_k,$$

(the $$p_j$$'s are determined since $$M$$ is deterministic) and to a timed behavior

$$\text{beh}(\alpha) = \kappa_0 \overset{d_1}{\longrightarrow} \kappa_0 \overset{i_1/o_1/p_1}{\longrightarrow} \kappa_1 \overset{d_2}{\longrightarrow} \kappa_1 \overset{\ldots}{\longrightarrow} \kappa_{k-1} \overset{i_k/o_k/p_k}{\longrightarrow} \kappa_k.$$
An isomorphism from $\sigma$ to $\sigma'$ is a list $f = f_0, \ldots, f_k$ of bijections such that (1) $f$ is an isomorphism from $unte\text{m}(\sigma)$ to $unte\text{m}(\sigma')$, and $\lambda_j = k_j \circ f^{-1}$ and $\lambda'_j = k'_j \circ f^{-1}$, for all $j$. Since $\sigma'$ is fully determined by $\sigma$ and $f$, we write $\sigma' = f(\sigma)$. Two timed behaviors $\sigma$ and $\sigma'$ are isomorphic if there exists an isomorphism $f$ from $\sigma$ to $\sigma'$. Since an isomorphism only renames timers, which do not appear in timed words, isomorphic timed behaviors induce identical timed words: $\sigma' = f(\sigma) \Rightarrow tw(\sigma') = tw(\sigma)$.

The following lemmas then follow:

**Lemma 3.1.** Let $\sigma$ be a timed behavior and let $f$ be an isomorphism for $\sigma$. Then $unte\text{m}(f(\sigma)) = f(unte\text{m}(\sigma))$.

**Lemma 3.2.** If untimed behaviors $\beta$ and $\beta'$ are isomorphic, then $\beta$ is feasible iff $\beta'$ is feasible.

Two MMTs $M$ and $N$ with the same sets of inputs are untimed equivalent, denoted $M \simeq_{unte\text{m}} N$, iff their sets of feasible untimed behaviors are isomorphic. The following basic property will be needed later on:

**Lemma 3.3.** Suppose $M, N$ are MMTs with $M \not\simeq_{unte\text{m}} N$. Then there exists a feasible untimed behavior $\beta$ of $M$ that is not isomorphic to any feasible untimed behavior of $N$.

Untimed equivalence is finer than timed equivalence:

**Theorem 3.4.** $M \simeq_{unte\text{m}} N$ implies $M \simeq_{time} N$.

**Proof.** Assume $M \simeq_{unte\text{m}} N$ and $w$ is a timed word of $M$. Since $\simeq_{time}$ is symmetric, it suffices to prove that $w$ is a timed word of $N$. Since $w$ is a timed word of $M$, there exists a timed run $\sigma$ of $M$ with $tw(\sigma) = w$. Let $\sigma = beh(\sigma)$ and $\beta = unto\text{m}(\sigma)$. Then $\beta$ is a feasible untimed behavior of $M$ and $tw(\sigma) = w$. Since $M \simeq_{unte\text{m}} N$, there exists an isomorphism $f$ such that $f(\beta) = f(\sigma) = f(\beta)$. Hence $N$ has an untimed run $\gamma'$ such that $unte\text{m}(\gamma') = \beta'$. Let $\sigma' = f(\beta)$. By Lemma 3.1, $\sigma'$ is a timed behavior with $unte\text{m}(\sigma') = unto\text{m}(f(\sigma)) = f(unte\text{m}(\sigma)) = f(\beta)$. Since $beh(\gamma') = unto\text{m}(\sigma') = \beta'$, $N$ has a timed run $\alpha'$ that is the pullback of $\gamma'$. Let $\beta' = \alpha'$. Note that $tw(\alpha') = tw(\sigma') = tw(f(\sigma)) = tw(\sigma) = w$. Hence $w$ is a timed word of $N$, as required.

The converse of Theorem 3.4 does not hold. This is due to the fact that an MMT may have timers that are always stopped or restarted before they expire. Such “ghost” timers are visible in the untimed semantics but cannot be observed in the timed semantics. Figure 7 gives an example of an MMT with a ghost timer. The MMT is equivalent to the MMT obtained by omitting the update $y := 60$ on the transition from $q_1$ to $q_2$. We say that an MMT $M$ is timed live if, for each feasible untimed behavior $\beta$ and for each timer $t$ that is running after $\beta$, there exists an untimed behavior $\beta_t$ consisting of transitions that leave $y$ unaffected, except for the last one in which $y$ expires, and such that $\beta' \circ \beta_t$ is feasible. Clearly, the MMT of Figure 7 is not timed live, as there is no way to extend the feasible untimed behavior $0 \xrightarrow{i/o \cdot y = 60} \{x \mid y < 60\}$ to an untimed behavior in which timer $y$ expires.

### 3.3 Equivalence timed/unteimed semantics

We will show that the timed semantics and the untimed semantics coincide for timer live MMTs in which at most one timer is (re)started on each transition. However, in order prove this result we need to do some preparatory work.

**Figure 7.** MMT with ghost timer $y$.

Often it is possible to slightly change the timing of events in a timed behavior, while preserving the associated untimed behavior. Consider, for instance, a timed behavior $k_0 \xrightarrow{d_1} k'_0 i_1/o_1/p_1 k_1 \xrightarrow{d_2} k_2' i_2/o_2/p_2 k_3 \xrightarrow{d_3} \cdots$ that contains an $i_j$-transition that is not a timeout and does not (re)start any timer. We can then schedule this transition slightly earlier. More precisely, if $0 < e < d_j$ and $e' = d_j + d_{j+1} - e$ then we can find $\kappa, \kappa'$ such that $k_0 \xrightarrow{d_1} k'_0 \xrightarrow{i_1/o_1/p_1} \cdots \xrightarrow{e} \kappa' i_j/o_j/p_j k' \xrightarrow{e'} \cdots$ is a timed behavior with the same underlying untimed behavior.

We may also be able to wiggle the timing of timeouts and transitions that (re)start a timer, but here we have to be more careful. If we shift the timing of an input event by a small amount then we must also shift the timing of a subsequent timeout that is triggered by this input. In addition, if the timeout starts another timer then we also need to shift the timeout event that this timer induces, etc. Let us formalize these ideas. Consider a timed behavior $\sigma$ as in equation (1) with events $i_p$ and $i_q$ with $p < q$. Then we say that $i_p$ triggers $i_q$ if there exists a timer $x$ such that: (a) event $i_p$ starts $x$, (b) for all $p < r < q$, $x$ is unaffected by event $i_r$, and (c) $i_q \xrightarrow{t} o_x$. A block of $\sigma$ is a maximal subset of indices $B = \{p_1, \ldots, p_n\}$ such that $i_p$ triggers $i_{p_n}$, $i_{p_n}$ triggers $i_{p_1}$, etc. Note that the collection of blocks of $\sigma$ partitions the set of indices $\{1, \ldots, k\}$. We refer to this partition as $\Pi_\sigma$. We say that timed behavior $\sigma$ contains a race if there is some index $j > 0$ and some timer $x$ such that $k_j(x) = 0$ and $i_j \neq o_x$. The following lemma allows us to shift all events in a block simultaneously forward or backward by a small amount, under the condition that there are no races.

**Lemma 3.5.** Suppose $\sigma$ is a timed behavior as in equation (1) without races. Suppose $B = \{p_1, \ldots, p_n\}$ is a block of $\sigma$, and suppose that $e$ is a real number whose absolute is smaller than any nonzero number that occurs in $\sigma$, that is $|e| < \min(\{1 \leq j \leq k \mid d_j \cup \text{ran}(k'_j) \setminus \emptyset\})$.

Then there exists a timed behavior $\sigma' = \lambda_0 \xrightarrow{i_0/o_0/p_0} \lambda_1 \xrightarrow{i_1/o_1/p_1} \lambda_2 \xrightarrow{\lambda_3} \cdots \xrightarrow{\lambda_k} \lambda_{k+1}$ without races such that $unte\text{m}(\sigma') = unto\text{m}(\sigma')$, $k_0 = \lambda_0$, and $e_j = d_j + e$ if $j-1 \not\in T \land j \in T$, $e_j = d_j$ if $j \not\in T \Rightarrow j \in T$, and $e_j = d_j - e$ if $j-1 \in T \land j \not\in T$.

In the presence of races, Lemma 3.5 does not hold. Consider the following timed behavior with blocks $\{1, 3\}, \{2\},$ and $\{4, 5\}$, visualized in Figure 8:

$$
\emptyset \xrightarrow{2} 0 \xrightarrow{i/o \cdot x = 2} (x = 2) \xrightarrow{1} (x = 1) \xrightarrow{i_2/o_2/y = 1} (x = y = 1)
$$
This timed behavior contains two races, after two and four time units, respectively. The first race is won by block \([1, 3]\), and the second race is won by block \([4, 5]\). As a result of these races, we cannot wiggle the timing of block \([1, 3]\) by any amount: if \(i_1\) occurs just a bit later then timer \(y\) must expire before timer \(x\), and if \(i_1\) occurs just a bit earlier then timer \(u\) must expire before timer \(z\). Note that when timer \(x\) expires timer \(y\) is stopped. This scenario is similar to the well-known Rush Hour puzzle game, in which one has to slide blocking vehicles out of the way to find a path for one specific red car to exit a parking lot. The next lemma asserts that we can always solve the puzzle for MMTs: for any timed behavior that contains races, an equivalent timed behavior exists without races. We may for instance slightly modify the timed behavior of Figure 8 by scheduling block \([2]\) a bit later and block \([4, 5]\) a bit earlier. Then all races have been eliminated and we can wiggle block \([1, 3]\).

![Figure 8: A timed behavior with two races](image)

**Lemma 3.6.** Let \(\sigma\) be a timed behavior. Then there is a timed behavior \(\sigma'\) without races s.t. untimed(\(\sigma\)) = untimed(\(\sigma'\)).

**Proof.** (Sketch) Let

\[
\sigma = \kappa_0 \xrightarrow{d_1} \kappa_0 \xrightarrow{i_1/o_1/p_1} \kappa_1 \xrightarrow{\cdots} \kappa_{\ell-1} \xrightarrow{i_{\ell}/o_{\ell}/p_{\ell}} \kappa_{\ell}
\]

be a timed behavior. For each index \(j\), each timer in the domain of \(\kappa_j\) has been started by some preceding event. Let \(\text{startedby}_{\gamma,j}: \text{dom}(\kappa_j) \to \Pi_\sigma\) be the function that maps each timer in the domain of \(\kappa_j\) to the block that contains the event that started this timer. Suppose that \(\sigma\) contains a race, that is, there is an index \(j > 0\) and a timer \(x\) such that \(\kappa_j'(x) = 0\) and \(i_j \neq \text{to}(x)\). Let \(B \in \Pi_\sigma\) be the block containing \(x\). Then we say that block \(B\) is the winner of the race and block \(\text{startedby}_{\gamma,j}(x)\) is a loser. Note that whenever there is a race at \(j\), this race has a single winner but it may have several losers. Moreover, each block can be loser in at most one race. A block that does not win any race can be wiggled forward by a small amount (if you don’t win you might as well start later). If block \(B\) wins a race from block \(B'\), then \(\max(B) > \max(B')\), that is, \(B\) contains an event that occurs later in \(\sigma\) than any event of \(B'\). This implies that the winning relation induces a partial order on the blocks of \(\Pi_\sigma\). Now consider the bottom elements in this partial order. These blocks do not win any race so we may wiggle them forward. Once we have eliminated the bottom elements we can work our way upwards in the partial order and wiggle all blocks forward one by one until no more races remain.

In a timed behavior, there is always an integer amount of time between events from the same block. This means that, if we consider the absolute time at which events occur, the fractional part of the absolute times of occurrence is the same for all events in a block. A timed behavior \(\sigma\) is transparent if tw(\(\sigma\)) is transparent. This implies that the fractional part of the absolute times of events from different blocks is different, and there are no races.

**Lemma 3.7.** For each feasible untimed behavior \(\beta\) there exists a transparent timed behavior \(\sigma\) such that \(\beta = \text{untimed}(\sigma)\).

**Proof.** Let \(\beta\) be a feasible untimed behavior. Then there exists a timed behavior \(\sigma\) such that \(\beta = \text{untimed}(\sigma)\). By Lemma 3.6, we may assume that \(\sigma\) contains no races. By repeated application of Lemma 3.5, we can wiggle the timing of all the blocks to make \(\sigma\) transparent. □

The following fundamental lemma now follows:

**Lemma 3.8.** Suppose \(\beta\) is a feasible untimed behavior that ends with timer set \(Y\), and \(\sigma_{i/o/p} \to Y'\) is an untimed behavior with \(i \in I\). Then \(\beta_{i/o/p} \to Y'\) is a feasible untimed behavior.

**Proof.** By Lemma 3.7, there exists a transparent timed behavior \(\sigma\) such that \(\beta = \text{untimed}(\sigma)\). Since \(\sigma\) is transparent, the last valuation of \(\sigma\) assigns a positive value to all timers in \(Y\). Thus we may extend \(\sigma\) by a small delay transition, followed by a discrete transition corresponding to \(Y\). This implies that untimed behavior \(\beta_{i/o/p} \to Y'\) is feasible. □

**Causality maps.** Consider an untimed behavior

\[
\beta = X_0 \xrightarrow{i_{1}/o_{1}/p_{1}} X_1 \xrightarrow{i_{2}/o_{2}/p_{2}} X_2 \cdots \xrightarrow{i_{k}/o_{k}/p_{k}} X_k.
\]

A causality map for \(\beta\) is a function that specifies, for each timer that expires, the index of the event that triggered this timeout. Formally, let \(\mathcal{T} = \{ j | i_j \in \text{to}(X) \}\) be the set of indices of \(\beta\) corresponding to a timeout. A causality map for \(\beta\) is a function \(c: \mathcal{T} \to \{1, \ldots, k\}\) that assigns to each index \(j\) with \(i_j = \text{to}(x)\) an index \(l < j\) such that \(p_l\) starts \(x\), and all events in between \(i_j\) and \(i_l\) do not affect \(x\).

**Lemma 3.9.** Each untimed behavior \(\beta\) that starts with the empty set of timers has a unique causality map \(c\).

We say that \(c\) is a causality map of a timed behavior \(\sigma\) if it is a causality map of \(\text{untimed}(\sigma)\), and we say that \(\sigma\) is a causality map of a timed run \(\alpha\) if it is a causality map of \(\text{beh}(\alpha)\). Lemma 3.9 implies that each timed run of an MMT has a unique causality map \(c\).

Consider a timed word \(w = d_1 i_1 o_1 d_2 i_2 o_2 \cdots d_k i_k o_k\). We want to know, for each timeout event in \(w\), by which event this timeout is triggered. Let \(\mathcal{T} = \{ j | i_j = \text{to} \}\) be the set of indices corresponding to a timeout. A causality map for \(w\) is a function \(c: \mathcal{T} \to \{1, \ldots, k\}\) that satisfies three conditions: (1) \(c\) is injective (at most one timer is started on each transition), (2) for all \(j, c(j) < j\) (a timeout is triggered by an earlier event), and (3) for all \(j, \sum_{i=j+1}^{\ell} d_i\) is an integer (timers expire after an integer delay).

**Lemma 3.10.** Suppose \(\alpha\) is a timed run of an MMT and \(c\) is the causality map of \(\alpha\). Then \(c\) is a causality map of tw(\(\alpha\)).

In general, a timed word may have multiple causality maps. However, we have the following lemma. Call a timed word transparent if the fractional part of the absolute times of all input events in \(I\) is different.

**Lemma 3.11.** Suppose \(\alpha\) is a timed run of an MMT, and \(w = \text{tw}(\alpha)\) is transparent. Then \(w\) has a unique causality map.
Each timed word of MMT $M$ with a unique causality map has a unique timed run that corresponds to it. The causality map tells us which timers time out during the trace, so we have complete information about the sequence of events that occurs.

A timed run of $M$ is fully determined by the sequence of time delays and events that occurs.

**Lemma 3.12.** Suppose $w$ is a timed word of MMT $M$ with a unique causality map. Then there is a unique timed run $\alpha$ of $M$ such that $w = tw(\alpha)$.

We are now prepared to state our first main result:

**Theorem 3.13.** Suppose that $M$ and $N$ are timer live MMTs. Then $M \approx_{\text{timed}} N$ implies $M \approx_{\text{untimed}} N$.

**Proof.** Suppose that $M \approx_{\text{timed}} N$. Let $\beta$ be a feasible untimed behavior of $M$. For reasons of symmetry, it suffices to prove that $N$ has a feasible untimed behavior $\beta'$ that is isomorphic to $\beta$.

Since $\beta$ is a feasible untimed behavior of $M$, $M$ has an untimed run $\gamma$ with $beh(\gamma) = \beta$. By Lemma 3.7, there exists a transparent timed behavior $\sigma$ such that $\beta = untime(\sigma)$. There exists a unique timed run $\alpha = pullback(\gamma, \sigma)$ of $M$ with $untime(\alpha) = \gamma$ and $beh(\alpha) = \sigma$. Thus $\sigma$ is a timed behavior of $M$. Let $w = tw(\alpha)$. Then $w$ is a transparent timed word of $M$. By Lemma 3.11, $w$ has a unique causality map $c$. Since $M \approx_{\text{timed}} N$, $w$ is also a timed word of $N$. By Lemma 3.12, $N$ has a unique timed run $\alpha'$ such that $w = tw(\alpha')$. Let $\beta' = untime(beh(\alpha'))$. Then $\beta'$ is a feasible untimed behavior of $N$. Note that the mappings $tw$, $untime$ and $beh$ all preserve the number of events, the sequence of inputs that occur (except for the names of the timers in timeouts), and the sequence of outputs. Thus $\beta$ and $\beta'$ have the same length, the same inputs (except for the timer names), and the same outputs. Moreover, by Lemmas 3.9 and 3.10, $\beta'$ and $\beta$ have the same causality map $c$.

By induction on the number of events in $\beta$ and $\beta'$, we prove that they are isomorphic. Since $\approx_{\text{untimed}}$ is symmetric, this suffices to prove the theorem.

**Lemma 3.14.** Suppose $N, M$ are timer live MMTs with $M \not\approx_{\text{timed}} N$. Then there exists a transparent timed word $w$ of $M$ that is not a timed word of $N$.

In the remainder of this article, we only consider timer live MMTs, which means we may assume equivalence of the timed and the untimed semantics.

### 4 From timed to untimed learning

In this section, we consider another instance of Angluin’s MAT framework. Again, the teacher knows an MMT $M$ but now the learner poses membership queries to learn the untimed behaviors of $M$, and if an equivalence query fails then the counterexample is an untimed behavior of $M$. We will show how an untimed teacher for MMTs can be implemented using a timed teacher.

#### 4.1 An untimed MAT framework for MMTs

Given the equivalence of the timed and the untimed semantics of MMTs, it is natural to also consider an untimed setting, in which a learner tries to construct an MMT based on membership queries for untimed behaviors.

In the untimed framework, the teacher does not reveal the names of the timers of $M$. It brings untimed behaviors into a “canonial” form, so that a learner can only observe these behaviors up to isomorphism. Let us formalize this idea. Suppose $\beta$ be an untimed behavior in which transitions update at most one timer. We say that $\beta$ is in canonical form if, for each $j$, the timer that is updated in the $j$-th event (if any) is equal to $x_j$. For each untimed behavior $\beta$ in which transitions update at most one timer, there is a unique untimed behavior $\beta'$ in canonical form that is isomorphic to $\beta$. We write $can(\beta)$ to denote this $\beta'$.
Consider an untimed behavior $\beta$ in canonical form:

$$\beta = X_0 \xrightarrow{i_0/o_0} X_1 \cdots \xrightarrow{i_k/o_k} X_k.$$  

Let $1 \leq l \leq j \leq k$, $d \in \mathbb{N}^{>0}$ and suppose that (1) $\rho_l$ either starts no timer or sets $x_l$ to $d$, (2) $\beta$ does not contain a timeout event $t_0[x_l]$. Then $\textit{adtime}(l, d, \beta)$ is the untimed behavior obtained from $\beta$ by replacing $\rho_l$ by the update $x_l := d$, and adding $x_l$ to the sets $X_l$ up to (and including) $X_j$. Write $\beta \subseteq \beta'$ if $\beta'$ can be obtained from $\beta$ by zero or more applications of function addtimer. Observe that $\subseteq$ is a partial order with as minimal elements untimed behaviors in which every timer that is started also times out. We call such untimed behaviors lean. We say that $\beta$ is a partial untimed behavior of $\textit{MMT}$ if $\beta$ is an untimed behavior $\beta'$ such that $\beta \subseteq \beta'$.

An untimed input word over $I$ is a sequence $u = i_1 \cdots i_k$ over $I$ such that (1) for all indices $j, l, t_j = t_0[x_j]$ implies $l < j$, and (2) each timer occurs at most once in a timeout event. The first condition says that if a timer expires it must have been set in a previous event, and the second condition expresses that each timer may expire at most once. Let $\beta$ be an untimed behavior in canonical form. We can associate a unique input word $uiw(\beta)$ to $\beta$ by removing all the output events, updates, and timer sets from $\beta$.

Our untimed learning setup, illustrated in Figure 9, is similar to the timed setup: again the teacher knows an MMT $\mathcal{M}$ and the learner initially only knows the sets of inputs of $\mathcal{M}$. Again, the learner may pose membership and equivalence queries. But now a membership query consists of an untimed input word $u$ over $I$. The teacher replies with a maximal feasible partial untimed behavior $\beta$ of $\mathcal{M}$ (in canonical form) such that $uiw(\beta)$ is a prefix of $u$. With an equivalence query, the learner asks if a hypothesis $\mathcal{H}$ with inputs $I$ is correct. Upon receiving $\mathcal{H}$, the teacher answers yes if $\mathcal{H} \equiv_{\text{untimed}} \mathcal{M}$. Otherwise it answers no and supplies a counterexample, which now is a feasible untimed behavior $\beta$ (in canonical form) that is a partial untimed behavior of $\mathcal{M}$ but not of $\mathcal{H}$ (by Lemma 3.3 such a counterexample exists).

### 4.2 Zones and Constraints

In order to implement an untimed teacher using a timed teacher, we need a basic machinery to determine whether an untimed behavior is feasible, i.e., whether it can be derived from a timed word. This can be done using well-established techniques for manipulating difference-bound matrices (DBMs) that are in standard use for analyzing timed automata [3, 8]. In this section, we describe an adaptation of such techniques to our setting.

Let $\beta$ be an untimed behavior in canonical form:

$$\beta = \emptyset \xrightarrow{i_0/o_0} X_1 \cdots \xrightarrow{i_k/o_k} X_k.$$  

We would like to determine for which values $d_1, \ldots, d_{k+1}$ a corresponding timed behavior

$$k_0 \xrightarrow{d_1} k'_0 \xrightarrow{i_0/o_0} \cdots \xrightarrow{d_k} k'_k \xrightarrow{i_k/o_k} k_{k+1}$$  

is possible. We solve this problem by producing a constraint over the values $d_1, \ldots, d_{k+1}$. In order to use DBM techniques, we first change representation by letting $t_j = d_1 + \cdots + d_j$ for $j = 1, \ldots, k+1$. Intuitively, $t_j$ is the time of occurrence of the $j$th transition in $\beta$. We now associate with $\beta$ a conjunction of difference constraints over $t_1, \ldots, t_{k+1}$, denoted $\text{constr}(\beta)$, consisting of

- for each index $j$ with $1 \leq j \leq k$: $0 < t_{j+1} - t_j$;
- for each timeout event $t_j = t_0[x_j]$: $t_j - t_1 = \rho_l(x_j)$;
- for each clock $x_l$ that is started but does not timeout: $t_j - t_1 < \rho_l(x_j)$, where $j$ is the largest index such that $x_l$ is in $X_j$.

By standard DBM techniques, we can check whether $\text{constr}(\beta)$ is satisfiable by saturating it (i.e., closing it under implied constraints, which will always be of form $n < t_j - t_l < m$ or $t_j - t_l = m$ for integers $n, m$), and checking that all such differences allow $t_j - t_l$ to be positive for each $j$. We infer that $\beta$ is feasible iff $\text{constr}(\beta)$ is satisfiable. Moreover, if $\beta$ is feasible, we can use the techniques of Section 3.3 to construct a solution such that $\text{frac}(t_j) \neq 0$ for each pair of distinct indices $j$ and $l$ with $t_j, t_l \in I$.

We can derive the following lemmas.

**Lemma 4.1.** Suppose $\beta, \beta'$ are untimed behaviors such that $\text{Zone}(\beta) = \text{Zone}(\beta')$. Let $y$ be any untimed behavior: Then $\beta \approx y$ is feasible iff $\beta' \approx y$ is feasible.

**Lemma 4.2.** (Zone(\beta) \mid \beta$ feasible untimed behavior of $\mathcal{M}$) is finite.

Let $\beta$ be a feasible untimed behavior and let $x \in X$ be a timer. Then we say that $x$ is expirable after $\beta$ if there exists a valuation in $\text{Zone}(\beta)$ in which $x$ is minimal.

**Lemma 4.3.** Suppose $\beta$ is a feasible untimed behavior with Last(\beta) = $Y$ and $Y \xrightarrow{t_0[x]/0} Y'$ is an untimed behavior. Then $x$ is expirable after $\beta$ iff $Y \xrightarrow{t_0[x]/0} Y'$ is feasible.

### 4.3 Building an untimed from a timed teacher

We will now show how to construct an adapter that transforms a teacher for the timed setting into a teacher for the untimed setting. The adapter maintains an integer variable $d_{\text{max}}$ to store an estimate of the maximal timeout value that occurs in $\mathcal{M}$. Initially, the adapter sets $d_{\text{max}}$ to an arbitrary value, which may be increased based on the (transparent) timed words that it receives from the timed teacher.

Suppose the untimed teacher receives a membership query, consisting of an untimed input word $u = i_1 \cdots i_k$. We present an algorithm that constructs a response for $u$, that is, a feasible partial untimed behavior $\beta$ of $\mathcal{M}$ in canonical form with $uiw(\beta)$ a maximal
prefix of \( u \). The algorithm maintains a variable \( B \) that stores the fragment of \( \beta \) computed thus far, and a counter \( j \) that ranges from 0 to \( k \). Initially \( B \) is set to the trivial untimed behavior \( \emptyset \) and \( j \) to 1. The algorithm then enters its main loop in which the following case statement is performed while \( j \leq k \). We will maintain as a loop invariant that \( B \) is a feasible untimed behavior with \( j - 1 \) inputs.

1. Case \( ij \in I \). Let \( N := B \cdot i/o/\rho_0 \emptyset \), where \( \omega \in O \) is an arbitrary output that acts as placeholder and \( \rho_0 \) is the empty update. (We omit the arrow for \( i/j/o/\rho_0 \) here.) Since \( B \) is feasible (by the loop invariant), it follows by Lemma 3.8 that extension \( N \) is also feasible. Thus the constraints in constraints\((N)\) are satisfiable and we may compute a solution \( t_1, \ldots, t_j \). Let \( t_0 = 0, d_j = t_j - t_{j-1} \), for \( j = 1, \ldots, j \), and let \( o_1, \ldots, o_{j-1} \) be the output events occurring in \( B \). Then \( w = d_j i_1 o_1 d_2 i_2 \cdots d_j i_j \rho_0 \) is a transparent timed word. Let \( u' = \text{tw}(w) \) be the corresponding transparent timed input word. Forward membership query \( u' \) to the timed teacher, and let \( w' \) be the response. If \( w' \) and \( w \) are equal, except possibly for the last output symbol, which is \( o_j \) in \( w' \), then we set \( B := B \cdot i/o/\rho_0 \emptyset, \) increment counter \( j \), and finish the body of the loop. Otherwise, \( w' \) contains some timeout event that was not supposed to happen according to untimed input word \( u \). Since \( u' \) is transparent, \( w' \) is transparent as well. Thus we may extract from \( w' \) which event \( l \) started the timer, to which value \( e \) the timer was set, and the index \( m \) of the timeout event. We set \( N := \text{addtimer}(l, e, m - 1, B) \) and finish the body of the loop.

2. Case \( ij = \tau[x_j], \) for \( l < j \). If timer \( x_j \) is started in \( B \) and initialized with the value \( e \) then let \( N := \text{addtimer}(l, e, j - 1, B) \cdot \tau[x_j]/\omega/o/\rho_0 \emptyset \). Check whether constraints\((N)\) is satisfiable. If so proceed to (*), otherwise exit the while loop. If timer \( x_j \) is not started in \( B \), let \( N^e = \text{addtimer}(l, e, j - 1, B) \cdot \tau[x_j]/\omega/o/\rho_0 \emptyset \), for \( e = 1, \ldots, d_{\text{max}} \). If there exists an \( e \) for which constraints\((N^e)\) is satisfiable, set \( N := N^e \) and proceed to (*), otherwise exit the while loop.

(*) Compute a solution \( t_1, \ldots, t_j \) for constraints\((N)\). Let \( t_0 = 0, d_j = t_j - t_{j-1}, \) for \( j = 1, \ldots, j \), and let \( o_1, \ldots, o_{j-1} \) be the outputs occurring in \( B \). Then \( w = d_j i_1 o_1 d_2 i_2 \cdots d_j i_j \rho_0 \) is a transparent timed word. Let \( u' = \text{tw}(w) \) be the corresponding transparent timed input word. Forward membership query \( u' \) to the timed teacher, and let \( w' \) be the response. Since \( u' \) is transparent, \( w' \) is transparent as well. If \( w' \) and \( w \) are equal, except possibly for the last output symbol, which is \( o_j \) in \( w' \), set \( B \) equal to \( N \) with the last output changed to \( o_j \), increment counter \( j \), and finish the body of the while loop. It may also occur that, even though the last event of \( w' \) is a timeout triggered by event \( i_j \), the value \( e' \) to which this timer is set is different from \( e \). In this case set \( B := \text{addtimer}(l, e', j - 1, B) \cdot \tau[x_j]/\omega/o/\rho_0 \emptyset, \) where \( o_j \) is the last output in \( w' \), increment counter \( j \) and finish the body of the while loop. Finally, it may occur that \( w' \) contains some timeout event that was not supposed to happen according to untimed input word \( u \). In this case, we extract from \( w' \) which event \( l \) started the timer, to which value \( d \) the timer was set, and the index \( m \) of the timeout event. Set \( B := \text{addtimer}(l, d, m - 1, B) \) and finish the body of the while loop.

After termination of the while loop, the algorithm returns the feasible untimed behavior \( B \). Note that the number of iterations of the main loop is linear in the size of \( u \).

Implementing equivalence queries is easy. Suppose that the untimed teacher receives an equivalence query \( \mathcal{H} \). Then we just forward this query to the timed teacher. If the timed teacher answers yes then \( \mathcal{H} \approx_{\text{timed}} M \). In this case, by Theorem 3.13, \( \mathcal{H} \approx_{\text{untimed}} M \), and thus the untimed teacher should also return the result yes. If the timed teacher answers no and returns a counterexample \( w \), then \( w \) is a transparent timed word of \( M \) but not of \( \mathcal{H} \). In this case, by Theorem 3.4, we may conclude that \( \mathcal{H} \neq_{\text{untimed}} M \), and thus the untimed teacher should also return a result no. Since \( w \) is a transparent timed word of \( M \), it follows by Lemma 3.11 that \( w \) has a unique causality map \( c \). This allows us to transform \( w \) into a feasible untimed behavior \( \beta \) in canonical form that is a partial untimed behavior of \( M \) but not of \( \mathcal{H} \), the causality map tells us exactly which timers are set and timeout. Thus the untimed teacher may return \( \beta \) as counterexample. If our estimate \( d_{\text{max}} \) of the maximal timeout value of \( M \) is too low, then it may occur that in \( w \) the timeout value of some timer is larger than \( d_{\text{max}} \). In this case, the value of \( d_{\text{max}} \) is updated to the newly observed maximal timeout value.

5 A Myhill Nerode Theorem for MMTs

We will now exploit the equivalence between timed and untimed semantics (Theorems 3.4 and 3.13). In this section, we develop a Nerode congruence for MMTs from their sets of untimed behaviors. In Section 6 present an approximation of this Nerode equivalence, to be used as a basis for the learning algorithm in Section 7. A Nerode congruence allows to build automata from their languages, which for MMTs are their untimed behaviors, in canonical form.

Definition 5.1. A timer language over \( I \) and \( O \) is a nonempty set \( S \) of feasible untimed behaviors in canonical form over \( I \) and \( O \) that satisfies the following five properties:

- no initial timers: \( \beta \in S \Rightarrow \text{Head}(\beta) = \emptyset, \)
- prefix closed: \( \beta \cdot y \in S \Rightarrow \beta \in S, \)
- behavior deterministic: \( \beta \overset{1/\omega}{\longrightarrow} X_1 \in S \land \beta \overset{1/\omega}{\longrightarrow} X_2 \in S \Rightarrow \omega = \emptyset \land \beta \overset{0/\emptyset}{\longrightarrow} X_1 = X_2, \)
- input complete: \( \beta \in S \land i \in I \Rightarrow \exists \rho, Y : \beta \overset{i/\rho}{\longrightarrow} Y \in S, \)
- timeout complete: \( \beta \in S \land x \text{ expirable after } \beta \Rightarrow \exists \rho, Y : \beta \overset{\text{to}(x)/\rho}{\longrightarrow} Y \in S. \)

During learning of an MMT, the Learner does not “see” all the timers in the sets of timers of an untimed behavior, only those that have been observed to expire at some later point. Recall that an untimed behavior is lean if it is canonical and includes only timers that expire during the behavior. Let \( \text{lean}(\beta) \) be the lean behavior obtained from a canonical behavior \( \beta \). Since the sequence of timer sets in a lean behavior is uniquely determined by the labels on its transition, we can denote a lean behavior simply by the sequence \( i_1/\omega_1/\rho_1 \cdots i_n/\omega_n/\rho_n \) of its labels. If some \( \rho_i \) is empty, we often omit it. Not that the last assignment of any lean behavior is empty.

Define a lean timer language to be the set of lean behaviors derived from a timer language. Given a lean timer language \( S \), we can obtain a corresponding timer language as follows. For a lean behavior, let \( \text{memy}(\beta) \) be the set of timers \( x_j \in x_1, \ldots, x_j \beta \) whose corresponding timeout event (of form \( \text{to}(x_j) \)) occurs (as an input)
in som continuation $β′$ of $β$ in $S$. Let $val_{S, β}$ map each timer $x_i$ in $mem_{S_β}(β)$ to the unique positive integer to which it is assigned in that extension. It is easy to transform a lean untimed language into a corresponding untimed one: simply replace each lean behavior by the untimed behavior obtained by letting $mem_{S_β}(β)$ be the set of timers after $β$, and assigning each occurring timer to $val_{S, β}(x_i)$ in the $i$th transition. In the following, when referring to “timer languages”, we will always mean “lean timer languages”.

The basis for a Nerode equivalence is to define residual languages. Intuitively, we would like to say that $β$ and $β′$ are equivalent if roughly $β · γ ∈ S ↔ β′ · γ ∈ S$. However, we must be careful with the names of timers that expire and/or are assigned in $γ$. We will therefore introduce conventions for naming timers in suffixes.

So, extend the set of timers by the set $Y = \{y_1, y_2, \ldots\}$ of suffix timers, which is disjoint from $X$. Let $TO(Y)$ be the set of timeout events of form $to[y_i]$ for $y_i ∈ Y$, and let $I = TO[X] ∪ TO(Y)$. Define a lean suffix behavior (lean suffix for short) to be a sequence $i_j / α_j / \cdots / i_m / α_m / \rho_m$ of input/output/assignment triples, in which each $i_j$ is in $I$, each $\rho_j$ may assign only to the timer $y_j$, each timeout event occurs at most once, and all timers that are assigned in $Y$ expire in some transition after their assignment.

For integer $k ≥ 0$, let $g_{−k}$ be the injective mapping on $Y$ which maps each $y_j$ to $x_{j+k}$. We apply mappings of form $g_{−k}$ to lean suffix behaviors in the natural way.

We can now define residual languages. For a (lean) timer language $S$ and lean behavior $β ∈ S$, let $β^{-1}S$ be the set of lean suffixes $γ$ such that there is a canonical behavior $β′$ with $β ⊆ β′$ such that $β′ · g_{−1}β(y) ∈ S$. Then $mem_{S_β}(β)$ is the set of timers $x_i$ in $x_1, \ldots, x_β | \beta$ whose corresponding timeout event (of form $to[x_i]$) occurs (as an input) in some suffix $γ$ in $β^{-1}S$, and $val_{S, β}$ maps each timer $x_i$ in $mem_{S_β}(β)$ to the unique positive integer to which it is assigned in the corresponding lean behavior $β′ · g_{−1}β(y) ∈ S$.

We can then define the Nerode equivalence.

**Definition 5.2.** Let $S$ be a lean timer language with $β, β′ ∈ S$. Let $f : mem_{S_β}(β) → mem_{S_β}(β′)$ be a bijection from $mem_{S_β}(β)$ to $mem_{S_β}(β′)$. Then $β$ and $β′$ are equivalent under $f$, written $β \equiv_{S, V} β′$ iff

\[ γ ∈ β^{-1}S \iff f(γ) ∈ β′^{-1}S \]

Intuitively, $β \equiv_{S, V} β′$ means that $β$ and $β′$ allow the same suffixes with inputs in $V$, after renaming timers assigned in $β$ by $f$. We write $β \equiv_{S, V} β′$ to denote that $β \equiv_{S, V} β′$ for some $f : mem_{S_β}(β) → mem_{S_β}(β′)$. Then $β$ and $β′$ are equivalent with respect to $V$.

**Example**

Let $S$ contain the lean behaviors

\[ i_1 / o_1 / \{x_1 := 5\} · to[x_1] / o_3 \text{ and } i_2 / o_2 / \{x_2 := 4\} · to[x_2] / o_3 \]

Let $β_1 = i_1 / o_1$ and $β_2 = i_2 / o_2$. Then $β_1^{-1}S$ contains the suffix $γ_1 = to[x_1] / o_3$ and $β_2^{-1}S$ contains the suffix $γ_2 = to[x_2] / o_3$. It is now possible that $β_1 \equiv_{S, V} β_2$ where $f$ maps $x_1$ to $x_2$ (whether $β_1 \equiv_{S, V} β_2$ actually holds depends also on the other behaviors in $S$).

**Theorem 5.3.** Let $S$ be a lean timer language. Then there exists an $MMT$ with lean timer language $S$ iff $∈S$ has finitely many equivalence classes (finite index).

### 6 Approximating the Nerode Equivalence

For the learning algorithm, we must define an overapproximation of the Nerode equivalence on untimed behaviors defined in Definition 5.2. This approximated equivalence can be inferred using a finite set of membership queries, and therefore be used as a basis for a learning algorithm, analogously to the use of an approximated Nerode equivalence in $L^*$ [1].

It seems natural to parameterize such an equivalence by a finite set $V$ of untimed input words (hereafter often called input suffixes), restricting Definition 5.2 to suffixes $γ$ and $f(γ)$ with $uiw(γ)$ and $uiw(f(γ))$ in $V$. Already here, we see that it is convenient to let $V$ be closed under permutations on timers in $X$, so that $uiw(γ) ∈ V$ iff $uiw(f(γ)) ∈ V$. Let us call such a set an adequate set. For an adequate set $V$ of input suffixes, and a lean behavior $β$, let $(β^{-1}S)_V$ be the set of suffixes $γ$ with $uiw(γ) ∈ V$. Let $mem_{S_β}(γ)$ be the set of timers $x_i$ in $x_1, \ldots, x_β | \beta$ whose corresponding timeout event (of form $to[x_i]$) occurs (as an input) in some suffix in $(β^{-1}S)_V$. Let $val_{S, V, β}$ map each timer $x_i$ in $mem_{S_β}(γ)$ to the unique positive integer to which it is assigned in the $i$th transition of $β$.

We can now define the approximated Nerode equivalence, which is parameterized on an adequate set of lean input suffixes.

**Definition 6.1.** Let $S$ be a timer language, let $β$ and $β′$ be canonical untimed behaviors in $S$, and let $V$ be an adequate set of lean input suffixes. Let $f : mem_{S_β}(γ) → mem_{S_β}(γ′)$ be a bijection from $mem_{S_β}(γ)$ to $mem_{S_β}(γ′)$. Then $β$ and $β′$ are equivalent with respect to $V$, written $β \equiv_{S, V, β′}$ iff

\[ γ ∈ (β^{-1}S)_V \iff f(γ) ∈ (β′^{-1}S)_V \]

Intuitively, $β \equiv_{S, V, β′}$ means that $β$ and $β′$ allow the same suffixes with inputs in $V$, after renaming timers assigned in $β$ by $f$. We write $β \equiv_{S, V, β′}$ to denote that $β \equiv_{S, V, β′}$ for some $f : mem_{S_β}(γ) → mem_{S_β}(γ′)$. The following standard theorem follows rather directly from the definitions.

**Theorem 6.2.** Let $S$ and $V$ be as above. $∈S$ is included in $∈S, V$. Moreover, if $∈S$ has finite index, then it is equal to $∈S, V$ for some finite set $V$.

### 7 Algorithm for Learning of MMTs

In this section, we present an algorithm for learning MMTs in then untimed MAT of 4.1, using the approximated Nerode equivalence presented in Section 6. The learning algorithm follows the standard pattern for active automata learning algorithms, such as $L^*$ [1]. It maintains a set $U$ of lean behaviors, called short prefixes, which represent states in the MMT to be constructed, and an overapproximation of the Nerode equivalence, parameterized by a set $V$ of input suffixes. The learning algorithm iterates two phases: hypothesis construction and hypothesis validation. During hypothesis construction, the approximation of the Nerode equivalence triggers the expansion of $U$ and $V$ until two convergence conditions are satisfied that allow a hypothesis automaton to be formed. During hypothesis validation, the hypothesis automaton is submitted in an equivalence query, and returned counterexamples are used to refine the Nerode equivalence by expanding $U$. Let us introduce the two conditions for convergence of the construction phase. For a lean behavior $β ∈ S$ and $γ ∈ β^{-1}S$, let $β_δ γ$ be the (unique) lean behavior $β′ · g_{−1}β(y)$ in $S$ with $β ⊆ β′$. Let $\mathcal{E}(β)$ be the set of $i ∈ I$ such that $β_δ / o / S$ and some $o$ (recall that the last assignment of a lean behavior is always empty). For $i ∈ \mathcal{E}(β)$, let $λ(β, i)$ be the unique output $o$ such that $β_δ / o / S$. Let $U$ be a prefix-closed set of lean behaviors, and let $V$ be an adequate set of input suffixes.
• $U$ is closed wrt $V$ if for each $\beta \in U$ and $i \in \mathbb{E}_S(\beta)$ there is a $\beta' \in U$ such that $\beta_S i / \lambda(\beta, i) \subseteq \mathbb{E}_V \beta'$.

• $U$ is timer-consistent wrt $V$ if for each $\beta \in U$ and $i \in \mathbb{E}_S(\beta)$ we have $\mathbb{E}_S \mathbb{V}(\beta_S i / \lambda(\beta, i)) \subseteq (\mathbb{E}_S \mathbb{V} \beta) \cup \{x_{|\beta|+1}\})$. Closedness ensures that each transition in the MMT to be constructed has a target state. Timer-consistency states that each timer which is needed after such a transition (i.e., a timer set during $\beta_S i / \lambda(\beta, i)$) is either a timer active after $\beta$ or is started by the last transition, thus getting the name $x_{|\beta|+1})$. Closedness and timer-consistency allow the construction of a hypothesis MMT.

Definition 7.1 (Hypothesis automaton). Let $U$ be a non-empty prefix-closed set of lean behaviors, and $V$ an adequate set of input suffixes such that $U$ is closed and timer consistent wrt $V$. Then the hypothesis automaton $\mathcal{H}(U, V)$ is the MMT $\mathcal{H}(U, V) = (I, O, Q, q_0, X, \delta, \lambda, \pi)$, where

- $Q = U$ and $q_0 = \epsilon$.
- $X$ maps each location $\beta \in U$ to $\mathbb{E}_S \mathbb{V}(\beta)$.
- Let $\beta \in U$ and $i \in \mathbb{E}_S(\beta)$. Then
  - $\delta(\beta, i)$ is the unique $o$ such that $\beta_S i / o \in S$.
  - $\mathbb{E}_S(\beta)$ is the unique $\beta' \in U$ such that there is an $f$ with $\beta_S i / \lambda(\beta, i) \subseteq \mathbb{E}_V \beta'$.
  - $\pi(\beta, i) : \mathbb{E}_S \mathbb{V}(\beta') \mapsto (\mathbb{E}_S \mathbb{V}(\beta) \cup \mathbb{E}_S \mathbb{V}(\beta'))$ is defined as $f^{-1}$ on $\mathbb{E}_S \mathbb{V}(\beta')$, except that it maps $f(x_{|\beta|+1})$ to $\mathbb{E}_S \mathbb{V}(\beta')$.

When $\beta = \hat{\beta}$ and $i$ is of form $t_0[x_j]$ with $x_j \in \mathbb{E}_S \mathbb{V}(\beta)$ but $t_0[x_j] \notin \mathbb{E}_S(\beta)$, we let $\delta(\beta, i) = \hat{\beta}$, and let $\pi(\beta, i)$ be the identity mapping on $\mathbb{E}_S \mathbb{V}(\beta)$.

The last case (where $t_0[x_j] \notin \mathbb{E}_S(\beta)$) constructs a transition that is not feasible, but which might anyway be syntactically present, since $x_j$ may expire after some continuation if $\beta$ and hence live.

Hypothesis construction performs membership queries in order to construct the sets of form $(\beta逆转1S)V$ and $(\beta逆转1S)\lambda(\beta, i)逆转1S)V$ for $\beta \in U$ and $i \in \mathbb{E}_S(\beta)$. Moreover, the sets $U'$ and $V'$ are expanded if needed to construct a hypothesis automaton.

More precisely, membership queries are first performed for all untimed input words of form $\text{input}(\beta \cdot i)$ for $\beta \in U$ and $i \in I$. This allows to determine $\mathbb{E}_S(\beta)$ and $\lambda(\beta, i)$ for $\beta \in U$ and $i \in \mathbb{E}_S(\beta)$. Thereafter, membership queries are performed for all untimed input words of form $\text{input}(\beta \cdot i) \cdot \text{output}(\beta \cdot i)$, where $\beta \in U$, $\alpha \in V$, and $i \in \mathbb{E}_S(\beta)$. Note that one need only consider $\alpha$ in which timeouts from $TQ[X]$ concern timers in $\{x_1, \ldots, x_{|\beta|}\}$ or in $\{x_1, \ldots, x_{|\beta|+1}\}$ for $\text{input}(\beta \cdot i)$. This allows to construct the sets of form $(\beta逆转1S)V$ and $(\beta逆转1S)\lambda(\beta, i)逆转1S)V$ for $\beta \in U$. It then allows to compute the approximated Nerode equivalence $\equiv_S \lambda \cdot V$ on the set of lean behaviors of form $\beta$ and $\beta_S i / \lambda(\beta, i)$ with $\beta \in U$ and $i \in \mathbb{E}_S(\beta)$. We then check whether $U$ and $V$ meet the convergence criteria.

• Whenever the set $U$ is not closed wrt $V$, then it is extended: if there is some $\beta \in U$ and $i \in \mathbb{E}_S(\beta)$ for which there is no $\beta' \in U$ such that $\beta_S i / \lambda(\beta, i) \equiv \mathbb{E}_V \beta'$, then $\beta_S i / \lambda(\beta, i)$ is added to $U$, triggering new membership queries.

• Whenever the set $U$ is not timer-consistent wrt $V$, then $V$ is extended: for timer $x_j$ in $(\mathbb{E}_S \mathbb{V}(\beta_S i / \lambda(\beta, i)) \cup \mathbb{E}_S \mathbb{V}(\beta) \cup \{x_{|\beta|+1}\})$, find a lean suffix $y$ in $(\mathbb{E}_S \mathbb{V}(\beta逆转1S)\lambda逆转1S)V$, whose last input is $t_0[x_j]$. This input is obviously missing from $(\beta逆转1S)V$. But, by adding the concatenation of $i$ with $\text{output}(\beta \cdot j)$ (where timers are appropriately renamed), the input $t_0[x_j]$ will also appear in $(\mathbb{E}_S \mathbb{V}(\beta逆转1S)\lambda逆转1S)V$. This extension of $V$ will subsequently trigger new membership queries.

When $U$ is closed and timer-consistent wrt $V$, then a hypothesis $\mathcal{H}(U, V)$ is constructed and validated by submitting it in an equivalence query. If the query returns “yes”, then the learning is completed, and $\mathcal{H}(U, V)$ accepts $S$. If the query returns a counterexample in the form of a behavior $\alpha$ on which $\mathcal{H}(U, V)$ and $S$ disagree, a procedure for counterexample processing, found in the appendix, is used to extend $V$ by a new input suffix $\alpha$ such that $U$ is no longer closed wrt $V$, as follows. We assume w.l.o.g. that no proper prefix of $\alpha$ is a counterexample. By the fact that $\alpha$ is a counterexample, we can find a lean suffix $y$ of $\alpha$, such that $\beta逆转1S i / o \equiv \mathbb{E}_V \beta'$ but $y \notin (\beta逆转1S i / \lambda(\beta, i)逆转1S)\lambda逆转1S V \lor y \notin \beta逆转1S V$ for some $\beta, \beta'$ in $U$ such that $\beta逆转1S i / o \equiv \mathbb{E}_V \beta'$ is used to construct the transition triggered by $i$ from $\beta$ in $\mathcal{H}(U, V)$. To see this, let $\alpha = i_1 / o_1 / \ldots / i_n / o_n / p_n$, and for $j = 0, \ldots, n$, define a lean behavior $\beta_j$ and a $\beta_j$-suffix as follows.

- $\beta_0 = \epsilon$, and $y_0$ is obtained by making $\alpha$ a lean suffix (i.e., renaming each timer $x_j$ to $y_j$).
- Let $y_j$ be of form $i_j / o_j / y_{j+1}$, let $\beta_j = \lambda(\beta_{j-1}, i_j)$, let $f$ be the mapping used to establish $\beta_{j-1}逆转1S i_j / o_j \equiv \mathbb{E}_V \beta_j$ in the construction of $\mathcal{H}(U, V)$, and let $y_j$ be obtained from $y_{j-1}$ by (i) applying $f$ to timers in $\{x_1, \ldots, x_{|\beta_j|}\}$, (ii) replacing $y_{j-1}$ by $f_j(x_{|\beta_j|-1})$, and (iii) replacing each $y_j$ by $y_{j+1}$.

The result is that $\beta_0 \ldots \beta_n$ is the sequence of states visited when $\mathcal{H}(U, V)$ processes $\alpha$, and $y_j$ is lean suffix that can be composed with $\beta_j$, which corresponds to a suffix of $\alpha$. By the fact that $\alpha$ is a counterexample, we have $y_j \in \beta逆转1S逆转1S V \lor y_n \in \beta逆转1S逆转1S V$ (since $y_n$ is the empty sequence), which implies that $y_{j-1} \in \beta逆转1S逆转1S V \lor y_j \in \beta逆转1S逆转1S V$ for some $j$; we can then take $\beta_{j-1}$ as $\beta$ and $\beta_j$ as $\beta'$, and $y_j$ as $y$. This means that $y$ is a new separating suffix, and that $V$ should be extended with $\text{input}(\gamma)$. After adding $\gamma$ and (its permutations) to $V$, $U$ is no longer closed wrt $V$. The algorithm can then resume a next round of hypothesis construction, which will eventually generate a new hypothesis, etc.

The algorithm enjoys properties analogous to those of, e.g., $L^*$ [1]. The additional complexity caused by timers is analogous to that caused by registers in learning of register automata [6, 16].

Theorem 7.2. Given an MMT $M$ whose canonical form has $n$ states, each of which has at most $r$ active timers, the procedure of this section terminates and produces an equivalent MMT in canonical form, using a number of queries that is at most polynomial in $n$ and doubly exponential in $r$.

At termination the hypothesis is correct, by definition of equivalence query. During the construction, $U$ and $V$ will expand until they represent $\equiv_S$, at which time the hypothesis will be the desired one. Let us analyze the number of queries that may be required in the worst case to learn an MMT whose canonical form has $n$ states and at most $r$ active timers in any state. In the below, let $|V|$ be the number of unique elements of $V$, before adding permutations.

• Hypothesis construction may need $n \cdot (|I| + r + 1) \cdot |V| \cdot r!$ membership queries in total. The factor $r!$ arises as the number of permutations of prefix-timers in each suffix.

• Processing a counterexample may need $\log(m)$ membership queries, using binary search, where $m$ is the length of the counterexample.
- Each equivalence query will result in refuting an equivalence of form $\beta \cdot i/o \not\equiv \gamma$, and extending $V$. There are at most $r!$ possible permutations $f$ for each $\beta \cdot i/o$, implying at most $n \cdot r!$ equivalence queries.
- Since each equivalence query adds at most one element to $V$, we have $|V| \leq n \cdot r!$ when the algorithm finishes.

8 Conclusions and Future Work

We have presented a new automaton-based model for timed systems, MMTs, which aims to be sufficiently simple to allow tractable learning algorithms, and sufficiently expressive to model common network protocols. For the MMT model we have developed a Nerode congruence, allowing to define canonical forms, and used it as the basis for an active learning algorithm, which generalizes $L^*$. A key technical result is the equivalence between the timed semantics, which is suitable to represent practical learning scenarios, and the untimed semantics, which is suitable for learning algorithms. This equivalence is embodied by an adapter, which transforms queries in one model to queries in the other.

The query complexity of our learning algorithm is polynomial in the number of states of the learned MMT, but doubly exponential in the number of simultaneously active timers. Since practical network protocols have at most a couple of simultaneously active timers, this leads us to believe that our work will be a suitable theoretical basis for practical learning algorithms for timed protocols.

Our work constitutes a major step towards a practical approach for active learning of timed systems. Such an approach would greatly enhance the applicability of active learning for reverse engineering of models of software and hardware systems. Future work includes to implement equivalence queries for MMTs, in the untimed case, equivalence queries for Mealy machines are approximated using conformance testing algorithms, for which a rich theory exists [20]. Our equivalence result between timed and untimed semantics may help to lift such algorithms to the setting of MMTs. A challenge is also to deal with timing uncertainties due to nondeterminism and imprecise measurements. In a realistic setting we may need more than one experiment to figure out which event causes a timeout. We may observe, for instance, that slight changes in the timing of certain inputs lead to corresponding changes in the timing of certain timeouts.

References