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, 27]. The modeling frameworks of [4, 27] 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, 27] 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. In a membership query, the learner supplies a sequence of

![Figure 1. MMT model for alternating-bit protocol sender](image-url)
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.

Summarizing, our paper contributes:
- 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 [26, 27]. 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 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 for many network protocols, extensions of Mealy machines, but still expressive enough for many network protocols.

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 $\mathcal{T}(X)$ be the set of *timeout events* of the form $\mathcal{T}(x)$ for $x \in X$. For a set $I$, let $\mathcal{I}$ be $I \cup \mathcal{T}(X)$. We view (partial) functions as sets of pairs. We write $A \leadsto B$ for the set of partial functions from $A$ to $B$, and $f \downharpoonright A$ for the restriction of function $f$ to $\text{dom}(f) \cap A$. We write $\mathcal{P}_\text{fin}(A)$ for the set of finite subsets of set $A$.

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

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

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 $\text{ran}(\rho) \subseteq X(q) \cup \mathbb{N}^{>0}$. When a timer expires it
We associate a timed word 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 \( \kappa \) 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 \( \kappa, \kappa' \) valuations and \( d \in \mathbb{R}^{\geq 0} \), we define a delay transition relation by: \( \kappa \xrightarrow{d} \kappa' \) iff

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

If the current valuation is \( \kappa \), then timeout event \( to[x] \) 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 \( \mathbb{N}^{\geq 0} \) in \( \mathbb{R}^{\geq 0} \). Then, for \( \kappa, \kappa' \) valuations, \( i \in I, o \in O \) and \( \rho \in X \xrightarrow{=} (X \cup \mathbb{N}^{\geq 0}) \), we define a discrete transition relation by: \( \kappa \xrightarrow{i/o/\rho} \kappa' \) iff

\[
\text{dom}(\rho) = \text{dom}(\kappa') \land \text{ran}(\rho) \subseteq \text{dom}(\kappa) \land \forall x \in X : i = to[x] \Rightarrow (\kappa(x) = 0 \land x \notin \text{ran}(\rho)).
\]

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

\[
(q, \kappa) \xrightarrow{d} (q', \kappa') \quad \text{and} \quad q \xrightarrow{i/o/\rho} q' \quad \text{and} \quad (q, \kappa) \xrightarrow{i/o/\rho} (q', \kappa')
\]

A timed run of \( M \) over \( w \) is a sequence

\[
\alpha = C_0 \xrightarrow{d_1} C_0' \xrightarrow{i_1/o_1/\rho_1} C_1 \xrightarrow{d_2} \cdots \xrightarrow{d_k} C_k \xrightarrow{i_k/o_k/\rho_k} C_k'
\]

of transitions between configurations \( C_i, C_i' \) 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 \( d_j \in \mathbb{R}^{\geq 0}, i_j \in I \cup \{to\}, \) 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 \( j, i_j' = i_j \) if \( i_j \in I \), and \( i_j' = to \) if \( i_j \in TO[X] \). 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 \( N \) with the same sets of inputs are timed equivalent, denoted \( M \approx_{\text{timed}} N \), iff they have the same sets of timed words.

**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 \( u = d_1 i_1 \cdots d_l i_l k_{l+1} \), where \( d_j \in \mathbb{R}^{\geq 0} \) and \( i_j \in I \), for all \( 1 \leq j \leq k \), and \( k_{l+1} \in \mathbb{R}^{\geq 0} \). We may associate a timed input word \( \text{tim}(w) \) 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, \( \text{tim}(7 i o i 1 o 1 to o') = 7 i 1 i 1 i 1 2 i 2 o 1 o 1 i 4 i 4 i 4 i 0 \). 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 \( M \) such that \( \text{tim}(w) \propto u \). We call \( w \) an outcome of running experiment \( u \) on \( M \). For instance, if we perform the experiment \( 1 \) or \( 7 \) on the MMT of Figure 1, then the unique outcome is \( 1 \) in \( send_0 \) to \( send_0 \) to \( send_0 \).

**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 3 (top), for instance, experiment \( 1i01i0 to o' \) has outcomes \( 1i01i0 to o' \) and \( 1i01i0 to o'' \).

The nondeterminism of the MMT of Figure 3 (top) is “uncontrollable” in the sense that it occurs irrespective of the timing of the inputs. Figure 3 (bottom) gives an example of an MMT that exhibits nondeterminism when the second input occurs exactly one time unit after the first input: experiment \( 1i1i1 \) has outcomes \( 1i01i01i01 to o' \) and \( 1i0i1i01 to o'' \). 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 [28], 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 \( (a) \rho \) is injective, and \( (b) \) at most one timer is
We now propose an instance of Angluin’s MAT framework for what the output is in response to a timed input word $z$.

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 if $\text{tvw}(w)$ 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 4, 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.

**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 $\text{tvw}(w) \preceq u$.

**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 3.9 asserts that such a timed word always exists when $H \not\approx_{\text{timed}} M$.)

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 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 $x$: $\rho(x) \in \mathbb{N}^+$ or $\rho(x) = x$.

In fact, since an update starts at most one timer, a run 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 $\text{untime}$ that forgets all timing information and keeps the transitions of the MMT, and an abstraction $\text{beh}$ 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, $\text{beh}(\text{untime}(x)) = \text{untime}(\text{beh}(x))$, 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

$$\beta = X_0 \xi_{ij/o_i/p_i} X_1 \xi_{jk/o_j/p_k} \cdots$$

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

$$\gamma = q_0 \xi_{ij/o_i/p_i} q_1 \xi_{jk/o_j/p_k} \cdots$$

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:

$$\text{beh}(\gamma) = X(q_0) \xi_{ij/o_i/p_i} X(q_1) \xi_{jk/o_j/p_k} \cdots$$

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

$$\sigma = k_0 \xi_{d_1/o_1/p_1} k_1 \xi_{d_2/o_2/p_2} \cdots$$

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

$$\text{untime}(\sigma) = \text{dom}(k_0) \xi_{i_1/o_1/p_1} \cdots \xi_{i_k/o_k/p_k} \text{dom}(k_k).$$

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

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

$$\text{tvw}(\sigma) = d_1 i_1' o_1 a_1 o_2' \cdots d_k i_k' o_k,$$

where for all $j$, $j' = ij$ if $ij \in I$, and $j' = \text{to}$ if $ij \in \text{TO}[X]$. Let $\alpha$ be a timed run of an MMT $M$:

$$\alpha = (q_0, k_0) \xi_{i_1/o_1/p_1} (q_1, k_1) \xi_{i_2/o_2/p_2} \cdots (q_k, k_k).$$

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

$$\text{untime}(\alpha) = q_0 \xi_{i_1/o_1/p_1} q_1 \xi_{i_2/o_2/p_2} \cdots q_k.$$
We say that \( M \approx \) isomorphism \( \rho \) to \( B \) to \( \beta \) behaviors are isomorphic. The following basic property will be Lemma 3.2. untimed behavior of \( \beta \) from any feasible untimed behavior of \( N \) with \( \beta \) and \( \beta \) behaviors, which deems two untimed behaviors equivalent if there is a unique timed run \( \gamma \) with \( \beta \) (untimed semantics) \( \beta \approx \) and \( \beta \) behaviors over \( I \) with \( \beta \) and \( \beta \) is feasible.

3.2 Definition untimed semantics

We would, intuitively, like to define the untimed semantics of an MMT \( M \) 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 \( \beta \) and \( \beta' \) be two untimed behaviors with the same length and outputs:

\[
\beta = X_0 \xmapsto{i_1/o_1/p_1} X_1 \xmapsto{i_2/o_2/p_2} \ldots X_k, \\
\beta' = Y_0 \xmapsto{i'_1/o'_1/p'_1} Y_1 \xmapsto{i'_2/o'_2/p'_2} \ldots Y_k.
\]

An isomorphism from \( \beta \) to \( \beta' \) is a list \( f = f_0, \ldots, f_k \) of bijections \( f_j : X_j \to Y_j \) such that for all \( j \geq 0 \): (1) for all \( x \in X_j \setminus \text{dom}(p_j) \), \( f_j(x) \neq f_{j-1}(x) \), (2) \( i_j' = i_j \) if \( i_j \in I \) and \( i_j' = \text{to}(f_{j-1}(x)) \) if \( i_j \neq \text{to}(x) \), for some \( x \in X_{j-1} \), and (3) \( \text{dom}(p_j') = f_j(\text{dom}(p_j)) \) and \( p_j(y) = p_j(f_j^{-1}(y)) \), for all \( y \in \text{dom}(p_j') \). In this case, since \( \beta' \) is fully determined by \( \beta \) and \( f \), we write \( \beta' = f(\beta) \). We say that \( \beta \) and \( \beta' \) are isomorphic if there exists an isomorphism \( f \) from \( \beta \) to \( \beta' \).

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.

Lemma 3.1. 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 \( M \approx \) untimed \( N \), iff their sets of feasible untimed behaviors are isomorphic. The following basic property will be needed later on:

Lemma 3.2. Suppose \( M \) and \( N \) are MMTs with \( M \approx \) untimed \( N \). 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 3.3. \( M \approx \) untimed \( N \) implies \( M \approx \) timed \( N \).

The converse of Theorem 3.3 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. We say that an MMT \( M \) is timer live if, for each feasible untimed behavior \( \beta \) and for each timer \( y \) that is running after \( \beta \), there exists an untimed behavior \( \beta_y \) 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.

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 prepartory work.

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

\[
\beta_0 \xmapsto{i_0/o_0/p_0} \beta_1 \xmapsto{i_1/o_1/p_1} \beta_2 \xmapsto{i_2/o_2/p_2} \ldots
\]

that contains an \( i \)-transition that is not a timeout and does not (re)start any timer. We can then schedule this transition slightly earlier. More precisely, if \( 0 < \varepsilon < \delta \) and \( \delta' = \delta + \varepsilon - \varepsilon \) then we can find \( \kappa, \kappa' \) such that

\[
\beta_0 \xmapsto{i_0/o_0/p_0} \beta_1 \xmapsto{i_1/o_1/p_1} \beta_2 \xmapsto{i_2/o_2/p_2} \ldots
\]

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 \( p_t \) and \( p \) with \( p < q \). Then we say that \( p_t \) triggers \( p \) if there exists a timer \( x \) such that: (a) event \( p_t \) starts \( x \), (b) for all \( p < r < q \), \( x \) is unaffected by event \( i_t \), and (c) \( e_t = \text{to}(x) \). A block of \( \sigma \) is a maximal subset of indices \( B = \{ p_1, \ldots, p_n \} \) such that \( p_t \) triggers \( p_{t'} \) triggers \( p_{t''} \), etc. Note that the collection of blocks of \( \sigma \) partitions the set of indices \( \{ 1, \ldots, k \} \). We refer to this partition as \( B \). We say that timed behavior \( \sigma \) contains a race if there is some index \( j > 0 \) and some timer \( x \) such that \( \kappa_j'(x) = 0 \) and \( i_j \neq \text{to}(x) \). 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 3.4. Suppose \( \lambda \) is a timed behavior as in equation (1) without races. Suppose \( \lambda = \{ p_1, \ldots, p_n \} \) is a block of \( \lambda \), and suppose that \( e \) is a real number whose absolute is smaller than any nonzero number that occurs in \( \sigma \), that is \( |e| < \min(\bigcup_{1 \leq j \leq k} \{ d_j \} \cup \text{ran}(\kappa'_j) \} \setminus \{ 0 \}) \).

Then there exists a timed behavior \( \sigma' = \lambda_0 \xmapsto{e_0} \lambda_1 \xmapsto{e_1} \ldots \xmapsto{e_k} \lambda_{k+1} \), without races such that \( \text{untimed}(\sigma') = \text{untimed}(\sigma) \), \( \kappa_0 = \lambda_0 \), and \( e_j = d_j + e \) if \( j - 1 \notin T \) and \( j \notin T \), \( e_j = d_j - e \) if \( j - 1 \in T \) and \( j \notin T \).

In the presence of races, Lemma 3.4 does not hold. Consider the timed behavior with blocks \( \{ 1, 3 \}, \{ 2 \}, \) and \( \{ 4, 5 \} \):

\[
\lambda_0 \xmapsto{1/\delta_0/\rho \rightarrow \sigma=1} \lambda_1 \xmapsto{1/\delta_2/\rho \rightarrow \sigma=1} \lambda_2 \xmapsto{1/\delta_3/\rho \rightarrow \sigma=1} \lambda_3 \xmapsto{1/\delta_4/\rho \rightarrow \sigma=1}
\]
(u = z = 1) \rightarrow (u = z = 0) \xrightarrow{\text{tw}(z) / \rho} (u = 0).

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 \(t_i\) occurs just a bit later then timer \(y\) must expire before timer \(x\), and if \(t'_i\) 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 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 \([1, 3]\).

**Lemma 3.5.** Let \(\sigma\) be a timed behavior. Then there is a timed behavior \(\sigma'\) without races s.t. \(\text{untimed}(\sigma) = \text{untimed}(\sigma')\).

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 \(\text{tw}(\sigma)\) 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 3.6.** For each feasible untimed behavior \(\beta\) there exists a transparent timed behavior \(\sigma\) such that \(\beta = \text{untimed}(\sigma)\).

The following fundamental lemma now follows:

**Lemma 3.7.** Suppose \(\beta\) is a feasible untimed behavior that ends with timer set \(Y\), and \(Y \xrightarrow{\rho} Y'\) is an untimed behavior with \(i \in I\).

Then \(\beta \xrightarrow{\rho} Y'\) is a feasible untimed behavior.

Consider a timed word \(w = d_1 i_1 o_1 d_2 i_2 o_2 \ldots d_k i_k o_k\). We want to know, for each timeout event in \(w\), by which event this timeout is triggered. Let \(T = \{j \mid i_j = \text{tol}\}\) be the set of indices corresponding to a timeout. A causality map for \(w\) is a function \(c : T \rightarrow \{1, \ldots, k\}\) 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 \(j, \sum_{i=j+1}^{k} d_i\) is an integer (timers expire after an integer delay). It is easy to see that a timed word has a causality map iff it is a timed word of an MMT. In general, a timed word may have multiple causality maps. However, a transparent timed word 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.

We are now prepared to state our main result:

**Theorem 3.8.** Suppose that \(M\) and \(N\) are timer live MMTs. Then \(M \approx_{\text{timed}} N\) implies \(M \approx_{\text{untimed}} N\).

**Lemma 3.9.** Suppose \(M, N\) are timer live MMTs with \(M \approx_{\text{timed}} N\). 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 “canonical” form, so that a learner can only observe these behaviors up to isomorphism. Let us formalize this idea. Suppose \(\beta\) is 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 \(\text{can}(\beta)\) to denote this \(\beta'\).

Consider an untimed behavior \(\beta\) in canonical form:

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

Let \(1 \leq l \leq j \leq k\), \(d \in \mathbb{N}^+\) and suppose that \((1)\) \(\rho_1\) either starts no timer or sets \(x_j\) to \(d\), \((2)\) \(\beta\) does not contain a timeout event \(\text{tol}(x_j)\). Then addtimer\((l, d, j, \beta)\) is the untimed behavior obtained from \(\beta\) by replacing \(\rho_1\) by the update \(x_l := d\), and adding \(x_j\) to the sets \(X_l\) up to \(l\) (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 MMT \(M\) iff \(M\) has 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 \(j, l, i_j = \text{tol}(x_j)\) 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 \(\text{uiw}(\beta)\) to \(\beta\) by removing all the output events, updates, and timer sets from \(\beta\).

Our untimed learning setup is similar to the timed setup: again the teacher knows an MMT \(M\) and the learner initially only knows the set of inputs \(I\) of \(M\). 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 \(M\) (in canonical form) such that \(\text{uiw}(\beta)\) is a prefix of \(u\). With an equivalence query, the learner asks if a hypothesis \(H\) with inputs \(I\) is correct. Upon receiving \(H\), the teacher answers yes if \(\text{can}(\beta) \equiv_{\text{untimed}} H\). 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 \( M \) but not of \( \mathcal{H} \) (by Lemma 3.2 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:

\[
\beta = \emptyset \xrightarrow{d_1/m/p_1} X_1 \cdots X_{k-1} \xrightarrow{i_1/o_1/p_2} X_k.
\]

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

\[
k_0 \xrightarrow{d_1} k'_{0'} \xrightarrow{i_1/o_1/p_2} \cdots \xrightarrow{d_k} k_{k'} \xrightarrow{i_k/o_k/p_k} k_{k+1}
\]

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 constraints(\( \beta \)), consisting of:

- for each index \( j \) with \( 1 \leq j \leq k \): \( 0 < t_{j+1} - t_j \),
- for each timeout event \( t_j = \text{to}(x_j); t_j - t_l = \rho(j,x_l) \),
- 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 \( x_j \in X_l \).

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

From the saturated version of constraints(\( \beta \)), we also derive a constraint, denoted Zone(\( \beta \)), which characterizes the possible timer valuations \( k_0 \) in a timed behavior of the above form. The constraint Zone(\( \beta \)) contains for each pair of timers \( x_i, x_j \) in \( X_k \) the conjunct

\[
(m + \rho_j(x_j) - \rho_i(x_i)) < k_j(x_j) - k_i(x_i) < (n + \rho_j(x_j) - \rho_i(x_i))
\]

whenever the saturated version of constraints(\( \beta \)) contains the conjunct \( m < t_j - t_i < n \). This can be derived using \( k_0(x_j) = \rho_j(x_j) - (t_k - t_j) \) and \( k_0(x_i) = \rho_i(x_i) - (t_k - t_i) \).

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 Zone(\( \beta \)) in which \( x \) is minimal.

**Lemma 4.1.** Suppose \( \beta \) is a feasible untimed behavior with Last(\( \beta \)) = \( Y \) and \( Y \xrightarrow{\text{to}(x)}/o/p \xrightarrow{\text{to}(x)}/o/p \xrightarrow{} Y' \) is an untimed behavior. Then \( x \) is expirable after \( \beta \) iff \( \beta \xrightarrow{\text{to}(x)}/o/p \xrightarrow{} Y' \) 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 \( M \). 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 \( M \) in canonical form with \( \text{uiw}(\beta) \) 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 \( i_j \in I \). Let \( N := B i_j/\omega/p_0 \emptyset \), where \( \omega \in O \) is an arbitrary output that acts as placeholder and \( p_0 \) is the empty update. (We omit the arrow for \( i_j/\omega/p_0 \emptyset \) here.) Since \( B \) is feasible (by the loop invariant), it follows by Lemma 3.7 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 \( w = d_1 i_1 o_1 d_2 i_2 o_2 \cdots d_j i_j o_j \) is a transparent timed word. Let \( u' = \text{tiw}(w) \) 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 \( B := B j/o_j/p_0 \emptyset \), 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 \( B := \text{addtimer}(l, e, m - 1, B) \) and finish the body of the loop.

2. Case \( i_j = \text{to}(x) \), for \( i < j \). If timer \( x \) is started in \( B \) and initialized with the value \( e \) then let \( N := \text{addtimer}(l, e, j - 1, B) \xrightarrow{\text{to}(x)}/o/p_0 \emptyset \). Check whether constraints(\( N \)) is satisfiable. If so proceed to (*), otherwise exit the while loop. If timer \( x \) is not started in \( B \), let \( N^e := \text{addtimer}(l, e, j - 1, B) \xrightarrow{\text{to}(x)}/o/p_0 \emptyset \), 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 \( w = d_1 i_1 o_1 d_2 i_2 o_2 \cdots d_j i_j o_j \) is a transparent timed word. Let \( u' = \text{tiw}(w) \) be the corresponding transparent timed input word. Forward membership query \( u' \) to the timed teacher, and let \( w' \) be the response. Since \( w' \) 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 \( B = B j/o_j/p_0 \emptyset \), increment counter \( j \), and finish the body of the 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 \( B := \text{addtimer}(l, e', j - 1, B) \xrightarrow{\text{to}(x)}/o_j/p_0 \emptyset \), where \( o_j \) is the last output in \( w' \), increment counter \( j \) and finish
After termination of the while loop, the algorithm returns the feasible untimed behavior. 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 \( \mathcal{H} \equiv_{\text{timed}} \mathcal{M} \). In this case, by Theorem 3.8, \( \mathcal{H} \equiv_{\text{untimed}} \mathcal{M} \), 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 3.3, we may conclude that \( \mathcal{H} \not\equiv_{\text{untimed}} \mathcal{M} \), and thus the untimed teacher should also return a result no. Since \( w \) is a transparent timed word of \( \mathcal{M} \), it follows 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 \( \mathcal{M} \) but not of \( \mathcal{H} \); the causality map tells us exactly which timers are set and timeout. Thus the untimed teacher may return \( \beta \) as counterexample. If our estimate \( d_{\text{max}} \) 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 \( d_{\text{max}} \). 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 3.3 and 3.8). 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:** \( \beta \in S \implies \text{Head}(\beta) = \emptyset \),
- **prefix closed:** \( \beta \cdot \gamma \in S \implies \beta \in S \),
- **behavior deterministic:** \( \beta \vdash [o_1/p_1] \sigma, X_1 \in S \land \beta \vdash [o_2/p_2] \sigma, X_2 \in S \implies o_1 = o_2 \land p_1 = p_2 \land X_1 = X_2 \),
- **input complete:** \( \beta \in S \land i \in I \implies \exists \alpha, \rho, Y : \beta \vdash [i/o/p] \alpha, \rho, Y \rightarrow Y \in S \),
- **timeout complete:** \( \beta \in S \land x \text{ expirable after } \beta \implies \exists \alpha, \rho, Y : \beta \vdash [\text{to}(x)/o/p] \alpha, \rho, Y \rightarrow Y \in S \).

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 \( i_1/o_1/p_1 \cdots i_n/o_n/p_n \) of its labels. If some \( p_j \) 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 \( \text{mem}_S(\beta) \) be the set of timers \( x_1 \) in \( x_1 \cdots x_1(\beta) \) whose corresponding timeout event (of form \( \text{to}(x_1) \)) occurs (as an input) in some continuation \( \beta' \) of \( \beta \) in \( S \). Let \( \text{val}_S, \beta(x_1) \) map each timer \( x_1 \) in \( \text{mem}_S(\beta) \) 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 \( \text{mem}_S(\beta) \) be the set of timers after \( \beta \), and assigning each occurring timer to \( \text{val}_S, \beta(x_1) \) in the 1st 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 \( \beta \) and \( \beta' \) are equivalent if roughly “\( \beta \cdot \gamma \in S \iff \beta' \cdot \gamma \in S' \)” But, however, we must be careful with the names of timers that expire and/or are assigned in \( \gamma \). 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 \( \text{TO} \{ Y \} \) be the set of timeout events of form \( \text{to}(y_1) \) for \( y_1 \in Y \), and let \( I' \) be \( I \cup \text{TO}(X) \cup \text{TO}(Y) \). Define a lean suffix behavior (lean suffix for short) to be a sequence \( i_1/o_1/p_1 \cdots i_m/o_m/p_m \) of input/output/assignment triples, in which each \( i_j \) is in \( I' \), each \( p_j \) may assign only to the timer \( y_1 \), each timeout event occurs at most once, and all timers that are assigned in \( \gamma \) expire in some transition after their assignment.

For integer \( k \geq 0 \), let \( g_{a,k} \) be the injective mapping on \( Y \) which maps each \( y_1 \) to \( y_{j+k} \). We apply mappings of form \( g_{a,k} \) to lean suffix behaviors in the natural way.

We can now define residual languages. For a (lean) timer language \( S \) and lean behavior \( \beta \in S \), let \( \beta^{-1}S \) be the set of lean suffixes \( \gamma \) such that there is a canonical behavior \( \beta' \) with \( \beta' \subseteq \beta' \) such that \( \beta' \cdot g_{a,k}(\beta(y)) \in S \). Then \( \text{mem}_S(\beta) \) is the set of timers \( x_1 \) in \( x_1 \cdots x_1(\beta) \) whose corresponding timeout event (of form \( \text{to}(x_1) \)) occurs (as an input) in some suffix \( \gamma \) in \( \beta^{-1}S \), and \( \text{val}_S, \beta(x_1) \) maps each timer \( x_1 \) in \( \text{mem}_S(\beta) \) to the unique positive integer to which it is assigned in the corresponding lean behavior \( \beta' \cdot g_{a,k}(\beta(y)) \) 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 \( f : \text{mem}_S(\beta) \rightarrow \text{mem}_S(\beta') \) be a bijection from \( \text{mem}_S(\beta) \) to \( \text{mem}_S(\beta') \). Then \( \beta \) and \( \beta' \) are **equivalent** under \( f \), written \( \beta \equiv_{f} \beta' \) iff

\[
\gamma \in \beta^{-1}S \quad \text{iff} \quad f(\gamma) \in \beta'^{-1}S
\]

Intuitively, \( \beta \equiv_{f} \beta' \) means that \( \beta \) and \( \beta' \) allow the same suffixes, after renaming timers assigned in \( \beta \) by \( f \). We write \( \beta \equiv_{S} \beta' \) to denote that \( \beta \equiv_{f} \beta' \) for some \( f : \text{mem}_S(\beta) \rightarrow \text{mem}_S(\beta') \).

**Example** Let \( S \) contain the lean behaviors

\[
i_1/o_1/x_1 := 5 \cdot \text{to}(x_1)/o_3 \land i_1/o_1/x_2 := 4 \cdot \text{to}(x_2)/o_3
\]

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 \( y_1 = \text{to}(x_1)/o_3 \) and \( \beta_2^{-1}S \) contains the suffix \( y_2 = \text{to}(x_2)/o_3 \). It is now possible that \( \beta_1 \equiv_{S} \beta_2 \), where \( f \) maps \( x_1 \) to \( x_2 \) (whether \( \beta_1 \equiv_{S} \beta_2 \) 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 $y$ and $f(y)$ with $u\text{w}(y)$ and $u\text{w}(f(y))$ in $V$. Already here, we see that it is convenient to let $V$ be closed under permutations on timers in $X$, so that $u\text{w}(y) \in V$ iff $u\text{w}(f(y)) \in V$. Let us call such a set adequate. For an adequate set $V$ of input suffixes, and a lean behavior $\beta$, let $(\beta^{-1}_{\chi})|_{V}$ be the set of suffixes $y$ with $u\text{w}(y) \in V$. Let $mem_{S,V}(\beta)$ be the set of timers $x_j$ in $x_1, \ldots, x_{|\beta|}$ whose corresponding timeout event (of form to$(x_j)$) occurs (as an input) in some suffix in $(\beta^{-1}_{\chi})|_{V}$. Let $val_{S,V,\beta}$ map each timer $x_j$ in $mem_{S,V}(\beta)$ to the unique positive integer to which it is assigned in the $i$th transition of $\beta$.

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 $\beta$ and $\beta'$ 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 $\beta$ and $\beta'$ are equivalent wrt $V$ under $f$, written $\beta \equiv^f_{S,V} \beta'$ iff

$$y \in (\beta^{-1}_{\chi})|_{V} \iff f(y) \in (\beta'^{-1}_{\chi})|_{V}$$

Intuitively, $\beta \equiv^f_{S,V} \beta'$ means that $\beta$ and $\beta'$ allow the same suffixes with inputs in $V$, after renaming timers assigned in $\beta$ by $f$. We write $\beta \equiv_{S,V} \beta'$ to denote that $\beta \equiv^f_{S,V} \beta'$ 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 the 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 $\gamma \in \beta^{-1}_{\chi}$, let $\beta_S$ be the (unique) lean behavior $\beta' = g_{\beta}((\gamma))$ in $S$ with $\beta \subseteq \beta'$. Let $ens_{\beta}(\beta)$ be the set of $\gamma \in U$ such that $\beta_S / \gamma \in S$ for some $\gamma$ (recall that the last assignment of a lean behavior is always empty). For $i \in ens_{\beta}(\beta)$, let $\lambda(\beta, i)$ be the unique output $o$ such that $\beta_S / o \in S$. 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 $i \in ens_{\beta}(\beta)$ there is a $\beta' \in \beta / i$ such that $\beta_S / \lambda(\beta, i) \equiv_{S,V} \beta'$.
- $U$ is timer-consistent wrt. $V$ if for each $\beta \in U$ and $i \in ens_{\beta}(\beta)$ we have $mem_{S,V}(\beta_S / \lambda(\beta, i)) \subseteq (mem_{S,V}(\beta) \cup x_{i|\beta+1|})$.

Closure 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 $\beta_S / \lambda(\beta, i)$) is either a timer active after $\beta$ is started by the last transition, thus getting the name $x_{i|\beta+1|}$. Closure 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 $\mathcal{H}(U, V) = (\chi, O, Q, q_0, X, \delta, \lambda, \pi, \pi')$, where

- $Q = U$ and $q_0 = \epsilon$,
- $X$ maps each location $\beta \in U$ to $mem_{S,V}(\beta)$,
- $\lambda$ is the unique such that $\beta_S / o \in S$.
- $\delta(\beta, i)$ is the unique $\beta' \in U$ such that there is an $f$ with $\beta_S / i \lambda(\beta, i) \equiv_{S,V} \beta'$.
- $\pi(\beta, i) : mem_{S,V}(\beta') \rightarrow (mem_{S,V}(\beta') \cup \mathbb{N}^+)$ is defined as $f^{-1}$ on $mem_{S,V}(\beta')$, except that it maps $f(x_{i|\beta+1|})$ to $val_{S,V,\beta}(f(x_{i|\beta+1|}))$ if $f(x_{i|\beta+1|}) \in mem_{S,V}(\beta')$.
- When $\beta \in U$ and $i$ is of form $to(x_j)$ with $x_j mem_{S,V}(\beta)$ but $to(x_j) \notin ens_{\beta}(\beta)$, we let $\lambda(\beta, i) = \epsilon$, $\delta(\beta, i) = \beta$, and let $\pi(\beta, i)$ be the identity mapping on $mem_{S,V}(\beta)$.

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

Hypothesis construction. performs membership queries in order to construct the sets of form $(\beta^{-1}_{\chi})|_{V}$ and $(\beta_S / \lambda(\beta, i)^{-1}_{\chi})|_{V}$ for $\beta \in U$ and $i \in ens_{\beta}(\beta)$. 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 $u\text{w}(\beta) : \beta \in U$ and $i \in I$: this allows to determine $ens_{\beta}(\beta)$ and $\lambda(\beta, i)$ for $\beta \in U$ and $i \in ens_{\beta}(\beta)$. Thereafter, membership queries are performed for all untimed input words of form $u\text{w}(\beta) : \beta \in U$, $\beta \in V$, and $i \in ens_{\beta}(\beta)$. Note that one need only consider $\gamma$ in which timeouts from $\mathcal{H}(X)$ concern timers in $\{x_1, \ldots, x_{|\beta|}\}$ (or in $\{x_1, \ldots, x_{|\beta+1|}\}$ for $u\text{w}(\beta) : \beta$). This allows to construct the sets of form $(\beta^{-1}_{\chi})|_{V}$ and $(\beta_S / \lambda(\beta, i)^{-1}_{\chi})|_{V}$ for $\beta \in U$. It then allows to compute the approximated Nerode equivalence $\equiv_S$ on the set of lean behaviors of form $\beta$ and $\beta_S / \lambda(\beta, i)$ with $\beta \in U$ and $i \in ens_{\beta}(\beta)$. 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 $i \in ens_{\beta}(\beta)$ for which there is no
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 algorithm is completed, and $\mathcal{H}(U, V)$ accepts $S$. If the query returns a counterexample in the form of a behavior $\sigma$ 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 $\nu$ such that $U$ 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 $S$, 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 $n \cdot (|V| + r + 1) \cdot |V| \cdot r!$ 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 $\beta \cdot i/o \equiv_{S, V} \beta'$, 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.

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.

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]. A challenge is also to deal with timing uncertainties due to nondeterminism and imprecise measurements.

**References**

A Appendix

The appendix contains proofs of technical results in the paper. The proofs are the following:

- Proof of Theorem 3.3, preceded by a small technical lemma.
- Proof of Lemma 3.5
- Proof of Lemma 3.6
- Proof of Lemma 3.7
- Some technical definitions and lemma’s about causality maps used in the proof of Theorem 3.8
- Proof of Theorem 3.8
- Two technical lemmas, Lemma A.6 and A.7, about zones used in the proof of Theorem 5.3
- Proof of Theorem 5.3
- The procedure for processing counterexamples in the learning algorithm

Lemma A.1. Let σ be a timed behavior and let f be an isomorphism for σ. Then untimed (f(σ)) = (f(untimed(σ)).

Theorem 3.3. M ≈untimed N implies M ≈timed N.

Proof. Assume M ≈untimed N and w is a timed word of M. Since ≈untimed 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 α of M with tw(α) = w. Let σ = beh(α) and β = untimed(σ). Then β is a feasible untimed behavior of M and tw(β) = w. Since M ≈untimed N, there exists an isomorphism f such that f(β) is a feasible untimed behavior of N. Hence N has an untimed run y’ such that untimed(y’) = β’. Let σ’ = f(σ). By Lemma A.1, σ’ is a timed behavior with untimed(σ’) = untimed(f(σ)) = f(untimed(σ)) = f(β) = β’. Since beh(β’) = untimed(σ’) = β’, N has a timed run α’ = pullback(y’, σ’) with beh(α’) = σ’. Note that tw(α’) = tw(σ’) = tw(f(σ)) = tw(σ) = w. Hence w is a timed word of N, as required.

Lemma 3.5. Let σ be any timed behavior. Then there exists a timed behavior σ’ without races such that untimed(σ) = untimed(σ’).

Proof. (Sketch) Let

\[\sigma = k_0 \xrightarrow{d_i} k_0 \xrightarrow{i_1/o_1/p_1} k_1 \xrightarrow{d_2} \ldots \xrightarrow{k_j-i_1/o_1/p_k} k_k\]

be a timed behavior. For each index j, each timer in the domain of k_j is started by some preceding event. Let startedby_j : dom(k_j) → Π_o be the function that maps each timer in the domain of k_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 k’_{j-1}(x) = 0 and i_j ≠ to[x]. Let B ∈ Π_o 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_o. 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.

Lemma 3.6. For each feasible untimed behavior β there exists a transparent timed behavior σ such that β = untimed(σ).

Proof. Let β be a feasible untimed behavior. Then there exists a timed behavior σ such that β = untimed(σ). By Lemma 3.5, we may assume that σ contains no races. By repeated application of Lemma 3.4, we can wiggle the timing of all the blocks to make σ transparent.

Lemma 3.7. Suppose β is a feasible untimed behavior that ends with timer set Y, and Y \xrightarrow{i_1/o_1/p_1} Y’ is an untimed behavior with i ∈ I. Then β \xrightarrow{i_1/o_1/p_1} Y’ is a feasible untimed behavior.

Proof. By Lemma 3.6, there exists a transparent timed behavior σ such that β = untimed(σ). 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_1/o_1/p_1} Y’. This implies that untimed behavior β \xrightarrow{i_1/o_1/p_1} Y’ is feasible.

Causality maps. Consider an untimed behavior

\[\mathbf{β} : \mathbf{X}_{1} \xrightarrow{i_1/o_1/p_1} \mathbf{X}_{1} \xrightarrow{i_2/o_2/p_2} \ldots \xrightarrow{i_k/o_k/p_k} \mathbf{X}_{k}\]

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

Lemma A.2. 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 untimed(σ), and we say that it is a causality map of a timed run α if it is a causality map of beh(α). Lemma A.2 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 | i_j = to\} be the set of indices corresponding to a timeout. A causality map for w is a function c : T → {1, ..., 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=1}^{c(j)} d_l is an integer (timers expire after an integer delay).

Lemma A.3. 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(α).

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 l is different.

Lemma A.4. Suppose α is a timed run of an MMT, and w = tw(α) 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 A.5. Suppose $w$ is a timed word of MMT $M$ with a unique causality map. Then there is a unique timed run $α$ of $M$ such that $w = tw(α)$.

Theorem 3.8. Suppose that $M$ and $N$ are timer live MMTs. Then $M \simeq_{\text{timed}} N$ implies $M \simeq_{\text{untimed}} N$.

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

Since $β$ is a feasible untimed behavior of $M$, $M$ has an untimed run $γ$ with $\text{beh}(γ) = β$. By Lemma 3.6, there exists a transparent-timed behavior $σ$ such that $β = \text{untimed}(α, σ)$. There exists a unique timed run $α = \text{pullback}(γ, σ)$ of $M$ with $\text{untimed}(α) = γ$ and $\text{beh}(α) = σ$. Thus $σ$ is a timed behavior of $M$. Let $w = tw(α)$.

Then $w$ is a timed word of $M$. By Lemma A.4, $w$ has a unique causality map $c$. Since $M \simeq_{\text{timed}} N$, $w$ is also a timed word of $N$.

By Lemma A.5, $N$ has a unique timed run $α'$ such that $w = tw(α')$. Let $β' = \text{untimed}(\text{beh}(α'))$. Then $β'$ is a feasible untimed behavior of $N$. Note that the mappings $tw$, $\text{untimed}$ and $\text{beh}$ 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 A.2 and A.3, $β'$ 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 $M \simeq_{\text{untimed}} M$, 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 $θ$, and thus trivially isomorphic.

For the induction step, suppose $β$ and $β'$ contain $k + 1$ events: $β = X_0 \xrightarrow{\alpha_1/p_1} X_1 \cdots \xrightarrow{\alpha_k/p_k} X_k \xrightarrow{\alpha_{k+1}/p_{k+1}} X_{k+1}$, $β' = Y_0 \xrightarrow{\alpha_1/τ_1} Y_1 \cdots \xrightarrow{\alpha_k/τ_k} Y_k \xrightarrow{\alpha_{k+1}/τ_{k+1}} Y_{k+1}$.

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 $δ' = f(δ)$. Our task is to extend this isomorphism to $β$ and $β'$.

Since $tw$, $\text{untimed}$ and $\text{beh}$ preserve inputs except for the timers in timeouts, either $i_{k+1} = 0$ or $i_{k+1}$ is active in $T_0(X)$. If $i_{k+1} = 0$, 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 $i'_{k+1} = to(f_j(x))$.

Now suppose that $x \in X_{k+1}$ is a timer that is active after $β$. Then, since $M$ is timer live, there exists an untimed behavior $β_x$ consisting of transitions that leave $x$ unaffected, except for the last one in which the time expires, and such that $β_x \cdot β_x$ is a feasible untimed behavior of $M$. Using the same construction as in the beginning of this proof, we may construct an untimed behavior $β'_x$ such that $β' \cdot β'_x$ is a feasible untimed behavior of $N$ such that $β \cdot β_x$ and $β' \cdot β'_x$ 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 $β' \cdot β'_x$. Then $m$ is in the domain of $c_x$ and $c_x(m) ≤ k + 1$. If $c_x(m) ≤ k$ 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 $f_{k+1} : X_{k+1} \rightarrow Y_{k+1}$ 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 $N$ 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 $β' \cdot β_y$ is a feasible untimed behavior of $N$. Using again the same construction as in the beginning of this proof, we may establish that $X_{k+1}$ contains a corresponding timer.

The following two lemmas will be used Theorem 5.3.

Lemma A.6. Suppose $β'$ are untimed behaviors such that $\text{Zone}(β) = \text{Zone}(β')$. Let $γ$ be any untimed behavior. Then $β \cdot γ$ is feasible iff $β' \cdot γ$ is feasible.

Lemma A.7. $\{\text{Zone}(β) \mid β$ feasible untimed behavior of $M\}$ is finite.

Proof. An MMT only has a finite number of timers that can only be set to a finite number of integer values. Since in an MMT the values of timers can only decrease, their values are bounded, so that only finitely many integers may appear in the conjuncts of $\text{Zone}(β)$. This the set of possible $\text{Zone}(β)$ is finite.

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

Proof. Let us first introduce some notation. For a lean behavior $β \in S$ and $γ \in β^{-1}S$, let $β \cdot γ$ be the (unique) lean behavior $β' \cdot γ$ in $S$ with $β \subseteq β'$. Let $\text{ens}(β)$ be the set of $i \in 1$ such that $β \cdot γ$ for some $o$ (recall that the last assignment of a lean behavior is always empty). For $i \in \text{ens}(β)$, let $δ(β, i)$ be the unique output $o$ such that $β \cdot γ \cdot o = o$.

“⇒” Let $M$ be an MMT, let $T$ be its set of feasible untimed behaviors, and let $S = \text{lean}(\text{can}(T))$. Then it follows from the definitions, Lemma 3.7 and Lemma 4.1, that $S$ is a timer language. Suppose that $β, β' \in S$. Then there are unique canonical $β, β'$ such that $β = \text{lean}(β)$ and $β' = \text{lean}(β')$ and (unique) $δ, δ' \in T$ with $\text{can}(δ) = β$ and $\text{can}(δ') = β'$. Since $β$ and $δ$ are isomorphic, there exists an isomorphism $h = h_0, \ldots, h_k$ from $β$ to $δ$. Similarly, since $β'$ and $δ'$ are isomorphic, there exists an isomorphism $h' = h_0', \ldots, h_k'$ from $β'$ to $δ'$. Suppose $δ$ and $δ'$ lead to the same state $q$ of $M$ and moreover $\text{Zone}(h^{-1}(δ)) = \text{Zone}(h'^{-1}(δ'))$. We claim that $β \equiv_S β'$. Let $f$ be the restriction of $(h')^{-1} \circ h_k$ to $\text{Last}(β)$. Suppose that $γ \in β^{-1}S$. Then there exists an untimed behavior $ζ$ such that $δ \cdot ζ \in T$ and $\text{can}(δ \cdot ζ) = β \cdot γ$. Then $δ \cdot ζ$ is a feasible behavior in $T$, it follows by Lemma A.6 that $δ \cdot ζ$ is a feasible behavior in $T$. It follows that $\text{can}(δ \cdot ζ) = β \cdot γ$ and hence that $(γ) \in β^{-1}S$. Clause (b) of Definition 5.2 follows by a symmetric argument. Since $M$ only has finitely many states and by Lemma A.7 the number of zones of feasible untimed behaviors of $M$ is finite, it follows that $S$ has finite index.

“⇐” Suppose $S$ is a (lean) timer language such that $S$ has a finite index. A subset $Q \subseteq S$ is a basis for $S$ if it contains a unique
representative from each equivalence class of \( \equiv_S \). Since \( \equiv_S \) has finite index, \( S \) has a finite basis \( Q \). Now MMT \( M = (I, O \cup \{ \bot \}, Q, \beta_0, X, \delta, \lambda, \pi) \) is defined as follows:

- \( \beta_0 \) is the unique element of \( Q \) such that \( \beta_0 \equiv_S \epsilon \).
- \( X(\beta) = \text{mem}_S(\beta) \).
- Let \( \beta \in U \) and \( i \in \text{ens}_S(\beta) \). Then
  - \( \lambda(\beta, i) \) is the unique \( o \) such that \( \beta \cdot i/o \in S \).
  - \( \delta(\beta, i) \) is the unique \( \beta' \in U \) such that there is an \( f \) with \( \beta \cdot i/\lambda(\beta, i) \xrightarrow{f} \beta' \).
  - \( \pi(\beta, i) : \text{mem}_S(\beta') \mapsto (\text{mem}_S(\beta) \cup \mathbb{N}^{>0}) \) is defined as \( f^{-1} \) on \( \text{mem}_S(\beta') \), except that it maps \( f(x_{N}) \) to \( \text{val} \beta_i f(x_{N + 1}) \) if \( f(x_{N + 1}) \in \text{mem}_S(\beta') \).
- When \( \beta \in U \) and \( i \) is of form \( \text{to} x_j \) with \( x_j \in \text{mem}_S(\beta) \) but \( \text{to} x_j \notin \text{ens}_S(\beta) \), we let \( \lambda(\beta, i) = \bot, \delta(\beta, i) = \beta \), and let \( \pi(\beta, i) \) be the identity mapping on \( \text{mem}_S(\beta') \).

The last case (where \( \text{to} x_j \notin \text{ens}_S(\beta) \)) constructs a transition that is not feasible, but which must anyway be syntactically present, since \( x_j \) may expire after some continuation if \( \beta \) is hence live.

It is routine to verify that \( M \) is a well-defined MMT whose set of feasible untimed behaviors is isomorphic to \( S \). \( \Box \)

**Processing Counterexamples in the Learning Algorithm** This section presents the procedure for extracting a new lean suffix from a counterexample produced by an equivalence query. Suppose that an equivalence query returns a counterexample in the form of a lean behavior \( \alpha \) on which \( H(U, V) \) and \( S \) disagree, this is used to extend \( V \) 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 \( \beta \cdot i/o \xrightarrow{f} \beta' \) but \( y \in (\beta \cdot i/\lambda(\beta, i))^{-1} S \nRightarrow y \in \beta'^{-1} S \) for some \( \beta, \beta' \in U \) such that \( \beta \cdot i/\lambda(\beta, i) \xrightarrow{f} \beta' \) is used to construct the transition triggered by \( i \) from \( \beta \) in \( H(U, V) \). To see this, let \( \alpha = t_1/o_1 t_1 o_2 t_2 o_2 \cdots o_n t_n o_n / \rho_n \), and for \( j = 0, \ldots, n \), define a lean behavior \( \beta_j \) and a \( \beta_j \cdot l \)-suffix as follows.

- \( \beta_0 = \epsilon \), and \( \gamma_0 \) is obtained by making \( \alpha \) a lean suffix (i.e., renaming each timer \( x_j \) to \( y_j \)).
- Let \( y_{j-1} \) be of form \( i_j / o_j \beta_j \cdot y_{j-1} \). Let \( \beta_j \) be \( \lambda(\beta_{j-1}, i_j) \), let \( f_j \) be the mapping used to establish \( \beta_{j-1} \cdot i_j / o_j \beta_j \xrightarrow{f_j} \beta_j \) in the construction of \( H(U, V) \), and let \( y_j \) be obtained from \( y_j' \) by (i) applying \( f_j \) to timers in \( \{ x_1, \ldots, x_{\beta_j} \} \), (ii) replacing \( y_j \) by \( f_j(x_{\beta_j + 1}) \), and (ii) replacing each \( y_l \) by \( y_l \).

The result is that \( \beta_0 \cdots \beta_n \) is the sequence of states visited when \( H(U, V) \) 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 \( \gamma_0 \in \beta_0^{-1} S \Rightarrow \gamma_n \in \beta_n^{-1} S \) (since \( \gamma_n \) is the empty sequence), which implies that \( y_{j-1} \in \beta_{j-1}^{-1} S \Rightarrow y_j \in \beta_j^{-1} S \) for some \( j \); we can then take \( \beta_{j-1} \) as \( \beta \) and \( \beta_j \) as \( \beta' \), and \( y \) as \( y_j \). This means that \( y \) is a new separating suffix, and that \( V \) should be extended with \( u \omega(y) \).