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:
Bengt Jonsson and Frits Vaandrager. 2018. Learning Mealy Machines with Timers. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

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 $stop(x)$ denotes that timer x is stopped. For readability we have omitted trivial self-loops. Λ 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

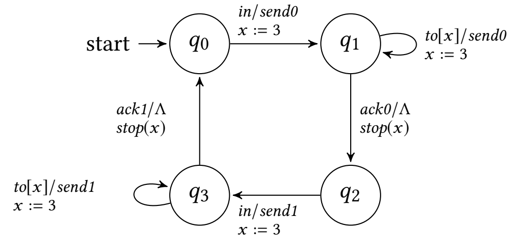

Figure 1. MMT model for alternating-bit protocol sender

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

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, parameterized by sets of suffixes, which is the basis for constructing hypothesis automata in the learning algorithm. The approximated Nerode equivalence allows us finally to present an active automata learning algorithm for MMTs. 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.

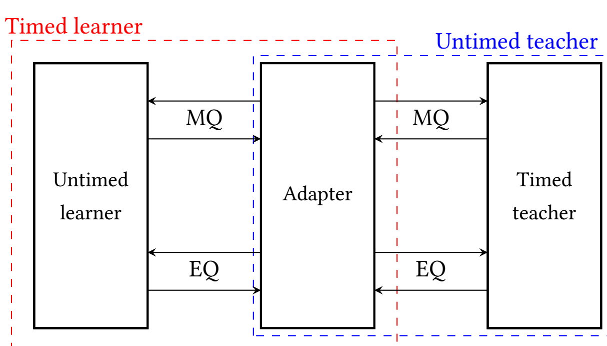

Figure 2. Learning architecture

Summarizing, our paper includes the following contributions:

- a new model of timed systems, MMTs, which is a small extensions of Mealy machines, but still expressive enough for many network protocols,
- an untimed semantics equivalent to the timed semantics for MMTs, which allows to define a Nerode equivalence as a basis for learning algorithms
- a tractable algorithm for learning MMTs.

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 presents 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 $X = \{x_1, x_2, x_3, \ldots\}$ of *timers*. Let $TO[X]$ be the set of *timeout events* of the form $to[x]$ for $x \in X$. For a set I, let $\hat{I}$ be $I \cup TO[X]$. We view (partial) functions as sets of pairs. We write $A \hookrightarrow B$ for the set of partial functions from A to B, and $f\lceil A$ for the restriction of function f to $\mathrm{dom}(f) \cap A$. We write $\mathcal{P}_{fin}(A)$ for the set of finite subsets of set 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 $I \cap TO[X] = \emptyset$,
- Q is a finite set of states, with $q_0 \in Q$ the initial state,
- $\mathcal{X} : Q \to \mathcal{P}_{fin}(X)$, with $\mathcal{X}(q_0) = \emptyset$,
- $\delta : Q \times \hat{I} \hookrightarrow Q$ is a transition function,
- $\lambda : Q \times \hat{I} \hookrightarrow O$ is an output function,
- $\pi : Q \times \hat{I} \hookrightarrow (X \hookrightarrow (X \cup \mathbb{N}^{>0}))$ is a timer update function.

Let $q \in Q$, $i \in \hat{I}$, $q' = \delta(q,i)$ and $\rho = \pi(q,i)$. We require that $\mathrm{dom}(\rho) = \mathcal{X}(q')$ and $\mathrm{ran}(\rho) \subseteq \mathcal{X}(q) \cup \mathbb{N}^{>0}$. When a timer expires it is stopped: if $i = to[x]$, for some x, then $x \notin \mathrm{ran}(\rho)$. 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 $i = to[x]$, for some $x \in \mathcal{X}(q)$. We write $q \xrightarrow{i/o/\rho} q'$ if $\delta(q,i) = q'$, $\lambda(q,i) = o$ and $\pi(q,i) = \rho$.

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

Semantics. We define the semantics of an MMT $\mathcal{M} = (I, O, Q, q_0, \mathcal{X}, \delta, \lambda, \pi)$ via an infinite state transition system that describes all possible configurations and transitions between them. A *valuation* is a partial function $\kappa : X \hookrightarrow \mathbb{R}^{\geq 0}$, defined on a finite subset of X, that assigns nonnegative real numbers as values to timers. We write $\mathit{Val}(Y)$ for the set of valuations with domain $Y \subseteq X$. A *configuration* of an MMT is a pair (q, κ), where $q \in Q$ is a state and $\kappa \in \mathit{Val}(\mathcal{X}(q))$ is a valuation. The *initial configuration* is the pair (q_0, κ_0), where κ_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 κ is a valuation in which 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 κ, κ' valuations and $d \in \mathbb{R}^{>0}$, we define a delay transition relation by: $\kappa \xrightarrow{d} \kappa'$ iff

$$\mathrm{dom}(\kappa) = \mathrm{dom}(\kappa') \wedge \forall x \in \mathrm{dom}(\kappa) : \kappa'(x) = \kappa(x) - d.$$

If the current valuation is κ, then timeout event $to[x]$ may occur only if $\kappa(x) = 0$. If an input or a timeout event occurs, κ is updated as specified by update function ρ. Let ι denote the embedding of $\mathbb{N}^{>0}$ in $\mathbb{R}^{\geq 0}$. Then, for κ, κ' valuations, $i \in \hat{I}$, $o \in O$ and $\rho \in X \hookrightarrow (X \cup \mathbb{N}^{>0})$, we define a discrete transition relation by: $\kappa \xrightarrow{i/o/\rho} \kappa'$ iff

$$\begin{aligned}
&\mathrm{dom}(\rho) = \mathrm{dom}(\kappa') \ \wedge \mathrm{ran}(\rho) \subseteq \mathrm{dom}(\kappa) \cup \mathbb{N}^{>0} \ \wedge \\
&\kappa' = (\kappa \cup \iota) \circ \rho \ \wedge \\
&\forall x \in X : i = to[x] \Rightarrow (\kappa(x) = 0 \wedge x \notin \mathrm{ran}(\rho)).
\end{aligned}$$

Transition relations $\xrightarrow{d}$ and $\xrightarrow{i/o/\rho}$ can be lifted to configurations. For all configurations (q, κ), (q', κ') of MMT $\mathcal{M}$,

$$\frac{q = q' \quad \kappa \xrightarrow{d} \kappa'}{(q,\kappa) \xrightarrow{d} (q',\kappa')} \qquad \frac{q \xrightarrow{i/o/\rho} q' \quad \kappa \xrightarrow{i/o/\rho} \kappa'}{(q,\kappa) \xrightarrow{i/o} (q',\kappa')}$$

A *timed run* of $\mathcal{M}$ over w is a sequence

$$\alpha \quad = \quad C_0 \xrightarrow{d_1} C_0' \xrightarrow{i_1/o_1} C_1 \xrightarrow{d_2} \cdots \xrightarrow{d_k} C_{k-1}' \xrightarrow{i_k/o_k} C_k$$

of transitions between configurations C_j, C_j' of $\mathcal{M}$, where C_0 is the initial configuration. A *timed word* over inputs I and outputs O is a sequence

$$w \quad = \quad d_1\ i_1\ o_1\ d_2\ i_2\ o_2 \cdots d_k\ i_k\ o_k,$$

where $d_j \in \mathbb{R}^{>0}$, $i_j \in I \cup \{to\}$, and $o_j \in O$. To each timed run α we associate a *timed word* by forgetting the configurations and the timers in timeout events:

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

where for all j,

$$i_j' \quad = \quad \begin{cases} i_j & \text{if } i_j \in I, \\ to & \text{if } i_j \in TO[X]. \end{cases}$$

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 $\mathcal{M}$ if $\mathcal{M}$ has a timed run α with $w = tw(\alpha)$. Two MMTs $\mathcal{M}$ and $\mathcal{N}$ with the same sets of inputs are *timed equivalent*, denoted $\mathcal{M} \approx_{timed} \mathcal{N}$, iff 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, I/O 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 Λ 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 Λ 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 Λ. We do not allow Λ as output event in timeout transitions $to[x]$: since timeout events themselves are not observable, we can ony 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 timelock following the timed word $1\ i\ o\ 1\ i\ o$: 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

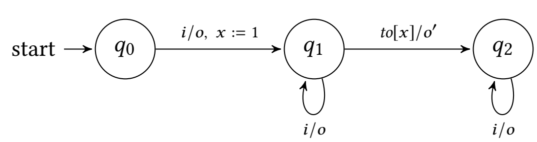

Figure 3. An MMT with a timelock

runs in which no “races” occur. This eliminates the above timelock, 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 $d_j \in \mathbb{R}^{>0}$ 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 $tiw(w)$

to each timed word w by removing the outputs events, removing the occurrences of *to*, replacing consecutive numbers by their sum, and possibly placing 0 at the end of the sequence. Thus, for instance, $tiw(7\ i\ o\ 1\ i\ o\ 1\ to\ o') = 7\ i\ 1\ i\ 1$ and $tiw(3\ i_1\ o_1\ 1.1\ i_2\ o_2\ 2\ to\ o_3\ 2.1\ i_4\ o_4) = 3\ i_1\ 1.1\ i_2\ 4.1\ i_4\ 0$. If u and u' are timed input words, then we write $u \propto u'$ 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 $\mathcal{M}$ such that $tiw(w) \propto u$. We call w an *outcome of running experiment* u on $\mathcal{M}$. For instance, if we perform the experiment 1 *in* 7 on the MMT of Figure 1, then the unique outcome is 1 *in send0* 3 *to send0* 3 *to send0*.

Nondeterminism. 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 nondeterministic behavior. For the MMT of Figure 4 (top), for instance, experiment $1\ i\ 1$ has outcomes $1\ i\ o\ 1\ to\ o'$ and $1\ i\ o\ 1\ to\ o''$.

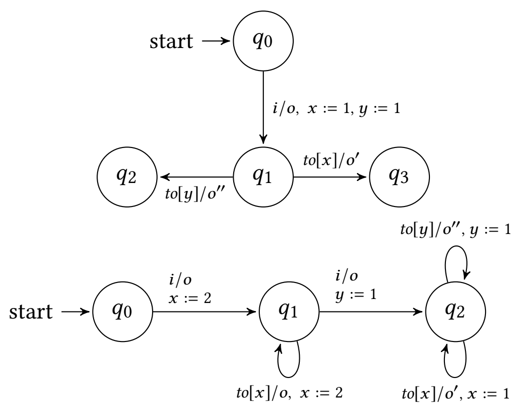

Figure 4. MMTs with "uncontrollable" and "controllable" nondeterminism. For clarity, we omitted some self-loops.

The nondeterminism of the MMT of Figure 4 (top) is "uncontrollable" in the sense that it occurs irrespective of the timing of the inputs. Figure 4 (bottom) gives an example of an MMT that exhibits nondeterminism when the second input occurs *exactly* one time unit after the first input: experiment $1\ i\ 1\ i\ 1$ has outcomes $1\ i\ o\ 1\ i\ o\ 1\ to\ o'$ and $1\ i\ o\ 1\ i\ o\ 1\ to\ o''$. This type of nondeterminism is "controllable" and will not occur if we carefully select the timing of inputs.

Further restrictions on MMTs. Although learning of nondeterministic systems has been studied in the literature [29], nondeterminism 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 $q \xrightarrow{i/o/\rho} q'$ of an MMT (a) ρ is injective, and (b) at most one timer is started by ρ. Condition (b) rules out the nondeterminism 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 $tiw(w)$ is transparent. It is easy to check that for each MMT $\mathcal{M}$ 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 simultanelously.

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 $\mathcal{M}$. Initially, the learner only knows the set I of inputs of $\mathcal{M}$. The learner may perform experiments (membership queries, MQ) to learn about the timed words of $\mathcal{M}$, and pose equivalence queries (EQ) to find out whether a constructed hypothesis is correct.

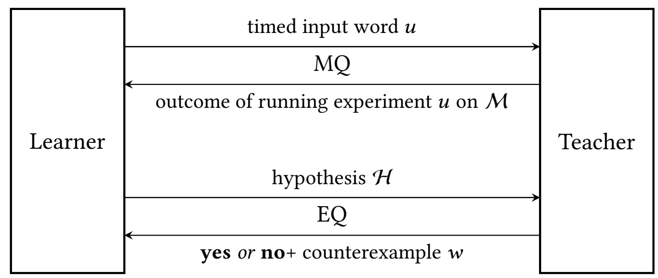

Figure 5. A timed MAT framework

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 $\mathcal{M}$ such that $tiw(w) \propto u$.

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

The main result of this paper is an algorithm that allows the learner to learn an MMT $\mathcal{H}$ that is timed equivalent to $\mathcal{M}$ 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 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 ρ that satisfy, for all x: $\rho(x) \in \mathbb{N}^{>0}$ or $\rho(x) = x$. 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 consists of pairs of states and timer valuations. This means that there are two natural abstractions of timed runs: an abstraction *untime* 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, $beh(untime(\alpha)) = untime(beh(\alpha))$, 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

$$\beta \quad = \quad X_0 \xrightarrow{i_1/o_1/\rho_1} X_1 \cdots \xrightarrow{i_k/o_k/\rho_k} X_k,$$

where $X_0 \subseteq X$ and, for each $j > 0$, $i_j \in \hat{I}$, $o_j \in O$, $\rho_j \in X \hookrightarrow \mathbb{N}^{>0}$, and $X_j \setminus X_{j-1} \subseteq \mathrm{dom}(\rho_j) \subseteq X_j \subseteq X$. Moreover, if $i_j = to[x]$, for some $j > 0$, then $x \in X_{j-1}$ and $x \notin X_j \setminus \mathrm{dom}(\rho_j)$. An *untimed run* of an MMT $\mathcal{M}$ is a sequence

$$\gamma \quad = \quad q_0 \xrightarrow{i_1/o_1/\rho_1} q_1 \cdots \xrightarrow{i_k/o_k/\rho_k} q_k$$

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

$$beh(\gamma) \quad = \quad \mathcal{X}(q_0) \xrightarrow{i_1/o_1/\rho_1} \mathcal{X}(q_1) \cdots \xrightarrow{i_k/o_k/\rho_k} \mathcal{X}(q_k).$$

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

$$\sigma \quad = \quad \kappa_0 \xrightarrow{d_1} \kappa_0' \xrightarrow{i_1/o_1/\rho_1} \kappa_1 \cdots \xrightarrow{d_k} \kappa_{k-1}' \xrightarrow{i_k/o_k/\rho_k} \kappa_k \quad (1)$$

of delay transitions and discrete transitions with, for each j, κ_j, κ_j' valuations and, for each $j > 0$, $d_j \in \mathbb{R}^{>0}$, $i_j \in \hat{I}$, $o_j \in O$, and $\rho_j \in X \hookrightarrow \mathbb{N}^{>0}$. To each timed behavior σ we associate an untimed behavior by forgetting the delay transitions and by replacing valuations by their domain:

$$untime(\sigma) \quad = \quad \mathrm{dom}(\kappa_0) \xrightarrow{i_1/o_1/\rho_1} \cdots \xrightarrow{i_k/o_k/\rho_k} \mathrm{dom}(\kappa_k).$$

We say that untimed behavior β is *feasible* if there exists a timed behavior σ such that $untime(\sigma) = \beta$.

We also associate a timed word to timed behavior σ by forgetting valuations, timers, and update functions:

$$tw(\sigma) \quad = \quad d_1\ i_1'\ o_1\ i_2'\ o_2 \cdots d_k\ i_k'\ o_k,$$

where for all j,

$$i_j' \quad = \quad \begin{cases} i_j & \text{if } i_j \in I, \\ to & \text{if } i_j \in TO[X]. \end{cases}$$

Let α be a timed run of an MMT $\mathcal{M}$:

$$\alpha \quad = \quad (q_0, \kappa_0) \xrightarrow{d_1} (q_0, \kappa_0') \xrightarrow{i_1/o_1} (q_1, \kappa_1) \cdots \xrightarrow{i_k/o_k} (q_k, \kappa_k).$$

Then α can be projected both to an untimed run of $\mathcal{M}$

$$untime(\alpha) \quad = \quad q_0 \xrightarrow{i_1/o_1/\rho_1} q_1 \cdots \xrightarrow{i_k/o_k/\rho_k} q_k$$

(the ρ_j's are determined since $\mathcal{M}$ is deterministic) and to a timed behavior

$$beh(\alpha) \quad = \quad \kappa_0 \xrightarrow{d_1} \kappa_0' \xrightarrow{i_1/o_1/\rho_1} \kappa_1 \cdots \xrightarrow{d_k} \kappa_{k-1}' \xrightarrow{i_k/o_k/\rho_k} \kappa_k.$$

We say that σ is an timed behavior of $\mathcal{M}$ if $\mathcal{M}$ has an timed run α with $beh(\alpha) = \sigma$. Note that $beh(untime(\alpha)) = untime(beh(\alpha))$ and $tw(\alpha) = tw(beh(\alpha))$. Thus the diagram of Figure 6, which summarizes the various types of runs and behaviors that we consider in this article, commutes. Conversely, if γ is an untimed run of $\mathcal{M}$ and σ is a timed behavior such that $beh(\gamma) = untime(\sigma)$, then there exists a unique timed run α of $\mathcal{M}$ with $untime(\alpha) = \gamma$ and $beh(\alpha) = \sigma$. We refer to α as $pullback(\gamma, \sigma)$.

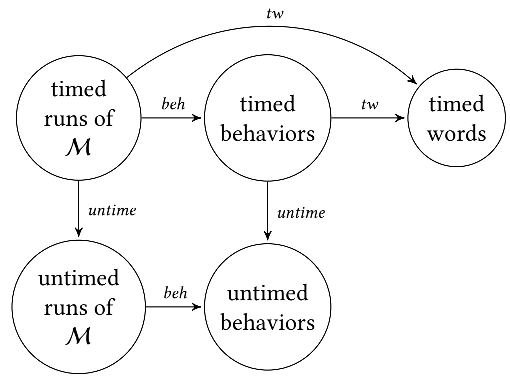

Figure 6. Relating different types of runs and behaviors

For any nonempty sequence σ, $\mathsf{Head}(\sigma)$ denotes the first element, $\mathsf{Tail}(\sigma)$ denotes the sequence obtained by removing the first element, and $\mathsf{Last}(\sigma)$ denotes the last element. Suppose β, β' are untimed behaviors over I and O such that $\mathsf{Last}(\beta) = \mathsf{Head}(\beta')$. Then the *sequential composition* of β and β', written $\beta \cdot \beta'$, is the untimed behavior $\beta\ \mathsf{Tail}(\beta')$. Behavior β is a *prefix* of behavior γ if there exists an behavior β' such that $\gamma = \beta \cdot \beta'$.

3.2 Definition untimed semantics

We would, intuitively, like to define the untimed semantics of an MMT $\mathcal{M}$ as the set of its feasible untimed behaviors. However, this semantics would then depend heavily on the identity of the timers. Therefore, we define an equivalence relation on untimed behaviors, which deems two untimed behaviors equivalent if there is a consistent renaming of timers that transforms the one into the other.

Let β and β' be two untimed behaviors with the same length and outputs:

$$\beta \quad = \quad X_0 \xrightarrow{i_1/o_1/\rho_1} X_1 \xrightarrow{i_2/o_2/\rho_2} X_2 \cdots \xrightarrow{i_k/o_k/\rho_k} X_k,$$

$$\beta' \quad = \quad Y_0 \xrightarrow{i_1'/o_1/\rho_1'} Y_1 \xrightarrow{i_2'/o_2/\rho_2'} Y_2 \cdots \xrightarrow{i_k'/o_k/\rho_k'} Y_k.$$

An *isomorphism* from β to β' is a list $f = f_0, \ldots, f_k$ of bijections $f_j : X_j \to Y_j$ such that for all $j > 0$: (1) for all $x \in X_j \setminus \mathrm{dom}(\rho_j)$), $f_j(x) = f_{j-1}(x)$, (2) $i_j' = i_j$ if $i_j \in I$ and $i_j' = to[f_{j-1}(x)]$ if $i_j = to[x]$, for some $x \in X_{j-1}$, and (3) $\mathrm{dom}(\rho_j') = f_j(\mathrm{dom}(\rho_j))$ and $\rho_j'(y) = \rho_j(f_j^{-1}(y))$, for all $y \in \mathrm{dom}(\rho_j')$. In this case, since β' is fully determined by β and f, we write $\beta' = f(\beta)$. We say that β and β' are *isomorphic* if there exists an isomorphism f from β to β'. Two sets of untimed behaviors A and B are *isomorphic* if for each untimed behavior of A there is an isomorphic untimed behavior in B, and vice versa. Isomorphisms can be lifted to timed behaviors in the obvious way. Let σ and σ' be two timed behaviors with the same length and outputs:

$$\sigma \quad = \quad \kappa_0 \xrightarrow{d_1} \kappa_0' \xrightarrow{i_1/o_1/\rho_1} \kappa_1 \cdots \xrightarrow{d_k} \kappa_{k-1}' \xrightarrow{i_k/o_k/\rho_k} \kappa_k,$$

$$\sigma' \quad = \quad \lambda_0 \xrightarrow{d_1} \lambda_0' \xrightarrow{i_1'/o_1/\rho_1'} \lambda_1 \cdots \xrightarrow{d_k} \lambda_{k-1}' \xrightarrow{i_k'/o_k/\rho_k'} \lambda_k.$$

An *isomorphism* from σ to σ' is a list $f = f_0, \ldots, f_k$ of bijections such that (1) f is an isomorphism from $untime(\sigma)$ to $untime(\sigma')$, and $\lambda_j = \kappa_j \circ f_j^{-1}$ and $\lambda'_j = \kappa'_j \circ f_j^{-1}$, for all j. Since σ' is fully determined by σ and f, we write $\sigma' = f(\sigma)$. Two timed behaviors σ and σ' are *isomorphic* if there exists an isomorphism f from σ to σ'. 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 σ be a timed behavior and let f be an isomorphism for σ. Then $untime(f(\sigma)) = f(untime(\sigma))$.*

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

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

Lemma 3.3. *Suppose $\mathcal{M}, \mathcal{N}$ are MMTs with $\mathcal{M} \not\approx_{untimed} \mathcal{N}$. Then there exists a feasible untimed behavior β of $\mathcal{M}$ that is not isomorphic to any feasible untimed behavior of $\mathcal{N}$.*

Untimed equivalence is finer than timed equivalence:

Theorem 3.4. *$\mathcal{M} \approx_{untimed} \mathcal{N}$ implies $\mathcal{M} \approx_{timed} \mathcal{N}$.*

Proof. Assume $\mathcal{M} \approx_{untimed} \mathcal{N}$ and w is a timed word of $\mathcal{M}$. Since $\approx_{timed}$ is symmetric, it suffices to prove that w is a timed word of $\mathcal{N}$. Since w is a timed word of $\mathcal{M}$, there exists a timed run α of $\mathcal{M}$ with $tw(\alpha) = w$. Let $\sigma = beh(\alpha)$ and $\beta = untime(\sigma)$. Then β is a feasible untimed behavior of $\mathcal{M}$ and $tw(\sigma) = w$. Since $\mathcal{M} \approx_{untimed} \mathcal{N}$, there exists an isomorphism f such that $\beta' = f(\beta)$ is a feasible untimed behavior of $\mathcal{N}$. Hence $\mathcal{N}$ has an untimed run γ' such that $untime(\gamma') = \beta'$. Let $\sigma' = f(\sigma)$. By Lemma 3.1, σ' is a timed behavior with $untime(\sigma') = untime(f(\sigma)) = f(untime(\sigma)) = f(\beta) = \beta'$. Since $beh(\gamma') = untime(\sigma') = \beta'$, $\mathcal{N}$ has a timed run $\alpha' = pullback(\gamma', \sigma')$ with $beh(\alpha') = \sigma'$. Note that $tw(\alpha') = tw(\sigma') = tw(f(\sigma)) = tw(\sigma) = w$. Hence w is a timed word of $\mathcal{N}$, 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 $\mathcal{M}$ is *timer live* if, for each feasible untimed behavior β and for each timer y that is running after β, there exists an untimed behavior β_y consisting of transitions that leave y unaffected, except for the last one in which y expires, and such that $\beta \cdot \beta_y$ is feasible. Clearly, the MMT of Figure 7 is not timer live, as there is no way to extend the feasible untimed behavior $\emptyset \xrightarrow{i/o/x:=1} \{x\} \xrightarrow{i/o/y:=60} \{x, y\}$ to an untimed behavior in which timer y expires.

3.3 Equivalence timed/untimed 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 prepatory work.

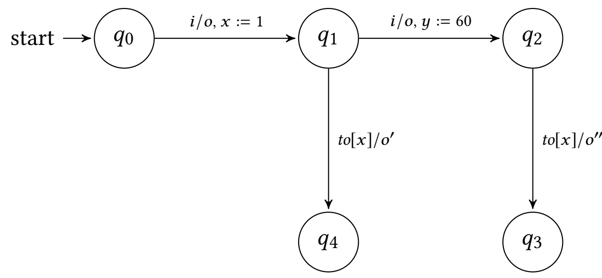

Figure 7. MMT with ghost timer y

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

$$\kappa_0 \xrightarrow{d_1} \kappa'_0 \xrightarrow{i_1/o_1/\rho_1} \kappa_1 \cdots \xrightarrow{d_j} \kappa'_{j-1} \xrightarrow{i_j/o_j/\rho_j} \kappa_j \xrightarrow{d_{j+1}} \cdots$$

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 κ, κ' such that

$$\kappa_0 \xrightarrow{d_1} \kappa'_0 \xrightarrow{i_1/o_1/\rho_1} \kappa_1 \cdots \xrightarrow{e} \kappa' \xrightarrow{i_j/o_j/\rho_j} \kappa \xrightarrow{e'} \cdots$$

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 σ 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) $i_q = to[x]$. A *block* of σ is a maximal subset of indices $B = \{p_1, \ldots, p_u\}$ such that i_{p_1} triggers i_{p_2}, i_{p_2} triggers i_{p_3}, etc. Note that the collection of blocks of σ partitions the set of indices $\{1, \ldots, k\}$. We refer to this partition as Π_σ. We say that timed behavior σ contains a *race* if there is some index $j > 0$ and some timer x such that $\kappa'_{j-1}(x) = 0$ and $i_j \neq to[x]$. 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 σ is a timed behavior as in equation (1) without races. Suppose $B = \{p_1, \ldots, p_u\}$ is a block of σ, and suppose that ϵ is a real number whose absolute is smaller than any nonzero number that occurs in σ, that is $|\ \epsilon\ | < \min(\bigcup_{1 \leq j \leq k} \{d_j\} \cup ran(\kappa'_j) \setminus \{0\})$. Then there exists a timed behavior $\sigma' = \lambda_0 \xrightarrow{e_1} \lambda'_0 \xrightarrow{i_1/o_1/\rho_1} \lambda_1 \xrightarrow{e_2} \lambda'_1 \xrightarrow{i_2/o_2/\rho_2} \lambda_2 \cdots \xrightarrow{e_k} \lambda'_{k-1} \xrightarrow{i_k/o_k/\rho_k} \lambda_k$ without races such that $untime(\sigma) = untime(\sigma')$, $\kappa_0 = \lambda_0$, and $e_j = d_j + \epsilon$ if $j-1 \notin T \wedge j \in T$, $e_j = d_j$ if $j-1 \in T \Leftrightarrow j \in T$, and $e_j = d_j - \epsilon$ if $j-1 \in T \wedge j \notin T$.*

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:

$$\emptyset \xrightarrow{7} \emptyset \xrightarrow{i1/o1/x \mapsto 2} (x = 2) \xrightarrow{1} (x = 1) \xrightarrow{i2/o2/y \mapsto 1} (x = y = 1)$$

$$\xrightarrow{1} (x = y = 0) \xrightarrow{to[x]/o3/u\mapsto 2} (u = 2) \xrightarrow{1} (u = 1) \xrightarrow{i4/o4/z\mapsto 1}$$
$$(u = z = 1) \xrightarrow{1} (u = z = 0) \xrightarrow{to[z]/o5} (u = 0).$$

This timed behavior contains two races, after two and four time units, respectively. The first races 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 $\{1,3\}$.

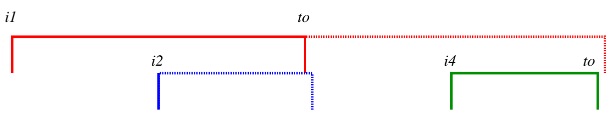

Figure 8. A timed behavior with two races

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

Proof. (Sketch) Let

$$\sigma \quad = \quad \kappa_0 \xrightarrow{d_1} \kappa_0' \xrightarrow{i_1/o_1/\rho_1} \kappa_1 \cdots \xrightarrow{d_k} \kappa_{k-1}' \xrightarrow{i_k/o_k/\rho_k} \kappa_k$$

be a timed behavior. For each index j, each timer in the domain of κ_j has been started by some preceding event. Let $startedby_j$: $\mathrm{dom}(\kappa_j) \to \Pi_\sigma$ be the function that maps each timer in the domain of κ_j to the block that contains the event that started this timer. Suppose that σ contains a race, that is, there is an index $j > 0$ and a timer x such that $\kappa_{j-1}'(x) = 0$ and $i_j \neq to[x]$. Let $B \in \Pi_\sigma$ be the block containing j. Then we say that block B is the *winner* of the race and block $startedby_{j-1}(x)$ 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 σ than any event of B'. This implies that the winning relation induces a partial order on the blocks of Π_σ. 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 σ 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 β there exists a transparent timed behavior σ such that $\beta = untime(\sigma)$.*

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

The following fundamental lemma now follows:

Lemma 3.8. *Suppose β is a feasible untimed behavior that ends with timer set Y, and $Y \xrightarrow{i/o/\rho} Y'$ is an untimed behavior with $i \in I$. Then $\beta \xrightarrow{i/o/\rho} Y'$ is a feasible untimed behavior.*

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

Causality maps. Consider an untimed behavior

$$\beta \quad = \quad X_0 \xrightarrow{i_1/o_1/\rho_1} X_1 \xrightarrow{i_2/o_2/\rho_2} X_2 \cdots \xrightarrow{i_k/o_k/\rho_k} X_k.$$

A causality map for β is a function that specifies, for each timer that expires, the index of the event that triggered this timeout. Formally, let $T = \{j \mid i_j \in TO[X]\}$ be the set of indices of β corresponding to a timeout. A *causality map* for β is a function $c : T \to \{1, \ldots, k\}$ that assigns to each index j with $i_j = to[x]$ an index $l < j$ such that ρ_l starts x, and all events in between i_l and i_j do not affect x.

Lemma 3.9. *Each untimed behavior β 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 σ if it is a causality map of $untime(\sigma)$, and we say that it is a causality map of a timed run α if it is a causality map of $beh(\alpha)$. 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 $T = \{j \mid i_j = to\}$ be the set of indices corresponding to a timeout. A *causality map* for w is a function $c : T \to \{1, \ldots, k\}$ 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, $\sum_{l=j+1}^{c(j)} d_l$ is an integer (timers expire after an integer delay).

Lemma 3.10. *Suppose α is a timed run of an MMT and c is the causality map of α. 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 α is a timed run of an MMT, and $w = tw(\alpha)$ is transparent. Then w has a unique causality map.*

Each timed word of MMT $\mathcal{M}$ 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 $\mathcal{M}$ is fully determined by the sequence of time delays and events that occurs.

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

We are now prepared to state our first main result:

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

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

Since β be a feasible untimed behavior of $\mathcal{M}$, $\mathcal{M}$ has an untimed run γ with $beh(\gamma) = \beta$. By Lemma 3.7, there exists a transparent timed behavior σ such that $\beta = untime(\sigma)$. There exists a unique timed run $\alpha = pullback(\gamma, \sigma)$ of $\mathcal{M}$ with $untime(\alpha) = \gamma$ and $beh(\alpha) = \sigma$. Thus σ is a timed behavior of $\mathcal{M}$. Let $w = tw(\alpha)$. Then w is a transparent timed word of $\mathcal{M}$. By Lemma 3.11, w has a unique causality map c. Since $\mathcal{M} \approx_{timed} \mathcal{N}$, w is also a timed word of $\mathcal{N}$. By Lemma 3.12, $\mathcal{N}$ has a unique time run α' such that $w = tw(\alpha')$. Let $\beta' = untime(beh(\alpha'))$. Then β' is a feasible untimed behavior of $\mathcal{N}$. 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 β and β' have the same length, the same inputs (except for the timer names), and the same outputs. Moreover, by Lemmas 3.9 and 3.10, β' and β have the same causality map c.

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

Induction base. If β and β' contain 0 events then they are both equal to the empty set of variables $\emptyset$, and thus trivially isomorphic.

For the induction step, suppose β and β' contain $k+1$ events:

$$\begin{aligned} \beta &= X_0 \xrightarrow{i_1/o_1/\rho_1} X_1 \cdots \xrightarrow{i_k/o_k/\rho_k} X_k \xrightarrow{i_{k+1}/o_{k+1}/\rho_{k+1}} X_{k+1}, \\ \beta' &= Y_0 \xrightarrow{i'_1/o_1/\tau_1} Y_1 \cdots \xrightarrow{i'_k/o_k/\tau_k} Y_k \xrightarrow{i'_{k+1}/o_{k+1}/\tau_{k+1}} Y_{k+1}. \end{aligned}$$

Let δ and δ' be the prefixes of β and β', respectively, containing k events. Then δ is also a feasible untimed behavior and, by induction hypothesis, there exists an isomorphism $f = f_0, \ldots, f_k$ such that $\delta' = f(\delta)$. Our task is to extend this isomorphism to β and β'.

Since tw, $untime$ and beh preserve inputs except for the timers in timeouts, either $i_{k+1} = i'_{k+1} \in I$ or $i_{k+1}, i'_{k+1} \in TO[X]$. If $i_{k+1} = to[x]$, for some $x \in X_k$, then i_{k+1} is triggered by a previous event i_j with $j = c(k+1)$ that started timer x, and this timer was left unaffected by all events from β in between i_j and i_{k+1}. In this case, $i'_{k+1} = to[x']$, for some $x' \in X_k$, and i'_{k+1} is triggered by a previous event i'_j with $j = c(k+1)$ that started timer x', and this timer was left unaffected by all events from β' in between i_j and i_{k+1}. Using the definition of an isomorphism, we may infer that $i'_{k+1} = to[f_k(x)]$.

Now suppose that $x \in X_{k+1}$ is a timer that is active after β. Then, since $\mathcal{M}$ is timer live, there exists an untimed behavior β_x consisting of transitions that leave x unaffected, except for the last one in which x expires, and such that $\beta \cdot \beta_x$ is a feasible untimed behavior of $\mathcal{M}$. Using the same construction as in the beginning of this proof, we may construct an untimed behavior β'_x such that $\beta' \cdot \beta'_x$ is a feasible untimed behavior of $\mathcal{N}$ such that $\beta \cdot \beta_x$ and $\beta' \cdot \beta'_x$ have the same length, the same inputs (except for the timer names) and the same causality map c_x. Let m be the index of the final event in $\beta' \cdot \beta'_x$. Then m is in the domain of c_x and $c_x(m) \leq k+1$. If $c_x(m) \leq k$ then we define $f_{k+1}(x) = f_k(x)$. Otherwise, if $c_x(m) = k+1$ then we know that timer x is started in the last transition of β. Since c_m is also a causality map for β', there is also a unique timer $x' \in Y_{k+1}$ that is started in the last transition of β'. In this case, we define $f_{k+1}(x) = x'$. Repeating this construction for all timers $x \in X_{k+1}$, we define a function $f_{k+1} : X_{k+1} \to Y_{k+1}$ and thus extend isomorphism f to β and β'. Note that f_{k+1} is surjective because for each $y \in Y_{k+1}$, since $\mathcal{N}$ is timer live, there exists an untimed behavior β_y consisting of transitions that leave y unaffected, except for the last one in which y expires, and such that $\beta' \cdot \beta_y$ is a feasible untimed behavior of $\mathcal{N}$. Using again the same construction as in the beginning of this proof, we may establish that X_{k+1} contains a corresponding timer. □

Note that in the above proof we only assume that $\mathcal{M}$ and $\mathcal{N}$ have the same transparent timed words. If we combine this observation with Theorem 3.4, it follows that $\mathcal{M}$ and $\mathcal{N}$ have the same timed words iff they have the same transparent timed words. Thus all the information about the behavior of an MMT is contained in its transparent timed words. This allows us to prove the following lemma:

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

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 $\mathcal{M}$ but now the learner poses membership queries to learn the *untimed* behaviors of $\mathcal{M}$, and if an equivalence query fails then the counterexample is an *untimed* behavior of $\mathcal{M}$. 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 $\mathcal{M}$. It brings untimed behaviors into a "canonical" form, so that a learner can only observe these behaviors up to isomorphism. Let us formalize this idea. Suppose β be an untimed behavior in which transitions update at most one timer. We say that β 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 β in which transitions update at most one timer, there is a unique untimed behavior β' in canonical form that is isomorphic to β. We write $can(\beta)$ to denote this β'.

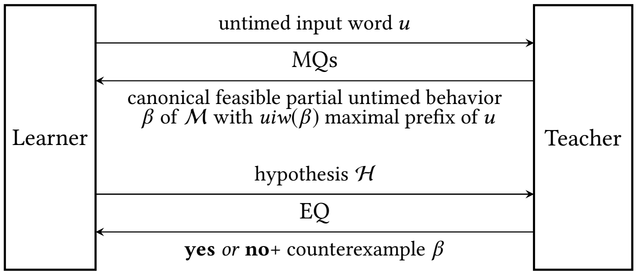

Figure 9. Untimed MAT framework

Consider an untimed behavior β in canonical form:

$$\beta \quad = \quad X_0 \xrightarrow{i_1/o_1,\rho_1} X_1 \cdots \xrightarrow{i_k/o_k,\rho_k} X_k.$$

Let $1 \leq l \leq j \leq k$, $d \in \mathbb{N}^{>0}$ and suppose that (1) ρ_l either starts no timer or sets x_l to d, (2) β does not contain a timeout event $to[x_l]$. Then addtimer(l, d, j, β) is the untimed behavior obtained from β by replacing ρ_l by the update $x_l := d$, and adding x_j to the sets X_l up to (and including) X_j. Write $\beta \sqsubseteq \beta'$ if β' can be obtained from β by zero or more applications of function addtimer. Observe that $\sqsubseteq$ 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 β is a *partial* untimed behavior of MMT $\mathcal{M}$ iff $\mathcal{M}$ has an untimed behavior β' such that $\beta \sqsubseteq \beta'$.

An *untimed input word* over I is a sequence $u = i_1 \cdots i_k$ over $\hat{I}$ such that (1) for all indices j, l, $i_j = to[x_l]$ 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 β be an untimed behavior in canonical form. We can associate a unique input word $uiw(\beta)$ to β by removing all the output events, updates, and timer sets from β.

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 set of inputs I of $\mathcal{M}$. Again, the learer 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 β of 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 $\mathcal{H} \approx_{untimed} \mathcal{M}$. Otherwise it answers *no* and supplies a counterexample, which now is a feasible untimed behavior β (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 β be an untimed behavior in canonical form:

$$\beta \quad = \quad \emptyset \xrightarrow{i_1/o_1/\rho_1} X_1 \cdots X_{k-1} \xrightarrow{i_k/o_k/\rho_k} X_k.$$

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

$$\kappa_0 \xrightarrow{d_1} \kappa_0' \xrightarrow{i_1/o_1/\rho_1} \quad \cdots \quad \xrightarrow{d_k} \kappa_k' \xrightarrow{i_k/o_k/\rho_k} \kappa_k \xrightarrow{d_{k+1}} \kappa_{k+1}'$$

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 jth transition in β. We now associate with β a conjunction of difference constraints over $t_1, \ldots, t_{k+1}$, denoted constraints(β), consisting of

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

By standard DBM techniques, we can check whether constraints(β) 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 $t_{j+1} - t_j$ be positive for each j. We infer that β is feasible iff constraints(β) is satisfiable. Moreover, if β is feasible, we can use the techniques of Section 3.3 to construt a solution such that frac$(t_j) \neq$ frac(t_l) for each pair of distinct indices j and l with $i_j, i_l \in I$.

From the saturated version of constraints(β), we also derive a constraint, denoted Zone(β), which characterizes the possible timer valuations κ_k in a timed behavior of the above form. The constraint Zone(β) contains for each pair of timers x_i, x_j in X_k the conjunct

$$(m + \rho_j(x_j) - \rho_i(x_i)) < \kappa_k(x_j) - \kappa_k(x_i) < (n + \rho_j(x_j) - \rho_i(x_i))$$

whenever the saturated version of constraints(β) contains the conjunct $m < t_j - t_i < n$. (This can be derived using $\kappa_k(x_j) = \rho_j(x_j) - (t_k - t_j)$ and $\kappa_k(x_i) = \rho_i(x_i) - (t_k - t_i)$).

We can derive the following lemmas.

Lemma 4.1. *Suppose β, β' are untimed behaviors such that* Zone$(\beta) =$ Zone(β'). *Let γ be any untimed behavior. Then $\beta \cdot \gamma$ is feasible iff $\beta' \cdot \gamma$ is feasible.*

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

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

Lemma 4.3. *Suppose β is a feasible untimed behavior with* Last$(\beta) =$ *Y and $Y \xrightarrow{to[x]/o/\rho} Y'$ is an untimed behavior. Then x is expirable after β iff $\beta \xrightarrow{to[x]/o/\rho} Y'$ 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 $d_{\max}$ to store an estimate of the maximal timeout value that occurs in $\mathcal{M}$. Initially, the adapter sets $d_{\max}$ 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 β 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 β 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 $i_j \in I$. Let $N := B\ i_j/\omega/\rho_0\ \emptyset$, where $\omega \in O$ is an arbitrary output that acts as placeholder and ρ_0 is the empty update. (We omit the arrow for $i_j/\omega/\rho_0$ 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 $w = d_1\ i_1\ o_1\ d_2\ i_2\ o_2 \cdots d_j\ i_j\ \omega$ is a transparent timed word. Let $u' = tiw(w)$ 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 $B := B\ i_j/o_j/\rho_0\ \emptyset$, 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 $B :=$ addtimer$(l, e, m-1, B)$ and finish the body of the loop.
2. Case $i_j = to[x_l]$, for $l < j$. If timer x_l is started in B and initialized with the value e then let $N :=$ addtimer$(l, e, j-1, B)\ to[x_l]/\omega/\rho_0\ \emptyset$. Check whether constraints(N) is satisfiable. If so proceed to (*), otherwise exit the while loop. If timer x_l is not started in B, let $N^e =$ addtimer$(l, e, j-1, B)\ to[x_l]/\omega/\rho_0\ \emptyset$, for $e = 1, \ldots, d_{\max}$. 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 $w = d_1\ i_1\ o_1\ d_2\ i_2\ o_2 \cdots d_j\ to[x_l]\ \omega$ is a transparent timed word. Let $u' = tiw(w)$ 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_l, the value e' to which this timer is set is different from e. In this case set $B :=$ addtimer$(l, e', j-1, B)\ to[x_l]/o_j/\rho_0\ \emptyset$, 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 $B :=$ 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 $\mathcal{H} \approx_{timed} \mathcal{M}$. In this case, by Theorem 3.13, $\mathcal{H} \approx_{untimed} \mathcal{M}$, 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 $\mathcal{M}$ but not of $\mathcal{H}$. In this case, by Theorem 3.4, we may conclude that $\mathcal{H} \not\approx_{untimed} \mathcal{M}$, and thus the untimed teacher should also return a result *no*. Since w is a transparent timed word of $\mathcal{M}$, 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 β in canonical form that is a partial untimed behavior of $\mathcal{M}$ but not of $\mathcal{H}$: the causality map tells us exactly which timers are set and timeout. Thus the untimed teacher may return β as counterexample. If our estimate $d_{\max}$ of the maximal timeout value of $\mathcal{M}$ is too low, then it may occur that in w the timeout value of some timer is larger than $d_{\max}$. In this case, the value of $d_{\max}$ 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*: $\beta \in S \Rightarrow \mathrm{Head}(\beta) = \emptyset$,
- *prefix closed*: $\beta \cdot \gamma \in S \Rightarrow \beta \in S$,
- *behavior deterministic*: $\beta \xrightarrow{i/o_1/\rho_1} X_1 \in S \wedge \beta \xrightarrow{i/o_2/\rho_2} X_2 \in S \Rightarrow o_1 = o_2 \wedge \rho_1 = \rho_2 \wedge X_1 = X_2$,
- *input complete*: $\beta \in S \wedge i \in I \Rightarrow \exists o, \rho, Y : \beta \xrightarrow{i/o/\rho} Y \in S$,
- *timeout complete*: $\beta \in S \wedge x$ expirable after $\beta \Rightarrow \exists o, \rho, Y : \beta \xrightarrow{to[x]/o/\rho} Y \in S$.

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 *lean*(β) be the lean behavior obtained from a canonical behavior β. 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 $i_1/o_1/\rho_1 \cdots i_n/o_n/\rho_n$ of its labels. If some ρ_j 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 $mem_S(\beta)$ be the set of timers x_i in $x_1, \ldots x_{|\beta|}$ whose corresponding timeout event (of form $to[x_i]$) occurs (as an input) in som continuation β' of β in S. Let $val_{S,\beta}$ map each timer x_i in $mem_S(\beta)$ 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 $mem_S(\beta)$ be the set of timers after β, and assigning each occurring timer to $val_{S,\beta}(x_i)$ in the ith 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 "$\beta \cdot \gamma \in S \iff \beta' \cdot \gamma \in S$". 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 $y_i \in Y$, and let $\tilde{I}$ be $I \cup TO[X] \cup TO[Y]$. Define a *lean suffix behavior* (*lean suffix* for short) to be a sequence $i_1/o_1/\rho_1 \cdots i_m/o_m/\rho_m$ of input/output/assignment triples, in which each i_j is in $\tilde{I}$, each ρ_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 $k \geq 0$, 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 langues. For a (lean) timer language S and lean behavior $\beta \in S$, let $\beta^{-1}S$ be the set of lean suffixes γ such that there is a canonical behavior β' with $\beta \sqsubseteq \beta'$ such that $\beta' \cdot g_{+|\beta|}(\gamma) \in S$. Then $mem_S(\beta)$ is the set of timers x_i in $x_1, \ldots x_{|\beta|}$ whose corresponding timeout event (of form $to[x_i]$) occurs (as an input) in some suffix $\hat{\gamma}$ in $\beta^{-1}S$, and $val_{S,\beta}$ maps each timer x_i in $mem_S(\beta)$ to the unique positive integer to which it is assigned in the corresponding lean behavior $\beta' \cdot g_{+|\beta|}(\hat{\gamma})$ in S.

We can then define the Nerode equivalence.

Definition 5.2. Let S be a lean timer language with $\beta, \beta' \in S$, let $f : mem_S(\beta) \rightarrow mem_S(\beta')$ be a bijection from $mem_S(\beta)$ to $mem_S(\beta')$. Then β and β' are *equivalent* under f, written $\beta \equiv_S^f \beta'$ iff

$$\gamma \in \beta^{-1}S \qquad \text{iff} \qquad f(\gamma) \in \beta'^{-1}S$$

Intuitively, $\beta \equiv_S^f \beta'$ means that β and β' allow the same suffixes, after renaming timers assigned in β by f. We write $\beta \equiv_S \beta'$ to denote that $\beta \equiv_S^f \beta'$ for some $f : mem_S(\beta) \rightarrow mem_S(\beta')$.

Example Let S contain the lean behaviors

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

Let $\beta_1 = i_1/o_1$ and $\beta_2 = i_1/o_1 \cdot i_2/o_2$. Then ${\beta_1}^{-1}S$ contains the suffix $\gamma_1 = to[x_1]/o_3$ and ${\beta_2}^{-1}S$ contains the suffix $\gamma_2 = to[x_2]/o_3$. It is now possible that $\beta_1 \equiv_S^f \beta_2$, where f maps x_1 to x_2 (whether $\beta_1 \equiv_S^f \beta_2$ 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 $\equiv_S$ 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 $f(\gamma)$ with $uiw(\gamma)$ and $uiw(f(\gamma))$ in V. Already here, we see that it is convenient to let V be closed under permutations on timers in X, so that $uiw(\gamma) \in V$ iff $uiw(f(\gamma)) \in V$. Let us call such a set *adequate*. For an adequate set V of input suffixes, and a lean behavior β, let $(\beta^{-1}S)|_V$ be the set of suffixes γ with $uiw(\gamma) \in V$. Let $mem_{S,V}(\beta)$ be the set of timers x_i in $x_1, \ldots x_{|\beta|}$ whose corresponding timeout event (of form $to[x_i]$) occurs (as an input) in some suffix in $(\beta^{-1}S)|_V$. Let $val_{S,V,\beta}$ map each timer x_i in $mem_{S,V}(\beta)$ to the unique positive integer to which it is assigned in the ith 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 $f : mem_{S,V}(\beta) \rightarrow mem_{S,V}(\beta')$ be a bijection from $mem_{S,V}(\beta)$ to $mem_{S,V}(\beta')$. Then β and β' are *equivalent wrp V* under f, written $\beta \equiv_{S,V}^f \beta'$ iff

$$\gamma \in (\beta^{-1}S)|_V \qquad \text{iff} \qquad f(\gamma) \in (\beta'^{-1}S)|_V$$

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

Theorem 6.2. *Let S and V be as above. $\equiv_S$ is included in $\equiv_{S,V}$. Moreover, if $\equiv_S$ has finite index, then it is equal to $\equiv_{S,V}$ 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 V.

Let us introduce the two conditions for convergence of the construction phase. For a lean behavior $\beta \in S$ and $\gamma \in \beta^{-1}S$, let $\beta;_S \gamma$ be the (unique) lean behavior $\beta' \cdot g_{+|\beta|}(\gamma)$ in S with $\beta \sqsubseteq \beta'$. Let $en_S(\beta)$ be the set of $i \in \hat{I}$ such that $\beta;_S i/o \in S$ for some o (recall that the last assignment of a lean behavior is always empty). For $i \in en_S(\beta)$, let $\lambda(\beta, i)$ be the unique output o such that $\beta;_S i/o \in S$. 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 $i \in en_S(\beta)$ there is a $\beta' \in U$ such that $\beta;_S i/\lambda(\beta,i) \equiv_{S,V} \beta'$.
- U is *timer-consistent* wrt. V if for each $\beta \in U$ and $i \in en_S(\beta)$ we have $mem_{S,V}(\beta;_S i/\lambda(\beta,i)) \subseteq (mem_{S,V}(\beta) \cup \{x_{(|\beta|+1)}\})$.

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 β 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 $\mathcal{H}(U,V) = (I,O,Q,q_0,\mathcal{X},\delta,\lambda,\pi)$, where

- $Q = U$ and $q_0 = \epsilon$,
- $\mathcal{X}$ maps each location $\beta \in U$ to $mem_{S,V}(\beta)$,
- Let $\beta \in U$ and $i \in en_S(\beta)$. Then
  - $\lambda(\beta,i)$ is the unique o such that $\beta;_S i/o \in S$.
  - $\delta(\beta,i)$ is is the unique $\beta' \in U$ such that there is an f with $\beta;_S i/\lambda(\beta,i) \equiv^f_{S,V} \beta'$.
  - $\pi(\beta,i) : mem_{S,V}(\beta') \mapsto (mem_{S,V}(\beta) \cup \mathbb{N}^{>0})$ is defined as f^{-1} on $mem_{S,V}(\beta')$, except that it maps $f(x_{(|\beta|+1)})$ to $val_{S,V,\beta'}(f(x_{(|\beta|+1)}))$ if $f(x_{(|\beta|+1)}) \in mem_{S,V}(\beta')$.
- When $\beta \in U$ and i is of form $to[x_j]$ with $x_i\, mem_{S,V}(\beta)$ but $to[x_j] \notin en_S(\beta)$, we let $\lambda(\beta,i) = \perp$, $\delta(\beta,i) = \beta$, and let $\pi(\beta,i)$ be the identity mapping on $mem_{S,V}(\beta)$.

The last case (where $to[x_j] \notin en_S(\beta)$) constructs a transition that is not feasible, but which must anyway be syntactically present, since x_j may expire after some continuation if β and is hence live.

Hypothesis construction, performs membership queries in order to construct the sets of form $(\beta^{-1}S)|_V$ and $(\beta;_S i/\lambda(\beta,i)^{-1}S)|_V$ for $\beta \in U$ and $i \in en_S(\beta)$. Moreoever, 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 $uiw(\beta) \cdot i$ for $\beta \in U$ and $i \in \hat{I}$: this allows to determine $en_S(\beta)$ and $\lambda(\beta,i)$ for $\beta \in U$ and $i \in en_S(\beta)$. Thereafter, membership queries are performed for all untimed input words of form $uiw(\beta)\cdot v$ and $uiw(\beta)\cdot i\cdot v$, where $\beta \in U, v \in V$, and $i \in en_S(\beta)$. Note that one need only consider v in which timeouts from $TO[X]$ concern timers in $\{x_1,\ldots,x_{|\beta|}\}$ (or in $\{x_1,\ldots,x_{(|\beta|+1)}\}$ for $uiw(\beta)\cdot i$). This allows to construct the sets of form $(\beta^{-1}S)|_V$ and $((\beta;_S i/\lambda(\beta,i))^{-1}S)|_V$ for $\beta \in U$. It then allows to compute the approximated Nerode equivalence $\equiv_{S,V}$ on the set of lean behaviors of form β and $\beta;_S i/\lambda(\beta,i)$ with $\beta \in U$ and $i \in en_S(\beta)$. 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 $en_S(\beta)$ for which there is no $\beta' \in U$ such that $\beta;_S i/\lambda(\beta,i) \equiv_{S,V} \beta'$, 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 $mem_{S,V}(\beta;_S i/\lambda(\beta,i)) \setminus (mem_{S,V}(\beta) \cup \{x_{(|\beta|+1)}\})$, find a lean suffix γ in $((\beta;_S i/\lambda(\beta,i))^{-1}S)|_V$, whose last input is $to[x_j]$. This input is obviously missing from $(\beta^{-1}S)|_V$. But, by adding the concatenation of i with $uiw(\gamma)$ (where timers are appropriately renamed), the input $to[x_j]$ will also appear in $mem_{S,V}(\beta)$. This extension of V will subsequently trigger new membership queries.

When U is closed and timer-consistent wrt. V, then a hypothesis MMT $\mathcal{H}(U,V)$ 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 α 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 v such that U is no longer closed wrt. V. as follows. We assume w.l.o.g. that no proper prefix of α is a counterexample. By the fact that α is a counterexample, we can find a lean suffix γ of α, such that $\beta\cdot i/o \equiv^f_{S,V} \beta'$ but $\gamma \in (\beta;_S i/\lambda(\beta,i))^{-1}S \not\Leftrightarrow \gamma \in \beta'^{-1}S$ for some $\beta,\beta' \in U$ such that $\beta;_S i/o \equiv^f_{S,V} \beta'$ is used to construct the transition triggered by i from β in $\mathcal{H}(U,V)$. To se this, let $\alpha = i_1/o_1/\rho_1 \cdots i_n/o_n/\rho_n$, and for $j = 0,\ldots,n$, define a lean behavior β_j and a β_j-suffix as follows.

- $\beta_0 = \epsilon$, and γ_0 is obtained by making α a lean suffix (i.e., renaming each timer x_l to y_l).
- Let γ_{j-1} be of form $\hat{i_j}/\hat{o_j}/\hat{\rho_j} \cdot \gamma'_{j-1}$, let β_j be $\lambda(\beta_{j-1},\hat{i_j})$, let f_j be the mapping used to establish $\beta_{j-1};_S \hat{i_j}/\hat{o_j} \equiv^{f_j}_{S,V} \beta_j$ in the construction of $\mathcal{H}(U,V)$, and let γ_j be obtained from γ'_{j-1} by (i) applying f_j to timers in $\{x_1,\ldots,x_{\beta_{j-1}}\}$, (ii) replacing y_1 by $f_j(x_{\beta_{j-1}+1})$, and (ii) replacing each y_l by y_{l-1}.

The result is that $\beta_0\ldots\beta_n$ is the sequence of states visited when $\mathcal{H}(U,V)$ processes α, and γ_j is lean suffix that can be composed with β_j, which corresponds to a suffix of α. By the fact that α is a counterexample, we have $\gamma_0 \in {\beta_0}^{-1}S \not\Leftrightarrow \gamma_n \in {\beta_n}^{-1}S$ (since γ_n is the empty sequence), which implies that $\gamma_{j-1} \in {\beta_{j-1}}^{-1}S \not\Leftrightarrow \gamma_j \in {\beta_j}^{-1}S$ for some j; we can then take β_{j-1} as β and β_j as β', and γ as γ_j. This means that γ is a new separating suffix, and that V should be extended with $uiw(\gamma)$. After adding v (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 $\mathcal{M}$ 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, we 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 $\beta \cdot i/o \equiv^{f}_{S,V} \beta'$, 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 consitutes 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

[1] D. Angluin. 1987. Learning Regular Sets from Queries and Counterexamples. *Inf. Comput.* 75, 2 (1987), 87–106.

[2] M. Benedikt, C. Ley, and G. Puppis. 2010. What you must remember when processing data words. *CEUR Workshop Proceedings* 619 (2010).

[3] J. Bengtsson and W. Yi. 2003. Timed Automata: Semantics, Algorithms and Tools. in *Advances in Petri Nets, LNCS* 3098 (2003), 87–124.

[4] B. Caldwell, R. Cardell-Oliver, and T. French. 2016. Learning Time Delay Mealy Machines From Programmable Logic Controllers. *IEEE Trans. on Automation Sc. and Eng.* 13, 2 (2016), 1155–1164.

[5] S. Cassel, F. Howar, B. Jonsson, M. Merten, and B. Steffen. 2015. A succinct canonical register automaton model. *J. Log. Algebr. Meth. Pr.* 84, 1 (2015), 54–66.

[6] S. Cassel, F. Howar, B. Jonsson, and B. Steffen. 2016. Active learning for extended finite state machines. *Formal Asp. Comput.* 28, 2 (2016), 233–263.

[7] J. de Ruiter and E. Poll. 2015. Protocol State Fuzzing of TLS Implementations. In *USENIX Security Symp.* USENIX, 193–206.

[8] D. Dill. 1990. Timing assumptions and verification of finite-state concurrent systems. in *CAV, LNCS* 407 (1990), 197–212.

[9] S. Drews and L. D'Antoni. 2017. Learning Symbolic Automata. in *TACAS, LNCS* 10205, 173–189.

[10] P. Fiterau-Brostean and F. Howar. 2017. Learning-Based Testing the Sliding Window Behavior of TCP Implementations. in *FMICS, LNCS* 10471 (2017), 185–200.

[11] P. Fiterău-Broştean, R. Janssen, and F.W. Vaandrager. 2016. Combining Model Learning and Model Checking to Analyze TCP Implementations. in *CAV, LNCS* 9780 (2016), 454–471.

[12] P. Fiterău-Broştean, R. Lenaerts, E. Poll, and etal. 2017. Model Learning and Model Checking of SSH Implementations. In *SPIN Symp.* ACM, 142–151.

[13] O. Grinchtein, B. Jonsson, and M. Leucker. 2010. Learning of event-recording automata. *Theoretical Computer Science* 411, 47 (2010), 4029 – 4054.

[14] O. Grinchtein, B. Jonsson, and P. Pettersson. 2006. Inference of Event-Recording Automata Using Timed Decision Trees. in *CONCUR, LNCS* 4137 (2006), 435–449.

[15] A. Hagerer, H. Hungar, O. Niese, and B. Steffen. 2002. Model Generation by Moderated Regular Extrapolation. in *FASE, LNCS* 2306 (2002), 80–95.

[16] F. Howar, B. Steffen, B. Jonsson, and S. Cassel. 2012. Inferring Canonical Register Automata. in *VMCAI, LNCS* 7148 (2012), 251–266.

[17] F. Howar, B. Steffen, and M. Merten. 2011. Automata Learning with Automated Alphabet Abstraction Refinement. in *VMCAI, LNCS* 6538 (2011), 263–277.

[18] M. Isberner, F. Howar, and B. Steffen. 2015. The Open-Source LearnLib - A Framework for Active Automata Learning. in *CAV, LNCS* 9206 (2015), 487–495.

[19] J. F. Kurose and K. W. Ross. 2013. *Computer Networking: a Top-Down Approach.* Pearson. Sixth edition.

[20] D. Lee and M. Yannakakis. 1996. Principles and methods of testing finite state machines — a survey. *Proc. IEEE* 84, 8 (1996), 1090–1123.

[21] S.-W. Lin, E. André, Y. Liu, J.Sun, and J.S. Dong. 2014. Learning assumptions for compositional verification of timed systems. *IEEE Trans. SE* 40, 2 (2014), 137–153.

[22] A. Maier. 2014. Online passive learning of timed automata for cyber-physical production systems. In *INDIN*. IEEE, 60–66.

[23] I.-E. Mens and O. Maler. 2015. Learning Regular Languages over Large Ordered Alphabets. *Logical Methods in Comp. Sc.* 11, 3 (2015).

[24] D. Peled, M.Y. Vardi, and M. Yannakakis. 2002. Black Box Checking. *Journal of Automata, Languages, and Combinatorics* 7, 2 (2002), 225–246.

[25] H. Raffelt, M. Merten, B. Steffen, and T. Margaria. 2009. Dynamic testing via automata learning. *STTT* 11, 4 (2009), 307–324.

[26] M. Schuts, J. Hooman, and F.W. Vaandrager. 2016. Refactoring of Legacy Software using Model Learning and Equivalence Checking: an Industrial Experience Report. In *Proceedings 12th International Conference on integrated Formal Methods (iFM),* Reykjavik, Iceland, June 1-3 *(LNCS)*, E. Ábrahám and M. Huisman (Eds.), Vol. 9681. 311–325.

[27] S. Verwer, M. de Weerdt, and C. Witteveen. 2011. The efficiency of identifying timed automata and the power of clocks. *Inf. Comput.* 209, 3 (2011), 606–625.

[28] S. Verwer, M. de Weerdt, and C. Witteveen. 2012. Efficiently identifying deterministic real-time automata from labeled data. *Machine Learning* 86, 3 (2012), 295–333.

[29] M. Volpato and J. Tretmans. 2014. Active Learning of Nondeterministic Systems from an ioco Perspective. in *ISOLA I, LNCS* 8802 (2014), 220–235.