Formal Methods in System Design (2021) 59:77–102
https://doi.org/10.1007/s10703-022-00396-6

Equivalence checking and intersection of deterministic timed finite state machines

Davide Bresolin[1] · **Khaled El-Fakih**[2] · **Tiziano Villa**[3] · **Nina Yevtushenko**[4]

Received: 31 October 2018 / Accepted: 17 July 2022 / Published online: 12 September 2022

Abstract

There has been a growing interest in defining models of automata enriched with time, such as finite automata extended with clocks (timed automata). In this paper, we study deterministic timed finite state machines (TFSMs), i.e., finite state machines with a single clock, timed guards and timeouts which transduce timed input words into timed output words. We solve the problem of equivalence checking by defining a bisimulation from timed FSMs to untimed ones and vice versa. Moreover, we apply these bisimulation relations to build the intersection of two timed finite state machines by untiming them, intersecting them and transforming back to the timed intersection. It is known that many problems like inclusion and equivalence checking are undecidable for timed automata. Our results show that TFSMs correspond to a decidable subclass of timed automata that admits a restricted form of ϵ-transitions (i.e., timeouts) where most of the relevant problems like equivalence and intersection are decidable.

✉ Davide Bresolin
davide.bresolin@unipd.it

Khaled El-Fakih
kelfakih@aus.edu

Tiziano Villa
tiziano.villa@univr.it

Nina Yevtushenko
evtushenko@ispras.ru

[1] Dipartimento di Matematica, University of Padova, Padova, Italy

[2] American University of Sharjah, Sharjah, United Arab Emirates

[3] Dipartimento di Informatica, University of Verona, Verona, Italy

[4] Ivannikov Institute for System Programming of the Russian Academy of Sciences & National Research University, Higher School of Economics, Moscow, Russia

1 Introduction

Finite automata (FA) and finite state machines (FSMs) are formal models widely used in the practice of engineering and science, e.g., in application domains ranging from sequential circuits, communication protocols, embedded and reactive systems, to biological modelling.

Since the 90s, the standard classes of FA have been enriched with the introduction of time constraints to represent more accurately the behaviour of systems in discrete or continuous time. Timed automata (TA) are such an example: they are finite automata augmented with a number of resettable real-time clocks, whose transitions are triggered by predicates involving clock values [3].

More recently, timed models of FSMs (TFSMs) have been proposed in the literature by the introduction of time constraints such as timed guards or timeouts. Timed guards restrict the input/output transitions to happen within given time intervals. The meaning of timeouts is the following: if no input is applied at a current state for some timeout period, the timed FSM moves from the current state to another state using a timeout function; e.g., timeouts are common in telecommunication protocols and systems.

For instance, the timed FSM proposed in [21, 22, 29] features: one clock variable, time constraints to limit the time elapsed at a state, and a clock reset when a transition is executed. Instead, the timed FSM proposed in [31, 37] features: one clock variable, time constraints to limit the time elapsed when an output has to be produced after an input has been applied to the FSM, a clock reset when an output is produced, and timeouts.

In [13] the following models of deterministic TFSMs with a single clock were investigated: TFSMs with only timed guards, TFSMs with only timeouts, and TFSMs with both timed guards and timeouts.

The problem of equivalence checking was solved for all three models, their expressive power compared, and subclasses of TFSMs with timeouts and with timed guards equivalent to each other were characterized (see Fig. 1 from [13] for a diagram showing the expressivity hierarchy of TFSMs with timed guards and timeouts, TFSMs with only timed guards, TFSMs with only timeouts, loop-free TFSMs with timeouts, TFSMs with LCRO - Left Closed Right Open - timed guards, and finally untimed FSMs). Equivalence checking was obtained by introducing relations of bisimulation that define untimed finite state machines whose states include information on the clock regions, such that the timed behaviours of two timed FSMs are equivalent if and only if the behaviours of the companion untimed FSMs are equivalent. This operation is reminiscent and stronger than the region graph construction for timed automata [3].

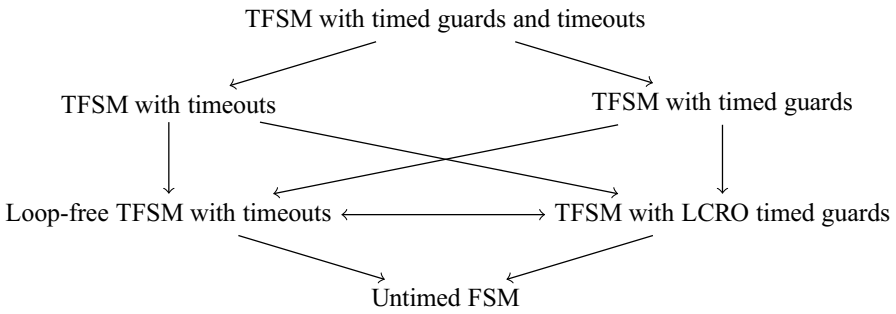


Fig. 1 Comparison of TFSM models

Here we work directly with deterministic TFSMs with both timed guards and timeouts, since they subsume the previous two models. For such TFSMs, we give the detailed construction of the untimed FSM from a timed FSM (what we get is the *FSM abstraction* of the TFSM), and then we provide the complete proof that we can describe the behavior of a TFSM using the corresponding untimed FSM, i.e., that two deterministic TFSMs are equivalent if and only if their timed-abstracted FSMs are equivalent.

Then we study the conditions under which the opposite transformation is possible: we take an untimed deterministic FSM that accepts and produces words from input and output alphabets (both including a special symbol that simulates the passing of time), and we build an equivalent deterministic TFSM with timeouts and timed guards, under the same notion of abstraction of timed words. This is the key technical result of this paper.

Finally, we apply the previous transformations to perform the intersection of two deterministic TFSMs, as an example of composition operator under which TFSMs are closed. We prove how the transformation from TFSMs to untimed FSMs of Sect. 2 and the transformation from untimed FSMs to TFSMs of Sect. 3 can be used to construct the intersection of two TFSMs.

We outline the structure of the paper. Section 2 introduces deterministic timed finite state machines with timed guards and timeouts, describes the untiming procedure to obtain a finite state machine and proves the bisimulation with the original timed one, from which an equivalence checking procedure follows. This is a revision of the material in [13], whereas the following sections are completely new. Section 3 describes the backward transformation from untimed FSMs to TFSMs and proves the backward bisimulation relation. The two results are used in Sect. 4 to compute the TFSM that is the intersection of two given deterministic TFSMs. Section 5 relates TFSMs to timed automata, and surveys expressiveness and complexity results of various models of timed automata, with final conclusions drawn in Sect. 6.

2 Models of timed FSMs (TFSMs)

Let A be a finite alphabet, and let $\mathbb{R}^+$ be the set of non-negative reals. A *timed symbol* is a pair (a, t) where $t \in \mathbb{R}^+$ is called the *timestamp* of the symbol $a \in A$. A timed word is then defined as a finite sequence $(a_1, t_1)(a_2, t_2)(a_3, t_3) \ldots$ of timed symbols where the sequence of timestamps $t_1 \leq t_2 \leq t_3 \leq \ldots$ is non decreasing. Timestamps represent the *absolute times* at which symbols are received or produced. In the following we will sometime also reason in terms of *relative times*, or *delays*, measured as the difference between the timestamps of two successive symbols. More formally, the delay of a symbol a_i is defined as $t_i - t_{i-1}$ when $i > 1$ and as t_1 when $i = 1$.

The timed models considered in this paper are initialized input/output machines that operate by reading a *timed input word* $(i_1, t_1)(i_2, t_2) \ldots (i_k, t_k)$ defined on some *input alphabet* I, and producing a corresponding *timed output word* $(o_1, t_1)\,(o_2, t_2) \ldots (o_k, t_k)$ on some *output alphabet* O. The production of outputs is assumed to be instantaneous: the timestamp of the j-th output o_j is the same of the j-th input i_j. Models where there is a delay between reading an input and producing the related output are possible but not considered here. Given a timed word $(a_1, t_1)(a_2, t_2) \ldots (a_k, t_k)$, $\text{Untime}((a_1, t_1)(a_2, t_2) \ldots (a_k, t_k)) = a_1 a_2 \ldots a_k$ denotes the word obtained when deleting the timestamps.

A timed possibly non-deterministic and partial FSM (TFSM) is an FSM augmented with a clock. The clock is a real number that measures the time delay at a state, and its

value is reset to zero when a transition is executed. In this section we introduce the TFSM model with both timed guards and timeouts defined in [13]. Such a model subsumes the TFSM model with timed guards only given in [21, 29] and the TFSM model with timeouts only given in [37, 53]. In addition, we establish a very precise connection between timed and untimed FSMs, showing that it is possible to describe the behavior of a TFSM using a standard FSM that is called the *FSM abstraction* of the TFSM.

A *timed guard* defines the time interval when a transition can be executed. Intuitively, a TFSM in the present state s accepts an input i at a time t only if t satisfies the timed guard of some transition labelled with input symbol i. The transition defines the output o to be produced and the next state s'. A *timeout* instead defines for how long the TFSM can wait for an input in the present state before spontaneously moving to another state. Each state of the machine has a timeout (possibly ∞) and all outgoing transitions of the state have timed guards with upper bounds less than the state timeout. The clock is reset to 0 every time the TFSM activates a transition or a timeout expires. Without loss of generality, we can assume that timeouts and boundaries of the timed guards are integers. If the bounds are rational numbers, they can always be transformed into integers by multiplying them by an appropriate scaling factor.

Definition 1 *(Timed FSM)* A timed FSM M is a finite state machine augmented with timed guards and timeouts. Formally, a timed FSM (TFSM) is a 6-tuple $(S, I, O, \lambda_S, s_0, \Delta_S)$ where S, I, and O are finite disjoint non-empty sets of states, inputs and outputs, respectively, s_0 is the initial state, $\lambda_S \subseteq S \times (I \times \Pi) \times O \times S$ is a transition relation where Π is the set of input timed guards, and $\Delta_S : S \to S \times (\mathbb{N} \cup \{\infty\})$ is a *timeout function* such that $\Delta_S(s)_{\downarrow\mathbb{N}} > 0$ for each $s \in S$. Each guard in Π is an interval $g = \langle t_{min}, t_{max} \rangle$ where t_{min} is a nonnegative integer, while t_{max} is either a nonnegative integer or ∞, $t_{min} \leq t_{max}$, and $\langle \in \{(, [\}$ while $\rangle \in \{),]\}$.

The *timed state* of a TFSM is a pair (s, x) such that $s \in S$ is a state of M and $x \in \mathbb{R}^+$ is the current value of the clock, with the additional constraint that $x < \Delta_S(s)_{\downarrow\mathbb{N}}$ (the value of the clock cannot exceed the timeout). If no input is applied at a current state s before the timeout $\Delta_S(s)_{\downarrow\mathbb{N}}$ expires, then the TFSM will move to anther state $\Delta_S(s)_{\downarrow S}$ as prescribed by the timeout function. If $\Delta_S(s)_{\downarrow\mathbb{N}} = \infty$, then the TFSM can stay at state s infinitely long waiting for an input. An input/output transition can be triggered only if the value of the clock is inside the guard $\langle t_{min}, t_{max} \rangle$ labeling the transition. *Transitions* between timed states can be of two types:

- *timed transitions* of the form $(s, x) \xrightarrow{t} (s', x')$ where $t \in \mathbb{R}^+$, representing the fact that a delay of t time units has elapsed without receiving any input. The relation $\xrightarrow{t}$ is the smallest relation closed under the following properties:
  - for every timed state (s, x) and delay $t \geq 0$, if $x + t < \Delta_S(s)_{\downarrow\mathbb{N}}$, then $(s, x) \xrightarrow{t} (s, x + t)$;
  - for every timed state (s, x) and delay $t \geq 0$, if $x + t = \Delta_S(s)_{\downarrow\mathbb{N}}$, then $(s, x) \xrightarrow{t} (s', 0)$ with $s' = \Delta_S(s)_{\downarrow S}$;
  - if $(s, x) \xrightarrow{t_1} (s', x')$ and $(s', x') \xrightarrow{t_2} (s'', x'')$ then $(s, x) \xrightarrow{t_1 + t_2} (s'', x'')$.
- *input/output transitions* of the form $(s, x) \xrightarrow{i,o} (s', 0)$, representing reception of the input symbol $i \in I$, production of the output $o \in O$ and reset of the clock. An input/output

transition can be activated only if there exists $(s, i, \langle t_{min}, t_{max}\rangle, o, s') \in \lambda_S$ such that $x \in \langle t_{min}, t_{max}\rangle$.

A *timed run* of a TFSM M interleaves timed transitions with input/output transitions. Given a timed input word $v = (i_1, t_1)(i_2, t_2)\dots(i_k, t_k)$, a timed run of M over v is a finite sequence $\rho = (s_0, 0) \xrightarrow{t_1} (s'_0, x_0) \xrightarrow{i_1,o_1} (s_1, 0) \xrightarrow{t_2-t_1} (s'_1, x_1) \xrightarrow{i_2,o_2} (s_2, 0) \xrightarrow{t_3-t_2} \dots \xrightarrow{i_k,o_k} (s_k, 0)$ such that s_0 is the initial state of M, and for every $j \geq 0$ $(s_j, 0) \xrightarrow{t_{j+1}-t_j} (s'_j, x_j) \xrightarrow{i_{j+1},o_{j+1}} (s_{j+1}, 0)$ is a valid sequence of transitions of M. The timed run ρ is said to *accept* the timed input word $v = (i_1, t_1)(i_2, t_2)\dots(i_k, t_k)$ and to *produce* the timed output word $u = (o_1, t_1)(o_2, t_2)\dots(o_k, t_k)$. The behavior of M is defined in terms of the input/output words accepted and produced by the machine. Notice that in our model timeouts are always greater than 0, but timed guards can be of the form [0, 0] and $[0, t_{max}\rangle$, and thus timed runs may include timed transitions with zero delay, i.e. of the form $(s, x) \xrightarrow{0} (s, x)$.

The usual definitions for FSMs of deterministic and non-deterministic, submachine, etc., can be extended to the timed FSM model considered here. In particular, a TFSM is *complete* if for each state s, input i and value of the clock x there exists at least one transition $(s, x) \xrightarrow{i,o} (s', 0)$, otherwise the machine is *partial*. A TFSM is *deterministic* if for each state s, input i and value of the clock $x < \Delta_S(s)_{\downarrow\mathbb{N}}$ there exists at most one input/output transition, otherwise is *non-deterministic*.

For the sake of simplicity, from now on we consider only *deterministic* machines (possibly partial), leaving the treatment of non-deterministic TFSMs to future work.

Definition 2 The behavior of a deterministic TFSM M is a partial mapping $B_M : (I \times \mathbb{R}^+)^* \mapsto (O \times \mathbb{R}^+)^*$ that associates every input word $w = (i_1, t_1)(i_2, t_2)\dots(i_k, t_k)$ accepted by M with the unique output word $B_M(w) = (o_1, t_1)(o_2, t_2)\dots(o_k, t_k)$ produced by M under input w, if it exists. When M is an untimed FSM the behavior is defined as a partial mapping $B_M : I^* \mapsto O^*$.

Two machines M and M' with the same input and output alphabets are *equivalent* if and only if they have same behavior, i.e, $B_M = B_{M'}$.

So for a partial and deterministic TFSM M, we have that for every input word w, $B_M(w)$ is either not defined or a singleton set. Moreover, we can consider the transition relation of the machine as a partial function $\lambda_S : S \times I \times \mathbb{R}^+ \mapsto S \times O$ that takes as input the current state s, the delay t and the input symbol i and produces the (unique) next state and output symbol $\lambda_S(s, t, i) = (s'', o)$ such that $(s, 0) \xrightarrow{t} (s', t') \xrightarrow{i,o} (s'', 0)$. With a slight abuse of the notation, we can extend it to a partial function $\lambda_S : S \times (I \times \mathbb{R}^+)^* \mapsto S \times O^*$ that takes as inputs the initial state s and a timed word w, and returns the state reached by the machine after reading w and the generated output word. We will use $s \xrightarrow{w,u} s'$ as a shorthand for $\lambda_S(s, w) = (s', u)$.

Abstracting TFSMs with timeouts and timed guards.

In this section we show how to build an abstract untimed FSM that describes the behaviour of a TFSM with guards. To do this we define an appropriate notion of abstraction of a timed word into an untimed word and a notion of bisimulation to compare a TFSM with guards with an untimed FSM. From the properties of the bisimulation relation, we conclude that the behaviour of the abstract untimed FSM is the abstraction of the behaviour of the TFSM.

For every $N \geq 0$, we define $\mathbb{I}_N$ as the set of intervals $\mathbb{I}_N = \{[n,n] \mid n \leq N\} \cup \{(n,n+1) \mid 0 \leq n < N\} \cup \{(N,\infty)\}$. Given a TFSM M, we define $\max(M)$ as the maximum between the greatest timeout value of the function Δ_S (different from ∞) and the greatest integer constant (different from ∞) appearing in the guards of λ_S. The set $\mathbb{I}_N$ defines a discretization of the clock values of TFSMs. The following lemma proves that such a discretization is correct, namely, that a TFSM cannot distinguish between two timed states where the discrete state is the same and the values of the clocks are in the same interval of $\mathbb{I}_N$.

Lemma 1 *Let $M = (S, I, O, \lambda_S, s_0, \Delta_S)$ be a deterministic TFSM, $N = \max(M)$, and let (s, x) and (s, x') be two timed states of M such that $x, x' \in \langle n, n' \rangle$ for some interval $\langle n, n' \rangle \in \mathbb{I}_N$. Then $\lambda_S(s, x, i) = \lambda_S(s, x', i)$ for every input symbol $i \in I$.*

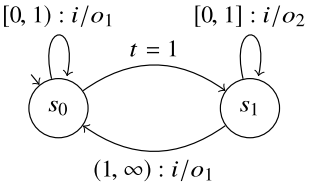
Proof Suppose by contradiction that there exist two timed states (s, x) and (s, x') such that $x, x' \in \langle n, n' \rangle$ for some $\langle n, n' \rangle \in \mathbb{I}_N$ and $\lambda_S(s, x, i) \neq \lambda_S(s, x', i)$. Since $x \neq x'$ we have that the interval $\langle n, n' \rangle$ must be an open interval of the form $(n, n+1)$ (it cannot be a point interval $[n, n]$) with $n = \lfloor x \rfloor = \lfloor x' \rfloor$ and $n+1 = \lceil x \rceil = \lceil x' \rceil$. Suppose, without loss of generality, that $\lambda_S(s, x, i)$ is defined and equal to (s', o). By the definition of TFSM we have that there exists a transition $(s, i, \langle t_{min}, t_{max} \rangle, o, s') \in \lambda_S$ such that $x \in \langle t_{min}, t_{max} \rangle$. Since t_{min}, t_{max} are nonnegative integers (or ∞), it is easy to see that $(n, n+1) \subseteq \langle t_{min}, t_{max} \rangle$. Hence, $x' \in \langle t_{min}, t_{max} \rangle$ and thus $\lambda_S(s, x', i) = (s', o) = \lambda_S(s, x, i)$, in contradiction with the hypothesis that $\lambda_S(s, x, i) \neq \lambda_S(s, x', i)$. □

We can exploit the discretization given by $\mathbb{I}_N$ to build the abstract FSM as follows. States of the abstract FSM will be pairs $(s, \langle n, n' \rangle)$ where s is a state of M and $\langle n, n' \rangle$ is either a point-interval $[n, n]$ or an open interval $(n, n+1)$ from the set $\mathbb{I}_N$ defined above, where $N = \max(M)$. Transitions can be either standard input/output transitions labelled with pairs from $I \times O$ or "time elapsing" transitions labelled with the special pair $(\mathbb{t}, \mathbb{t})$, which intuitively represents a time delay $0 < t^* < 1$ without inputs.

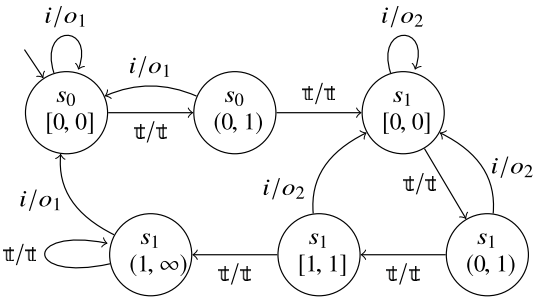
Definition 3 Given a TFSM with timeouts and timed guards $M = (S, I, O, \lambda_S, s_0, \Delta_S)$, let $N = \max(M)$. We define the $\mathbb{t}$-*abstract FSM* $A_M = (S \times \mathbb{I}_N, I \cup \{\mathbb{t}\}, O \cup \{\mathbb{t}\}, \lambda_A, (s_0, [0,0]))$ as the untimed FSM such that:

- $(s, [n,n]) \xrightarrow{\mathbb{t},\mathbb{t}} (s, (n, n+1))$ if and only if $n+1 \leq \Delta_S(s)_{\downarrow\mathbb{N}}$;
- $(s, (n, n+1)) \xrightarrow{\mathbb{t},\mathbb{t}} (s, [n+1, n+1])$ if and only if $n+1 < \Delta_S(s)_{\downarrow\mathbb{N}}$;
- $(s, (n, n+1)) \xrightarrow{\mathbb{t},\mathbb{t}} (s', [0,0])$ if and only if $\Delta_S(s) = (s', n+1)$;
- $(s, [N,N]) \xrightarrow{\mathbb{t},\mathbb{t}} (s, (N, \infty))$ and $(s, (N, \infty)) \xrightarrow{\mathbb{t},\mathbb{t}} (s, (N, \infty))$ if and only if $\Delta_S(s)_{\downarrow\mathbb{N}} = \infty$;
- $(s, \langle n, n' \rangle) \xrightarrow{i,o} (s', [0,0])$ if and only if there exists $(s, i, \langle t, t' \rangle, o, s') \in \lambda_S$ such that $\langle n, n' \rangle \subseteq \langle t, t' \rangle$.

Figure 2 shows an example of a TFSM with timeouts and its $\mathbb{t}$-abstraction. In this case the untimed abstraction accepts untimed input words on $I \cup \{\mathbb{t}\}$. The delay is implicitly represented by sequences of the special input symbol $\mathbb{t}$ interleaving the occurrences of the real input symbols from I. The representation of delays in the abstraction is quite involved:



(a) TFSM with timeouts and timed guards M



(b) $\mathbb{t}$-Abstract untimed FSM A_M

Fig. 2 $\mathbb{t}$-abstraction of TFSM with timeout and timed guards

- two input symbols from I with *no interleaving* of $\mathbb{t}$ symbols represent a delay of 0 time units;
- an *even number* $2n$ of $\mathbb{t}$ symbols represents a delay of *exactly* n time units;
- an *odd number* $2n + 1$ of $\mathbb{t}$ symbols represents a delay t included in the open interval $(n, n + 1)$.

The notion of abstraction of a timed word captures the above intuition.

Definition 4 Let $\mathbb{t}(t)$ be a function mapping a delay $t \in \mathbb{R}^+$ to a sequence of $\mathbb{t}$ as follows: $\mathbb{t}(0) = \varepsilon$, $\mathbb{t}(t) = \mathbb{t}^{2t}$ if $\lfloor t \rfloor = t$ and $t > 0$, $\mathbb{t}(t) = \mathbb{t}^{2\lfloor t \rfloor + 1}$ otherwise. Given a finite alphabet A and a finite timed word $v = (a_1, t_1)\,(a_2, t_2)(a_3, t_3) \ldots (a_m, t_m)$, we define its $\mathbb{t}$-*abstraction* as the finite word $\mathbb{t}(v) = \mathbb{t}(t_1)a_1\mathbb{t}(t_2 - t_1) \ldots \mathbb{t}(t_j - t_{j-1})a_j\mathbb{t}(t_{j+1} - t_j) \ldots \mathbb{t}(t_{m-1} - t_m)a_m$.

$\mathbb{t}$-bisimulation connects *timed states* (s, x) of a timed FSM with states of an untimed FSM. Conditions 1. and 2. formalize the connection between timed transitions and the special symbol $\mathbb{t}$. Conditions 3. and 4. formalize the connection between the actual input/output transitions in the two machines.

Definition 5 Given a TFSM with timed guards and timeouts $T = (S, I, O, \lambda_S, s_0, \Delta_S)$ and an untimed FSM $U = (R, I \cup \{\mathbb{t}\}, O \cup \{\mathbb{t}\}, \lambda_R, r_0)$, a $\mathbb{t}$-*bisimulation* is a relation $\sim \subseteq (S \times \mathbb{R}^+) \times R$ that respects the following conditions for every pair of states $(s, x) \in S \times \mathbb{R}^+$ and $r \in R$ such that $(s, x) \sim r$:

1. if $(s,x) \xrightarrow{t} (s',x')$ with $0 < t < 1$ and either $x \in \mathbb{N}$ or $x + t \in \mathbb{N}$ then there exists $r' \in R$ such that $r \xrightarrow{\mathbb{t},\mathbb{t}} r'$ and $(s',x') \sim r'$;
2. if $r \xrightarrow{\mathbb{t},\mathbb{t}} r'$ then for every $0 < t < 1$ such that either $x \in \mathbb{N}$ or $x + t \in \mathbb{N}$ there exists $(s',x') \in S \times \mathbb{R}^+$ such that $(s,x) \xrightarrow{t} (s',x')$ and $(s',x') \sim r'$;
3. if $(s,x) \xrightarrow{i,o} (s',0)$ then there exists $r' \in R$ such that $r \xrightarrow{i,o} r'$ and $(s',0) \sim r'$;
4. if $r \xrightarrow{i,o} r'$ then there exists $(s',0) \in S \times \mathbb{R}^+$ such that $(s,x) \xrightarrow{i,o} (s',0)$ and $(s',0) \sim r'$.

T and U are $\mathbb{t}$-*bisimilar* if there exists a $\mathbb{t}$-bisimulation $\sim \subseteq (S \times \mathbb{R}^+) \times R$ such that $(s_0,0) \sim r_0$.

To prove that $\mathbb{t}$-bisimilar machines have the same behavior we need to introduce the following technical result, connecting timed transitions with the special symbol $\mathbb{t}$.

Lemma 2 *Given a TFSM with timed guards and timeouts* $T = (S, I, O, \lambda_S, s_0, \Delta_S)$ *and an untimed FSM* $U = (R, I \cup \{\mathbb{t}\}, O \cup \{\mathbb{t}\}, \lambda_R, r_0)$, *every* $\mathbb{t}$*-bisimulation relation* $\sim \subseteq (S \times \mathbb{R}^+) \times R$ *respects the following properties for every* $(s,0) \sim r$ *and* $t > 0$:

(i) *if* $(s,0) \xrightarrow{t} (s',x')$ *then there exists* r' *such that* $(s',x') \sim r'$ *and* $r \xrightarrow{\mathbb{t}(t),\mathbb{t}(t)} r'$;
(ii) *if* $r \xrightarrow{\mathbb{t}(t),\mathbb{t}(t)} r'$ *then there exists* $(s',x') \sim r'$ *such that* $(s,0) \xrightarrow{t} (s',x')$.

Proof The proof is by induction on the number of symbols n in $\mathbb{t}(t)$. For the basis of the induction, suppose $n = 1$ and let $(s,0) \sim r$: by the definition of $\mathbb{t}(t)$, we have that $0 < t < 1$. The two properties are a direct consequence of the definition of $\mathbb{t}$-bisimulation. By condition 1 of Definition 5, we have that for every $0 < t < 1$, $(s,0) \xrightarrow{t} (s',x')$ implies that there exists r' such that $(s',x') \sim r'$ and $r \xrightarrow{\mathbb{t},\mathbb{t}} r'$. By condition 2 of Definition 5, we have that for every $0 < t < 1$, $r \xrightarrow{\mathbb{t},\mathbb{t}} r'$ implies that there exists $(s',x') \sim r'$ such that $(s,0) \xrightarrow{t} (s',x')$.

For the inductive case, suppose that $n \geq 1$ and that the Lemma holds for $n-1$. Now, let $(s,0) \xrightarrow{t} (s',x')$. Two cases may arise: either $\lfloor t \rfloor = t$ or $\lfloor t \rfloor > t$. In the former case, consider the timed state (s'',x'') such that $(s,0) \xrightarrow{t-0.5} (s'',x'') \xrightarrow{0.5} (s',x')$.[1] Since the number of symbols in $\mathbb{t}(t-0.5)$ is exactly $n-1$, by inductive hypothesis we have that there exists r'' such that $(s'',x'') \sim r''$ and $r \xrightarrow{\mathbb{t},\mathbb{t}^{n-1}} r''$. By condition 1 of Definition 5, we have that there exists r' such that $(s',x') \sim r'$ and $r'' \xrightarrow{\mathbb{t},\mathbb{t}} r'$ and thus that $r \xrightarrow{\mathbb{t}(t),\mathbb{t}(t)} r'$. To prove property *(ii)*, suppose $r \xrightarrow{\mathbb{t}(t),\mathbb{t}(t)} r'$ and consider the state r'' such that $r \xrightarrow{\mathbb{t},\mathbb{t}^{n-1}} r'' \xrightarrow{\mathbb{t},\mathbb{t}} r'$. By inductive hypothesis we have that there exists $(s'',x'') \sim r''$ such that $(s,0) \xrightarrow{t-0.5} (s'',x'')$. By condition 2 of Definition 5 it is possible to find a state (s',x') such that $(s',x') \sim r'$ and $(s'',x'') \xrightarrow{0.5} (s',x')$. This shows that $(s,0) \xrightarrow{t} (s',x')$. When $\lfloor t \rfloor > t$, we can consider the timed state (s'',x'') such that $(s,0) \xrightarrow{\lfloor t \rfloor} (s'',x'') \xrightarrow{t-\lfloor t \rfloor} (s',x')$. Since the number of symbols in $\mathbb{t}(\lfloor t \rfloor)$ is exactly$n-1$,

[1] Here 0.5 is an arbitrary value chosen for the sake of simplicity. Indeed, the argument holds for every delay $0 < t^* < 1$.

by an argument similar to the above we can prove that both properties *(i)* and *(ii)* hold also in this case, concluding the proof. □

The following lemma proves that $\mathbb{t}$-bisimilar machines have the same behavior.

Lemma 3 *Given a TFSM with timeouts and timed guards* $T = (S, I, O, \lambda_S, s_0, \Delta_S)$ *and an untimed FSM* $U = (R, I \cup \{\mathbb{t}\}, O \cup \{\mathbb{t}\}, \lambda_R, r_0)$, *if there exists a* $\mathbb{t}$*-bisimulation* $\sim$ *such that* $(s_0, 0) \sim r_0$ *then for every timed input word* $v = (i_1, t_1) \dots (i_m, t_m)$ *we have that* $\mathbb{t}(B_T(v)) = B_U(\mathbb{t}(v))$.

Proof We prove the lemma by showing that the following claim holds:

for every pair of states $s \in S$ *and* $r \in R$ *such that* $(s, 0) \sim r$ *and timed word* v, $\lambda_S(s, v) = (s', w)$ *if and only if* $\lambda_R(r, \mathbb{t}(v)) = (r', \mathbb{t}(w))$ *with* $(s', 0) \sim r'$.

We prove the claim by induction on the length m of the input word. Suppose $m = 1$, $v = (i_1, t_1)$ and $w = (o_1, t_1)$. We have to show that $\lambda_S(s, (i_1, t_1)) = (s', (o_1, t_1))$ if and only if $\lambda_R(r, \mathbb{t}(i_1, t_1)) = (r', \mathbb{t}(o_1, t_1))$ for some r' such that $(s', 0) \sim r'$.

To prove the direct implication, suppose $\lambda_S(s, v) = (s', w)$. By the definition of TFSM we have that $\lambda_S(s, (i_1, t_1)) = (s_1, (o_1, t_1))$ if and only if there exists a timed state (s', x') such that $(s, 0) \xrightarrow{t_1} (s', x') \xrightarrow{i_1, o_1} (s_1, 0)$. We distinguish between two cases depending on the value of t_1.

- If $t_1 = 0$, then $(s, 0) \xrightarrow{0} (s, 0) \xrightarrow{i_1, o_1} (s_1, 0)$ and by condition 3 of the definition of $\mathbb{t}$ -bisimulation (since $(s, 0) \xrightarrow{i_1, o_1} (s_1, 0)$), there exists $r_1 \in R$ such that $r \xrightarrow{i_1, o_1} r_1$. Hence, $\lambda_R(r, \mathbb{t}(i_1, t_1)) = (r_1, \mathbb{t}(o_1, t_1))$.
- If $t_1 > 0$, by Lemma 2 (i), there exists r' such that $r \xrightarrow{\mathbb{t}(t_1), \mathbb{t}(t_1)} r'$ and $(s', x') \sim r'$. By condition 3 of the definition of $\mathbb{t}$-bisimulation (since $(s', x') \xrightarrow{i_1, o_1} (s_1, 0)$), we have that it is possible to find a state $r_1 \in R$ such that $r' \xrightarrow{i_1, o_1} r_1$. This implies that under input $\mathbb{t}(t_1) i_1 = \mathbb{t}(i_1, t_1)$ the FSM U produces the output word $\mathbb{t}(t_1) o_1 = \mathbb{t}(o_1, t_1)$, and thus we can conclude that $\lambda_R(r, \mathbb{t}(i_1, t_1)) = (r_1, \mathbb{t}(o_1, t_1))$.

To prove the converse implication, suppose $\lambda_R(r, \mathbb{t}(i_1, t_1)) = (r_1, \mathbb{t}(o_1, t_1))$. We distinguish between two cases depending on the value of t_1.

- If $t_1 = 0$, then by the assumption $\lambda_R(r, \mathbb{t}(i_1, 0)) = (r_1, \mathbb{t}(o_1, 0))$ we have that $r \xrightarrow{i_1, o_1} r_1$, and so by condition 4 of the definition of $\mathbb{t}$-bisimulation, there exists $(s_1, 0) \in S \times \mathbb{R}^+$ such that $(s, 0) \xrightarrow{i_1, o_1} (s_1, 0)$. Hence, $\lambda_S(s, (i_1, 0)) = (s_1, (o_1, 0))$.
- If $t_1 > 0$, then by the assumption $\lambda_R(r, \mathbb{t}(i_1, t_1)) = (r_1, \mathbb{t}(o_1, t_1))$ there exists $r' \in R$ such that $r \xrightarrow{\mathbb{t}(t_1), \mathbb{t}(t_1)} r' \xrightarrow{i_1, o_1} r_1$. By Lemma 2 (ii), there exists $(s', x') \in S \times \mathbb{R}^+$ such that $(s, 0) \xrightarrow{t_1} (s', x')$ and $(s', x') \sim r'$. By condition 4. of the definition of $\mathbb{t}$-bisimulation, we have that there exists a timed state $(s_1, 0)$ such that $(s', x') \xrightarrow{i_1, o_1} (s_1, 0)$. This implies that under input (i_1, t_1) the TFSM T produces the timed output word (o_1, t_1), and thus we can conclude that $\lambda_S(s, (i_1, t_1)) = (s_1, (o_1, t_1))$.

Since our machines may be partial, we have that $\lambda_S(s, (i_1, t_1))$ and $\lambda_R(r, \mathbb{t}(v))$ are not necessarily defined. However, the above argument also shows that $\lambda_S(s, (i_1, t_1))$ is defined if and only if $\lambda_R(r, \mathbb{t}(v))$ is defined.

To prove the inductive case, suppose $m > 1$, $v = (i_1, t_1) \dots (i_m, t_m)$ and $w = (i_1, t_1) \dots (i_m, t_m)$. Now, let $v' = (i_1, t_1) \dots (i_{m-1}, t_{m-1})$ and $w' = (o_1, t_1) \dots (o_{m-1}, t_{m-1})$. By inductive hypothesis, we have that $\lambda_S(s, v') = (s_{m-1}, w')$ if and only if $\lambda_R(r, \mathbb{t}(v')) = (r_{m-1}, \mathbb{t}(w'))$ for some $(s_{m-1}, 0) \sim r_{m-1}$, and that $\lambda_S(s_{m-1}, (i_m, t_m - t_{m-1})) = (s_m, (o_m, t_m - t_{m-1}))$ if and only if $\lambda_R(r_{m-1}, \mathbb{t}(i_m, t_m - t_{m-1})) = (r_m, \mathbb{t}(o_m, t_m - t_{m-1}))$ for some $(s_m, 0) \sim r_m$. This implies that $\lambda_S(s, v'(i_m, t_m)) = (s_m, w'(o_m, t_m))$ if and only if $\lambda_R(r, \mathbb{t}(v)) = \lambda_R(r, \mathbb{t}(v'(i_m, t_m))) = (r_m, \mathbb{t}(w')\mathbb{t}(o_m, t_m - t_{m-1})) = (r_m, \mathbb{t}(w))$, and thus that the claim holds also for m.

To conclude the proof of the Lemma it is sufficient to recall that from the definition of behaviour we have that $B_T(v) = w$ if and only if $\lambda_S(s_0, v) = (s_m, w)$ for some state $s_m \in S$. From $(s_0, 0) \sim r_0$ (hypothesis of the lemma) we can conclude that $\lambda_R(r_0, \mathbb{t}(v)) = (r_m, \mathbb{t}(w))$ and thus that $B_U(\mathbb{t}(v)) = \mathbb{t}(w) = \mathbb{t}(B_T(v))$. □

Theorem 1 *A TFSM with timeouts and timed guards M is $\mathbb{t}$-bisimilar to the abstract FSM A_M.*

Proof The relation $\sim = \{((s, x), (s, \langle n, n' \rangle)) \mid x \in \langle n, n' \rangle\}$ is a $\mathbb{t}$-bisimulation for M and A_M. □

We can use the above theorem to solve the equivalence problem for TFSM with timed guards.

Corollary 1 *Let M and M' be two TFSM with timeouts and timed guards. Then M and M' are equivalent if and only if the two abstract FSM A_M and $A_{M'}$ are equivalent.*

Proof The claim is a direct consequence of Theorem 1 and Lemma 3. □

3 From untimed FSMs to TFSMs

In the previous section we have shown how to build an abstract untimed FSM that represents the behaviour of a TFSM, by means of appropriate notions of bisimulation and of abstraction of timed words. In this section we study the conditions under which the opposite transformation is possible: we take an untimed FSM that accepts and produces words from input and output alphabets that include the special symbol $\mathbb{t}$, and we show how to build an equivalent TFSM with timeouts and timed guards, under the same notion of abstraction of timed words.

Now, let I and O be, respectively, the input and output alphabets of our machines. We are interested in studying untimed FSMs that accept words in $(I \cup \{\mathbb{t}\})^*$ and produce words in $(O \cup \{\mathbb{t}\})^*$. Clearly, not all untimed FSMs represent valid timed behaviours. In particular, since in our TFSMs model outputs are instantaneously produced when an input is received, and since a TFSM cannot stop the advancing of time, we have that a deterministic untimed FSM $U = (R, I \cup \{\mathbb{t}\}, O \cup \{\mathbb{t}\}, \lambda_R, r_0)$ can be transformed into a TFSM only if every state r of U respects the following two conditions:

1. $\lambda_R(r, \mathbb{t})$ is defined and such that $\lambda_R(r, \mathbb{t}) = (r', \mathbb{t})$ for some $r' \in R$ (when the input $\mathbb{t}$ is received, the FSM should produce the output $\mathbb{t}$);
2. for every input $i \in I$, if $\lambda(r, i)$ is defined then $\lambda(r, i) = (r', o)$ for some output $o \in O$ and state $r' \in R$ (when an input from I is received, the FSM produces an output from O).

We call any untimed FSM that respects the above two conditions *time progressive*.

Notice that the $\mathbb{t}$-abstractions of a deterministic TFSM built following Definition 3 is always a time progressive deterministic FSM. In the following we prove that every *deterministic time progressive* FSM can be transformed into an equivalent deterministic TFSM with timeouts and timed guards. Since we cannot directly compare the behavior of an untimed FSM with the behavior of a timed FSM, we will use the notion of $\mathbb{t}$-abstraction of a timed word (Definition 4) to compare timed and untimed machines.

Definition 6 Given a deterministic and time progressive FSM $U = (R, I \cup \{\mathbb{t}\}, O \cup \{\mathbb{t}\}, \lambda_R, r_0)$, and a TFSM with timed guards and timeouts $T = (S, I, O, \lambda_S, s_0, \Delta_S)$, we say that T *refines* U if and only if for every timed input word $v = (i_1, t_1) \dots (i_m, t_m)$ we have that $B_U(\mathbb{t}(v)) = \mathbb{t}(B_T(v))$.

The intuition behind the construction is the following. Since we start from a deterministic and time progressive FSM U, from every state of U there exists exactly one transition with input $\mathbb{t}$ (and output $\mathbb{t}$). Hence, given a state s we can build the (infinite) "delay run"

$$\rho^s_{\mathbb{t}} = s \xrightarrow{\mathbb{t},\mathbb{t}} s_1 \xrightarrow{\mathbb{t},\mathbb{t}} s_2 \xrightarrow{\mathbb{t},\mathbb{t}} \dots$$

Since the number of states of U is finite, we have that the delay run is "lasso shaped", namely, that it consists of a prefix $s \xrightarrow{\mathbb{t},\mathbb{t}} \dots \xrightarrow{\mathbb{t},\mathbb{t}} s_p$ followed by the infinite repetition of a loop $s_p \xrightarrow{\mathbb{t},\mathbb{t}} \dots \xrightarrow{\mathbb{t},\mathbb{t}} s_p$.

The refined TFSM T will have the same set of states of U. Then, for every state s the delay run is computed, and the transitions and timeouts are defined as follows:

- every *I*/*O* transition leaving a state in the prefix is replaced with a timed transition from s with an appropriate timed guard;
- a timeout corresponding to the length of the prefix forces T to switch from s to a state in the loop.

Algorithm 1 Transform a FSM into a TFSM with timeouts and timed guards

Require: A time progressive and deterministic FSM $U = (S, I \cup \{\mathbb{t}\}, O \cup \{\mathbb{t}\}, \lambda_U, s_0)$
Ensure: A TFSM $T = (S, I, O, \lambda_T, s_0, \Delta_T)$ that refines U

```
1: function REFINE(U)
2:     λ_T ← ∅
3:     Δ_T ← ∅
4:     T ← (S, I, O, λ_T, s_0, Δ_T)
5:     for all s ∈ S do
6:         ADDTIMEDTRANS(s, U, T)
7:     end for
8:     return T
9: end function
```

Algorithms 1 and 2 describe the above procedure in detail. To simplify the code, we will unfold the final loop once, and put the timeout in correspondence to the second occurrence of s_p in the delay run. Moreover, since U is assumed to be deterministic, we consider the transition relation as a partial function $\lambda_U : S \times (I \cup \{\mathbb{t}\}) \mapsto S \times (O \cup \{\mathbb{t}\})$ returning the next state and the output.

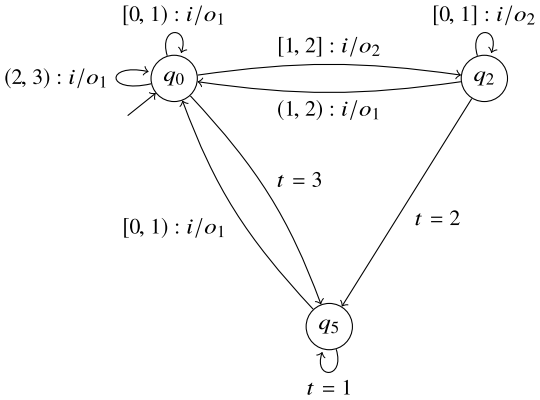


Fig. 3 Example of application of Algorithm 1

Algorithm 2 Add timed transitions to a state s

```
1:  function AddTimedTrans(s, U, T)
2:      for all r ∈ S do
3:          Marked(r) ← False
4:      end for
5:      r ← s
6:      g ← [0, 0]
7:      while not Marked(r) do
8:          Marked(r) ← True
9:          for all i ∈ I such that λ_U(r, i) is defined do
10:             (r′, o) ← λ_U(r, i)
11:             add (s, i, g, o, r′) to λ_T
12:         end for
13:         r ← λ_U(r, ⊥) ↓_S
14:         if g = [n, n] then
15:             g ← (n, n + 1)
16:         else if g = (n, n + 1) then
17:             g ← [n + 1, n + 1]
18:         end if
19:     end while
20:     if g = [n, n] then
21:         Δ_T(s) = (r, n)          ▷ prefix of even length: set the timeout and return
22:     else if g = (n, n + 1) then  ▷ prefix of odd length: unfold it one more step
23:         for all i ∈ I do
24:             (r′, o) ← λ_U(r, i)
25:             add (s, i, g, o, r′) to λ_T
26:         end for
27:         r ← λ_U(r, ⊥) ↓_S
28:         Δ_T(s) = (r, n + 1)
29:     end if
30: end function
```

Figure 3 shows the TFSM with timeouts and timed guards that can be obtained by applying Algorithm 1 to the untimed FSM of Fig. 2(b), where the states have been renamed as follows:

$$(s_0, [0,0]) = q_0 \quad (s_0, (0,1)) = q_1 \quad (s_1, [0,0]) = q_2$$
$$(s_1, (0,1)) = q_3 \quad (s_1, [1,1]) = q_4 \quad (s_1, (1,\infty)) = q_5$$

In the picture, transitions with adjacent guards have been merged: for instance, the application of the algorithm creates the transition $(q_0, i, o_1, [0, 0], q_0)$ and the transition $(q_0, i, o_1, (0, 1), q_0)$ that are merged into the unique transition $(q_0, i, o_1, [0, 1), q_0)$ in the picture. The picture includes only the states that are reachable from the initial state q_0. This shows that in the final result only the three states q_0, q_2 and q_5 are relevant: the other states have been replaced by either timed guards or timeouts.

To better understand how Algorihm 1 works, let us review the application of function AddTimedTrans (Algorithm 2) to the initial state q_0 (state $(s_0, [0, 0])$ in the picture) of the untimed FSM A_M of Fig. 2(b). The procedure starts by unmarking all states of A_M and by initialising the current state r to q_0 and the current guard g to [0, 0]. Then the while loop of lines 7–19 follows the sequence of $\mathbb{t}/\mathbb{t}$ transitions in A_M, marking the states it reaches, until a previously marked state is found. At lines 7-9, for every I/O transition exiting the current state, a corresponding timed transition labelled with the current value of g is added to the TFSM. Then the current state r is updated to the next state in the sequence of $\mathbb{t}/\mathbb{t}$ transitions and g is increased following the sequence $[0, 0], (0, 1), [1, 1], (1, 2), \ldots$. In this example, the first iteration of the while loop considers all I/O transitions exiting from the state q_0 of A_M, namely the transition $q_0 \xrightarrow{i/o_1} q_0$, and adds the transition $q_0 \xrightarrow{[0,0]:i/o_1} q_0$ to the TFSM (the initial value of g is indeed [0, 0]). Then r is updated to q_1, g to (0, 1) and the second iteration is started. The transition $q_1 \xrightarrow{i/o_1} q_0$ corresponds to the transition $q_0 \xrightarrow{(0,1):i/o_1} q_0$ in the TFSM. Notice that the starting state of the timed transition is still q_0. The $\mathbb{t}/\mathbb{t}$ transition between q_0 and q_1 of A_M models the fact that the machine waits for a time included in the interval (0, 1) before accepting an input. This situation is modelled in the TFSM by adding the guard (0, 1) to the transition while keeping q_0 as starting state. Then the loop continues by adding the following transitions to the TFSM:

$$q_0 \xrightarrow{[1,1]:i/o_2} q_2 q_0 \xrightarrow{(1,2):i/o_2} q_2 \qquad q_0 \xrightarrow{[2,2]:i/o_2} q_2 q_0 \xrightarrow{(2,3):i/o_1} q_0$$

At this point, the current state r of A_M is q_5 (i.e., $(s_1, (1, \infty))$) and the guard g is (2, 3). Because of the self loop on $\mathbb{t}/\mathbb{t}$ of A_M in state q_5, at the end of the loop r does not change and g is updated to [3, 3]: a previously marked state is reached and the loop terminates. Lines 20–29 of AddTimedTrans set the timeout at state q_0 to $(q_5, 3)$, terminating the function call. The value of the timeout is set to 3 because the first marked state is reached after 6 $\mathbb{t}/\mathbb{t}$ transitions, which corresponds to 3 time units. A subsequent call to AddTimedTrans on state q_5 will set the timeout at state q_5 to $(q_5, 1)$ (i.e., the self-loop on $t = 1$ depicted in the figure), to model the fact that in the untimed FSM A_M there is a self-loop on $\mathbb{t}/\mathbb{t}$ at state q_5. In this way, the sequence of $\mathbb{t}/\mathbb{t}$ transitions $q_0 \xrightarrow{\mathbb{t}/\mathbb{t}} q_1 \xrightarrow{\mathbb{t}/\mathbb{t}} q_2 \xrightarrow{\mathbb{t}/\mathbb{t}} q_3 \xrightarrow{\mathbb{t}/\mathbb{t}} q_4 \xrightarrow{\mathbb{t}/\mathbb{t}} q_5 \xrightarrow{\mathbb{t}/\mathbb{t}} q_5 \xrightarrow{\mathbb{t}/\mathbb{t}} q_5 \xrightarrow{\mathbb{t}/\mathbb{t}} q_5 \xrightarrow{\mathbb{t}/\mathbb{t}} q_5 \xrightarrow{\mathbb{t}/\mathbb{t}} \ldots$ of A_M is replaced by the sequence of timeout transitions $q_0 \xrightarrow{3} q_5 \xrightarrow{1} q_5 \xrightarrow{1} \ldots$. In both cases the machines can wait in q_5 forever, if no input is received in the first 3 time units. The application of AddTimedTrans to the other states of A_M builds the rest of the TFSM.

By applying the equivalence checking methodology presented in Sect. 2, we can verify that the TFSM of Fig. 3 is indeed equivalent to the TFSM of Fig. 2a. Figure 4 shows the $\mathbb{t}$-abstraction of the TFSM of Fig. 3, which is equivalent to the FSM of Fig. 2b ($\mathbb{t}$ -abstraction of the TFSM of Fig. 2a). By standard FSM state-minimization of the FSM in Fig. 4, we get a reduced FSM isomorphic to the one in Fig. 2b: the two FSMs are untimed $\mathbb{t}$-abstractions, and thus it is safe to use the standard FSM state-minimization algorithm to test for equivalence, since the algorithm preserves the language of untimed FSMs. Since

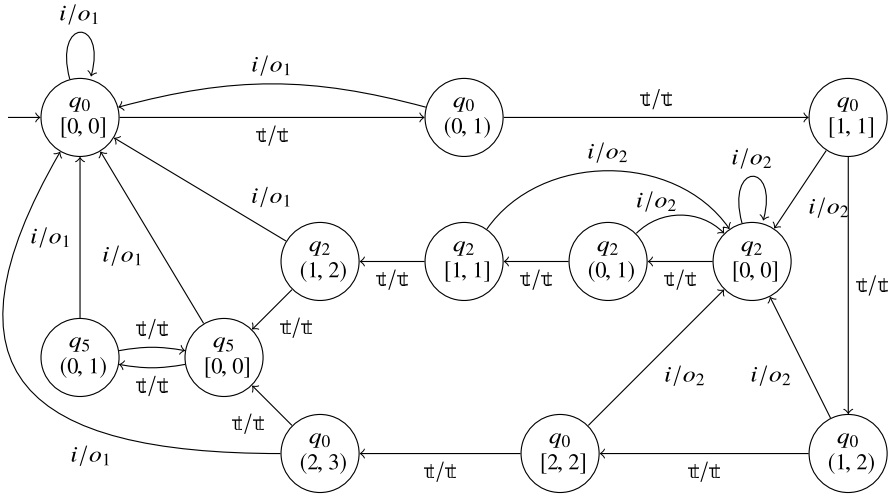


Fig. 4 $\mathbb{t}$-abstraction of the TFSM in Fig. 3

the two $\mathbb{t}$-abstractions are equivalent, equivalence of the TFSM of Fig. 3 with the TFSM of Fig. 2a follows from Corollary 1.

To formally prove the correctness of our construction we show that the TFSM T obtained from Algorithm 1 is $\mathbb{t}$-bisimilar to U. Then, by Lemma 3, we can immediately conclude that T is a refinement of U.

Theorem 2 *Given a time progressive and deterministic FSM* $U = (S, I \cup \{\mathbb{t}\}, O \cup \{\mathbb{t}\}, \lambda_U, s_0)$, *Algorithm* 1 *builds a TFSM with timeouts and timed guards* $T = (S, I, O, \lambda_T, s_0, \Delta_T)$ *for which there exists a* $\mathbb{t}$*-bisimulation* $\sim$ *such that* $(s_0, 0) \sim s_0$.

Proof Let $U = (S, I \cup \{\mathbb{t}\}, O \cup \{\mathbb{t}\}, \lambda_U, s_0)$ be a time progressive and deterministic FSM, and let $T = (S, I, O, \lambda_T, s_0, \Delta_T)$ be the TFSM built by Algorithm 1. We define the following relation between states of T and states of U:

$$\sim = \{((s, x), r) \mid r = \hat{\lambda}_U(s, \mathbb{t}(x)) \downarrow_S\} \tag{1}$$

where $\hat{\lambda}_U : S \times (I \cup \{\mathbb{t}\})^* \mapsto S \times (O \cup \{\mathbb{t}\})^*$ is the usual extension of the transition function λ_U to input words.

We show that $\sim$ is indeed a $\mathbb{t}$-bisimulation between T and U by proving that the function AddTimedTransition (Algorithm 2) respects the following invariant:

INV $(s, x) \sim r$ for all $x \in g$, and all conditions of Definition 5 are respected by the transitions in λ_T.

Before entering into the while loop, AddTimedTransition sets $r = s$ and $g = [0, 0]$. Since $\mathbb{t}(0) = \varepsilon$, we have that $(s, 0) \sim s$, and since λ_T is empty, Definition 5 is trivially respected.

Consider now a generic iteration of the while loop (lines 7–19). By the invariant, we have that $(s, x) \sim r$ for all $x \in g$. The **for** loop (lines 9–12) iterates through all transitions of U activated by an actual input $i \in I$, adding a transition (s, i, o, g, r') to λ_T for

every transition $(r, i, o, r') \in \lambda_U$. Hence, for every $x \in g$ we have that $(s, x) \xrightarrow{i,o} (r', 0)$ and $r \xrightarrow{i,o} r'$. Since $(s, x) \sim r$ and $(r', 0) \sim r'$, we have that conditions 3 and 4 of Definition 5 are respected. After updating λ_T, lines 13–18 update the value of r and g. Let us call r_{old} and g_{old} the values of r and g before the update. Then, r is set to the $\mathbb{t}$-successor of r_{old} and g is updated to the "next interval" as follows:

if $g_{old} = [n, n]$ then $g = (n, n + 1)$;
if $g_{old} = (n, n + 1)$ then $g = [n + 1, n + 1]$.

We consider the two cases separately. If $g = [n, n]$ then the only possible state (s, x) such that $x \in [n, n]$ is (s, n). Moreover, by the definition of $\sim$, since $(s, n) \sim r_{old}$ we have that $r_{old} = \hat{\lambda}_U(s, \mathbb{t}(n)) \downarrow_S$, with $\mathbb{t}(n) = \mathbb{t}^{2n}$. Since line 13 updates r to $\lambda_U(r_{old}, \mathbb{t})$, and since $\mathbb{t}(x + t) = \mathbb{t}^{2n+1}$ for every $0 < t < 1$, we have that $r = \hat{\lambda}_U(s, \mathbb{t}(x + t)) \downarrow_S$. Hence, since $(s, n) \xrightarrow{t} (s, n + t)$, $(s, n + t) \sim r$ and $r_{old} \xrightarrow{\mathbb{t},\mathbb{t}} r$ we have that conditions 1 and 2 of Definition 5 are respected. By a similar argument, if $g = (n, n + 1)$ we can show that $(s, x) \xrightarrow{t} (s, x + t)$, $(s, x + t) \sim r$ and $r_{old} \xrightarrow{\mathbb{t},\mathbb{t}} r$ for every x and t such that $0 < t < 1$ and $x + t = n + 1$, respecting conditions 1 and 2 of Definition 5 also in this case. Hence, every iteration of the while loop respects the invariant.

The loop terminates when r is a marked state, that is, when it reaches the first repetition of a state in the delay run from s. Lines 20–29 take care of setting appropriately the timeout at state s. Two different situations may arise: either $g = [n, n]$ or $g = (n, n + 1)$ for some $n \in \mathbb{N}$. In the former case, the state r is repeated after an even number of transitions, which corresponds to an integer time delay. Hence, the timeout at s is set to $\Delta_S(s) = (r, n)$. Consider now the predecessor r_{pred} of r in the delay run. By the invariant, we have that $(s, x) \sim r_{pred}$ for every $x \in (n - 1, n)$. Hence, we have that $(s, x) \xrightarrow{n-x} (r, 0)$ for every $n - 1 < x < n$, $r_{pred} \xrightarrow{\mathbb{t},\mathbb{t}} r$, $(s, x) \sim r_{pred}$ and $(r, 0) \sim r$, respecting conditions 1 and 2 of Definition 5. In the latter case ($g = (n, n + 1)$), r is repeated after an odd number of transitions. Since the timeout at s must be an integer value, lines 22–29 repeat the construction of the while loop one more time and then update r to a state that corresponds to precisely $n + 1$ time units before setting the timeout. As in the previous case, we can prove that the invariant is respected.

To conclude the proof we observe that Algoritm 1 executes AddTimedTransition on every state $s \in S$. Hence, the final TFSM T is in relation $\sim$ with U. Since $\sim$ respects all conditions of Definition 5, we have that it is a $\mathbb{t}$-bisimulation between T and U such that $(s_0, 0) \sim s_0$. □

Corollary 2 *Given a time progressive and deterministic FSM* $U = (S, I \cup \{\mathbb{t}\}, O \cup \{\mathbb{t}\}, \lambda_U, s_0)$, *Algorithm* 1 *builds a TFSM with timeouts and timed guards* $T = (S, I, O, \lambda_T, s_0, \Delta_T)$ *that refines* U.

The computational complexity of Algorithm 1 is quadratic in the number of states of the FSM U, as proved by the following theorem.

Theorem 3 *Given a time progressive and deterministic FSM* $U = (S, I \cup \{\mathbb{t}\}, O \cup \{\mathbb{t}\}, \lambda_U, s_0)$, *Algorithm* 1 *runs in* $O(|S|^2 \cdot |I|)$ *time*.

Proof The computational complexity of Algorithm 1 depends on the computational complexity of Algorithm 2. Algorithm 2 adds timed transitions originating from a state s by traversing the delay run $\rho^s_{\mathbb{t}}$ starting from s, marking the visited states and adding at most $|I|$ timed transitions for each visited state. Since marked states are never unmarked and the **while** loop stops at the first already marked state, the loop is executed for at most $|S| + 1$ iterations. Hence, the time complexity of Algorithm 2 is $O(|S| \cdot |I|)$. Since Algorithm 1 calls AddTimedTransition (Algorithm 2) on all states $s \in S$, the time complexity of the complete algorithm is $O(|S|^2 \cdot |I|)$. □

4 Intersection of TFSMs

In this section we apply the previous transformations to perform the intersection of TFSMs. In general, TFSMs can be composed to build complex systems out of simpler components. Several composition operators exist for untimed FSMs, the most relevant ones being the intersection operator, the serial composition, and synchronous and asynchronous parallel composition (see [49]). Parallel composition of TA was discussed in [41]. Preliminary work on parallel composition of TFSMs with timed guards and output delays can be found in [33], and on parallel composition of TFSMs with timeouts and output delays in [30]. When extending compositions to Timed FSMs, one must verify that TFSMs are closed under the type of composition of interest. In our setting, this means that the behaviour of the composed system should be represented by a machine with only a single clock. Here we focus on the intersection operator for which we show that closure holds.

In the following we show how the transformation from TFSMs to untimed FSMs of Sect. 2 and the transformation from untimed FSMs to TFSMs of Sect. 3 can be used to implement the intersection of TFSMs. Suppose that we have two TFSMs M_1 and M_2 and that we want to compute the intersection $M_1 \cap M_2$ whose behaviour is the intersection of the behaviours of M_1 and M_2. We can proceed as follows:

1. compute the $\mathbb{t}$-abstract FSMs A_{M_1} and A_{M_2} as in Definition 3 for, respectively, M_1 and M_2;
2. intersect A_{M_1} and A_{M_2} using the standard algorithm for untimed FSMs, obtaining the untimed FSM $C = A_{M_1} \cap A_{M_2}$;
3. compute the TFSM T that is $\mathbb{t}$-bisimilar with C using Algorithm 1.

The following theorem shows that T is equivalent to the intersection of M_1 and M_2.

Theorem 4 *Let M_1 and M_2 be two deterministic TFSMs, and let $T = \text{REFINE}A_{M_1} \cap A_{M_2}$. Then, for every timed input word $v = (i_1, t_1) \dots (i_k, t_k)$ we have that $B_T(v) = w = (o_1, t_1) \dots (o_k, t_k)$ if and only if $B_{M_1}(v)$ and $B_{M_2}(v)$ are defined and such that $B_{M_1}(v) = B_{M_2}(v) = w$.*

Proof Let M_1 and M_2 be two deterministic TFSMs, and let A_{M_1} and A_{M_2} be their respective $\mathbb{t}$-abstractions. By Definition 3 we have that A_{M_1} and A_{M_2} are deterministic and time progressive. Hence, the intersection $A_{M_1} \cap A_{M_2}$ is also deterministic and time progressive and Algorithm 1 can be applied to obtain the TFSM T.

To prove the direct implication, let $v = (i_1, t_1) \dots (i_k, t_k)$ be an input timed word and suppose that $B_T(v) = w$ for some timed output word $w = (o_1, t_1) \dots (o_k, t_k)$. Since

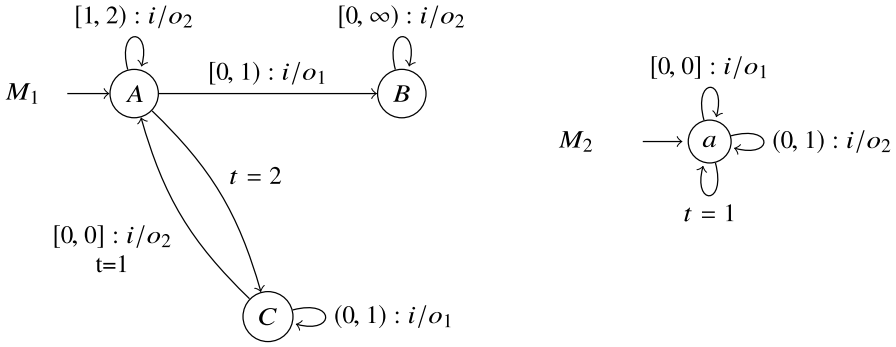


Fig. 5 TFSMs M_1 and M_2 to be intersected

$T = \text{REFINE}A_{M_1} \cap A_{M_2}$, by Corollary 2 we have that T refines $A_{M_1} \cap A_{M_2}$. Hence, by Definition 6 we have that $B_{A_{M_1} \cap A_{M_2}}(\mathbb{t}(v)) = \mathbb{t}(B_T(v)) = \mathbb{t}(w)$. Since $A_{M_1} \cap A_{M_2}$ is the intersection of A_{M_1} and A_{M_2}, we have that $B_{A_{M_1}}(\mathbb{t}(v)) = B_{A_{M_2}}(\mathbb{t}(v)) = \mathbb{t}(w)$. Since A_{M_1} and A_{M_2} are the $\mathbb{t}$ -abstraction of M_1 and M_2, by Theorem 1 and Lemma 3 we have that $\mathbb{t}(w) = B_{A_{M_1}}(\mathbb{t}(v)) = \mathbb{t}(B_{M_1}(v))$ and $\mathbb{t}(w) = B_{A_{M_2}}(\mathbb{t}(v)) = \mathbb{t}(B_{M_2}(v))$. This proves that $B_{M_1}(v)$ and $B_{M_2}(v)$ are defined and such that $B_{M_1}(v) = B_{M_2}(v) = w$.

To prove the opposite implication, let $v = (i_1, t_1) \dots (i_k, t_k)$ be an input timed word and suppose that $B_{M_1}(v)$ and $B_{M_2}(v)$ are defined and such that $B_{M_1}(v) = B_{M_2}(v) = w$ for some timed output word $w = (o_1, t_1) \dots (o_k, t_k)$. Since A_{M_1} and A_{M_2} are the $\mathbb{t}$-abstraction of M_1 and M_2, by Theorem 1 and Lemma 3 we have that $B_{A_{M_1}}(\mathbb{t}(v)) = \mathbb{t}(B_{M_1}(v)) = \mathbb{t}(w)$ and $B_{A_{M_2}}(\mathbb{t}(v)) = \mathbb{t}(B_{M_2}(v)) = \mathbb{t}(w)$. Hence, the intersection $A_{M_1} \cap A_{M_2}$ is such that $B_{A_{M_1} \cap A_{M_2}}(\mathbb{t}(v)) = B_{A_{M_1}}(\mathbb{t}(v)) = B_{A_{M_2}}(\mathbb{t}(v)) = \mathbb{t}(w)$. Since $T = \text{REFINE}A_{M_1} \cap A_{M_2}$, by Corollary 2 and Definition 6 we have that $\mathbb{t}(B_T(v)) = B_{A_{M_1} \cap A_{M_2}}(\mathbb{t}(v)) = \mathbb{t}(w)$. Hence, we have proved that $B_T(v) = w$. □

As an example, consider the TFSMs M_1 and M_2 of Fig. 5, and suppose we want to compute the *intersection* $M_1 \cap M_2$. Following the above procedure, the first step is to obtain the $\mathbb{t}$-abstract FSMs A_{M_1} and A_{M_2} in Fig. 6. Then, by applying the standard constructions for intersection and minimization of untimed FSMs, we obtain the machine C depicted in Fig. 7 and finally, using Algorithm 1, the TFSM $T = \text{REFINE}A_{M_1} \cap A_{M_2}$ of Fig. 8. It is worth pointing out that the intersection of two complete and deterministic TFSMs is still a deterministic machine, but it may be partial. This is indeed the case of our example: for instance, when the TFSM in Fig. 8 is in state 0 it can react to the input i only when the clock is in the intervals [0, 0] or (1, 2). No behaviour is specified when the clock is inside the interval (0, 1] and [2, 3). In states 1 and 13 no behaviour is specified when the clock has an integer value smaller than the timeout (0, 1, 2 and 3 for state 1, 0 for state 13).

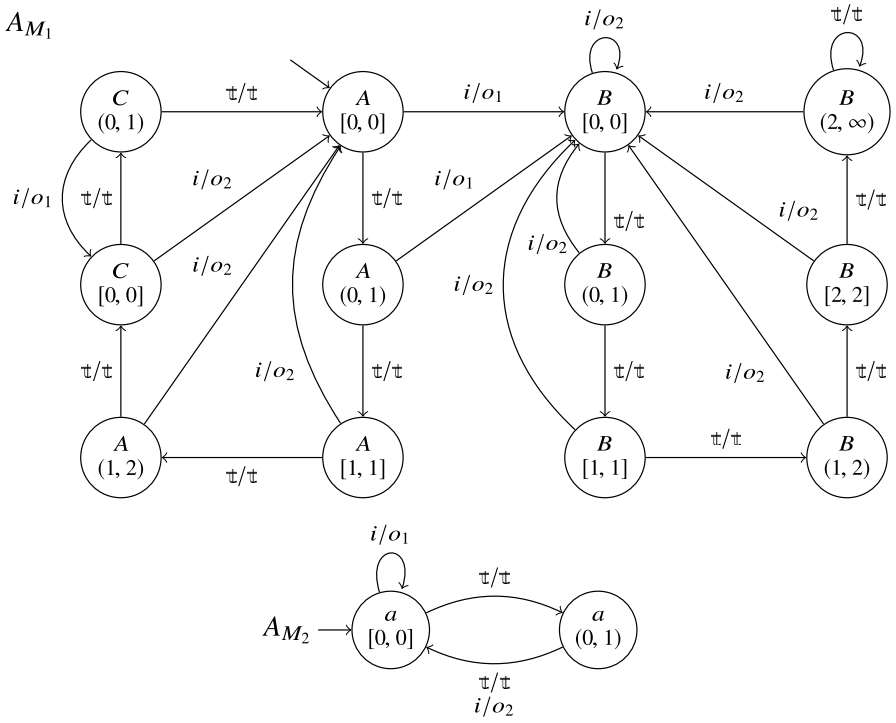


Fig. 6 Untimed abstractions of M_1 and M_2

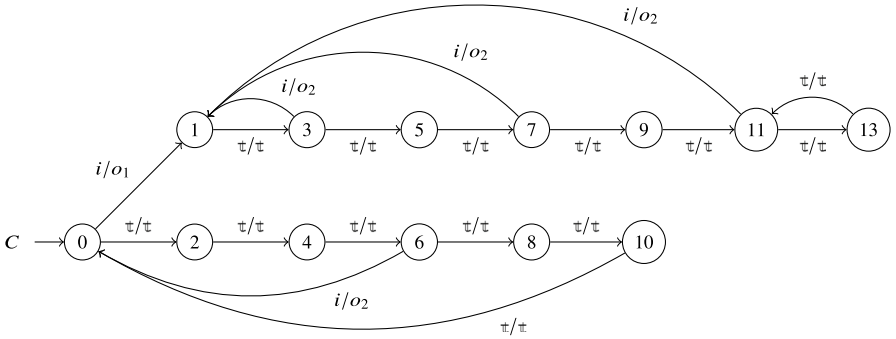


Fig. 7 The intersection of A_{M_1} and A_{M_2}

5 Timed FSMs and Timed Automata

In this section, we compare TFSMs with Timed Automata (TAs), and survey the known results on the expressivity and computability of various classes of TAs, according to their computational resources. The landscape of finite automata augmented with time is much more complex than in the case of untimed ones, where both language recognizers (FA) and producers (FSMs) share the fact that there is an underlying common model which

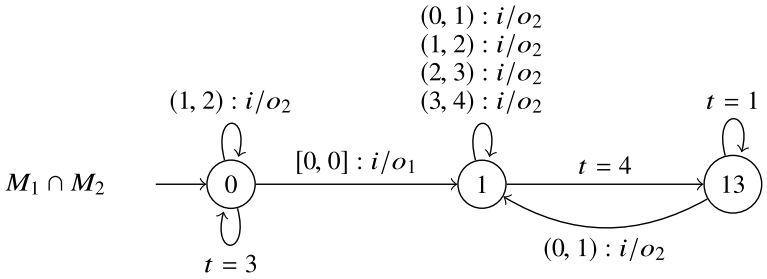


Fig. 8 The TFSM for $M_1 \cap M_2$

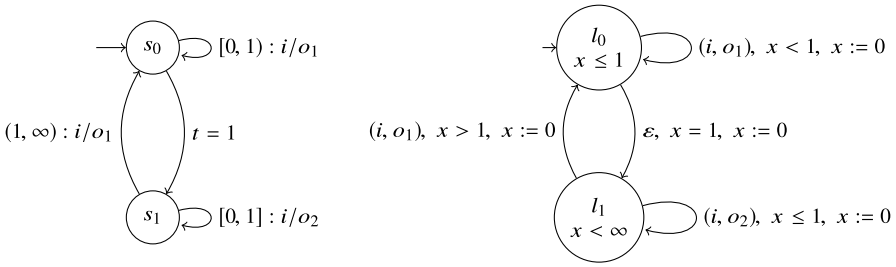


Fig. 9 Transformation from TFSM (on the left) to ε-timed automaton (on the right)

corresponds to regular languages (FSMs transform regular input languages into regular output languages). TAs are the most common formalism obtained by adding timing constraints (as clocks) to finite-state automata [3], defining timed regular recognizers. TAs are a more expressive model than TFSMs because they allow multiple clocks, invariants as conditions on clocks associated to a location, guards as conditions on clocks associated to a transition, resets by which a clock may be reset to 0 or may be kept unchanged, and states which are products of a location and clock valuations. Excellent surveys about the classes of TAs proposed in the literature can be found in [27, 52].

TFSMs can be transformed into TAs with ε-transitions (called also in the literature silent transitions or internal transitions or non-observable transitions) by the following transformation:

- there is one location of the TA for every state of the TFSM;
- given the input and output alphabets I and O of the TFSM, the alphabet of the TA is given by $I \times O$;
- as in the TFSM, the TA has a single clock, reset to zero at every transition;
- intervals on transitions are replaced with guards;
- timeouts of the TFSM are replaced by invariants and ε- transitions.

An example of such transformation is shown in Fig. 9, where on the left there is a TFSM and on the right the corresponding TA.

This reduction is not necessarily practical, since decision problems are in general undecidable for timed automata, even for restricted versions of them. In the following we mention some of these relevant results. For a classic survey on decision problems for timed automata, see [5], where the following results can be found:

1. TAs are closed under union, intersection, projection, but not under complementation.
2. The language emptiness problem is PSPACE-complete (a by-product of reachability analysis obtained by means of the region construction).
3. The universality, inclusion and equivalence problems for TAs are undecidable.
4. Deterministic TAs are closed under union. intersection and complementation, but not under projection. The language emptiness, universality, inclusion and equivalence problems for deterministic TAs are PSPACE-complete.

Further results are proved in [25] and [26], e.g., that one cannot decide whether a given timed automaton is determinizable or whether the complement of a timed regular language is timed regular.

One may wonder whether the complexity goes down, if we reduce the resources of the timed automaton. The answer is sometimes yes, but only in very restricted cases. In [1, 40] it is shown that the problem of checking language inclusion $L(A) \subseteq L(B)$ of TAs A and B is decidable if B has no ε-transitions, and either B has only one clock, or the guards of B use only the constant 0. These two cases are essentially the only decidable instances of language inclusion, in terms of restricting the various resources of timed automata. Similar conclusions for the universality problem (does a given TA accept all timed words) are drawn in [2]: the one-clock universality problem is undecidable for TAs over infinite words, and decidable for TAs over finite words, but undecidable for both if ε-transitions are allowed. Model checking and reachability of timed automata with one or two clocks are discussed in [24, 35].

It is a fact that reducing resources, like the number of clocks, may simplify some problems, but allowing ε-transitions, even with few resources, makes the problems as hard as in the general case. A score of papers [8–10, 17] investigated the expressiveness of timed automata augmented with ε-transitions, and proved the following results:

1. The class of timed languages recognized by timed automata with ε-transitions is more robust and expressive than those without them.
2. A timed automaton with ε-transitions that do not reset clocks can be transformed into an equivalent one without ε-transitions (equivalent means with the same timed language).
3. A (non-Zenonian) timed automaton such that no ε-transitions that reset clocks lie on a direct cycle can be transformed into an equivalent one without ε-transitions.
4. There is a timed automaton, with an ε-transition which resets clocks on a cycle, which is not equivalent to any timed automaton without ε-transitions.

More undecidability questions for timed automata with ε-transitions were answered in [12], e.g.: given a timed automaton with ε-transitions, it is undecidable to determine if there exists an equivalent timed automaton without ε-transitions. The problem of removing ε-transitions got a new twist in [20], where it was shown that if one allows periodic clock constraints and periodic resets (updates), then we can remove ε-transitions from a timed automaton; moreover, the authors proved that periodic updates are necessary, defining a language that cannot be accepted by any timed automaton with periodic constraints and transitions which reset clocks to 0 and no ε-transitions.

The direct transformation of TFSMs to untimed FSMs given in Sect. 2, paired with the opposite transformation from untimed FSMs to TFSMs given in Sect. 3, shows that it is possible to solve many interesting problems for TFSMs by using the standard algorithms for untimed FSMs and translating back and forth to TFSMs. Our previous work

in [13] showed that timeouts cannot be removed from TFSMs without restricting the expressive power of the model. Hence, our TFSM formalism can be viewed as a subclass of TAs with one clock that admits a restricted form of ε-transitions (i.e., timeouts), which cannot be removed, and where most of the relevant problems like equivalence and intersection are decidable.

In conclusion, timed automata are a rich model with and without ε-transitions, therefore in general their decision problems are undecidable or very difficult also for restricted versions, even more so if ε-transitions are admitted. TFSMs correspond to a decidable subclass of timed automata with ε-transitions that has never been identified before.

Related works on learning timed automata prove that deterministic timed automata with one clock can be learned efficiently [6, 47], while TAs with two or more clocks can not be learned efficiently [48]. This result has been exploited by Caldwell et al. to define the model of Time Delay Mealy Machines (TDMM) [15]: a class of FSMs with delays between application of an input and observation of an output that shares many features with our TFSM model. TDMMs are expressive enough to model PLC software and to be learned efficiently from PLC software. Another recent model of Mealy machines with a single timer (MM1Ts) introduced by Vaandrager et al. [46] can be learned efficiently by learning algorithms obtained via a reduction to the problem of learning Mealy machines. MM1Ts allow the clock to be stopped or time out in later transitions and are more expressive than TDMMs, where the clock is reset on every transition.

Another interesting restricted model are Real-Time Automata (RTAs) introduced by C. Dima [19] in 2001: they are finite automata with a labeling function (from states to an alphabet) and a time labeling function (from states to rational intervals) which together define the label of a state. RTAs work over signals that are functions with finitely many discontinuities from non-negative rational intervals $[0, e)$ (with $e > 0$) to an alphabet, so that the domain of a signal is partitioned into finitely many intervals where the signal is constant. A run is associated with a signal iff there is a sequence of partitioning points consistent with the state labels (stuttering, i.e., repetition of signal values is allowed); signals associated with an accepting run are the timed language associated to an RTA. The author states in [19] that RTAs can be viewed as a class of state-labeled timed automata over timed words (instead than signals) with a single clock which is reset at every transition (stuttering being reduced to ε-transitions). Moreover, it is claimed that RTAs are the largest timed extension of finite automata whose emptiness and universality problems are decidable, ε-transitions can be removed, there is a determinization construction, are closed under complementation, and a version of Kleene theorem holds. RTAs share some similarities with TFSMs, since they both have a single clock that is reset at every transition. Timeouts are not present in RTAs, which instead use a Kleene algebra on intervals to represent periodic timed guards, needed to remove ε-transitions.

More complex classes of timed automata have been studied, in which the interplay between variants of the basic constituents defining them yields interesting combinations of expressivity and computability.

Event-Clock Automata [4, 18, 28] (ECAs) are a determinizable robust subclass of timed automata. Event-clock automata are characterized with respect to timed automata by the fact that explicit resets of clocks are replaced by a predefined association with the input symbols such that for each input $x \in \Sigma$: a global recorder clock records the time elapsed since the last occurrence of x and a global predictor clock measures the time required for the next occurrence of x (clock valuations are determined only by the input timed words). They are closed under Boolean operations (TAs are not closed under complement) and

language inclusion is PSPACE-complete for them (it is undecidable for TAs). It is mentioned in [19] that RTAs are incomparable with ECAs, which are the largest known determinizable subclass of timed automata, since RTAs may accept languages that ECAs cannot. Since ECAs can use multiple clocks, they can accept languages that TFSMs cannot. However, TFSMs can accept some of the languages accepted by RTAs but not by ECAs.

Timed Automata with Non-Instantaneous Actions [7] are such that an action can take some time to be completed; they are more expressive than timed automata and less expressive than timed automata with ϵ-transitions. They share similarities with TFSMs where the production of outputs is not instantaneous but occurs after some time from the reception of inputs. Updatable Timed Automata were introduced in [11] as an extension to update the clocks in a more elaborate way than simply resetting them to 0; their emptiness problem is undecidable, but there are interesting decidable subclasses. Any updatable automaton belonging to some decidable subclass can be effectively transformed into an equivalent timed automaton without updates, but with ϵ-transitions. Given that they allow multiple clocks and an elaborate way to reset them, Updatable Timed Automata are more expressive than TFSMs.

A complete taxonomy of timed automata is presented in [27], and issues of undecidability are discussed in depth in [38]. Properties of timed automata are contrasted in [14] with those of a special class of hybrid automata with severe restrictions on the discrete transitions: hybrid systems with strong resets, which have the property that all the continuous variables are non-deterministically reset after each discrete transition (differently from timed automata, where flow rates are constant, and it is not compulsory to reset variables on each discrete transition). Connections between timed automata and timed discrete-event models are explored in [43].

The trade-off in preferring TAs vs. TFSMs depends also on the specific problem at hand. For instance, TAs and TFSMs are used when deriving tests for discrete event systems. However, methods for direct derivation of complete test suites over TAs return infinite test suites [44]. Therefore, to derive complete finite test suites with a guaranteed fault coverage, a TA is usually converted to an FSM and FSM-based test derivation is then used (see [23, 42]). Therefore, TFSMs may be preferred over TAs and other models when the derivation of complete tests is required (as done in [22] for TFSMs with timed guards), even though the test suites so obtained are rather long. We mention also that the FSM abstraction introduced in this paper was used in [45], to derive complete finite test suites for TFSMs with both timeouts and timed guards. Since FSMs are used for testing, state distinguishability, and state identification problems of hardware and software designs (see [16, 32, 36]), TFSMs and similar formalisms may be applied to the timed versions of these problems, instead than using TAs, as witnessed by a number of application reports e.g. to derive tests for telecommunication protocols and microcontroller systems (see [21, 22, 29–31, 34, 37, 45]). The critical role played in general by modeling timeouts to detect races in protocols of networking systems is discussed in [51]. The problem of equivalence of FSMs with timed guards and timeouts is implicitly discussed in [39], where the equivalence of two complete TFSMs is not checked by deriving and comparing the FSM abstractions of the machines, rather it is addressed by the use of a so-called distinguishing automaton that in fact is the intersection of the TFSMs.

6 Conclusions

We investigated deterministic TFSMs with a single clock, with both timed guards and timeouts. We showed that the behaviours of the timed FSMs are equivalent if and only if the behaviours of the companion untimed FSMs obtained by time-abstracting bisimulations are equivalent, so that they exhibit a good trade-off between expressive power and ease of analysis.

Then we defined and proved the correctness of the backward construction from Untimed FSMs to TFSMs. The construction starts from any deterministic FSM recognizing a subset of the language $\left((\mathbb{t}/\mathbb{t})^* I/O\right)^* (\mathbb{t}/\mathbb{t})^*$ and builds a deterministic TFSM that recognizes the corresponding timed language. Using the two constructions we showed how to intersect two deterministic TFSMs, first by transforming them into untimed FSMs, then applying the standard intersection algorithm for untimed FSMs, and then transforming back into a deterministic TFSM.

The results presented in this paper can be extended along many directions. Here we point out some of the most relevant open questions:

1. Find the direct intersection of TFSMs without abstracting time, and compare the direct approach based on TFSMs with the one based on time-abstracted FSMs.
2. Study the complement operation for TFSMs: can it be done without unfolding the TFSM and considering all time instances ?
3. Study synchronous and parallel composition of TFSMs to define and solve equations over deterministic TFSMs [50].
4. Study the previous operations for TFSMs restricted so that all states have the same timeout: are the operations on them computationally easier ?
5. Study the previous operations for TFSMs extended with output delays [37], where output symbols at each transition may be issued with some delay.
6. Study the previous operations for non-deterministic TFSMs (NDTFSMs).

Acknowledgements Davide Bresolin and Tiziano Villa acknowledge partial support from the project INdAM, GNCS 2020 (Strategic Reasoning and Automated Synthesis of Multi-Agent Systems) funded by MIUR (Italian Ministry of Education, University and Research). Tiziano Villa was partially supported by MIUR, "Project Italian Outstanding Departments, 2018-2022". Nina Yevtushenko was partly supported by the Ministry of Science and Higher Education of the Russian Federation (grant number 075-15-2020-788).

Funding Open access funding provided by Università degli Studi di Padova within the CRUI-CARE Agreement.

References

1. Abdulla P, Deneux J, Ouaknine J, Worrell J (2005) Decidability and complexity results for timed automata via channel machines. In: L. Caires, G. Italiano, L. Monteiro, C. Palamidessi, M. Yung (eds.) Automata, languages and programming, lecture notes in computer science, vol. 3580, pp. 1089–1101. Springer Berlin Heidelberg. https://doi.org/10.1007/11523468_88
2. Abdulla PA, Deneux J, Ouaknine J, Quaas K, Worrell J (2008) Universality analysis for one-clock timed automata. Fundam Inform 89(4):419–450
3. Alur R, Dill DL (1994) A theory of timed automata. Theoret Comput Sci 126(2):183–235. https://doi.org/10.1016/0304-3975(94)90010-8
4. Alur R, Fix L, Henzinger TA (1999) Event-clock automata: a determinizable class of timed automata. Theoret Comput Sci 211(1–2):253–273. https://doi.org/10.1016/S0304-3975(97)00173-4
5. Alur R, Parthasarathy M (2004) Decision problems for timed automata : a Survey. Lecture notes in computer science 3185:1–24. https://doi.org/10.1007/978-3-540-30080-9_1
6. An J, Chen M, Zhan B, Zhan N, Zhang M (2020) Learning one-clock timed automata. In: Proc. of the 26th international conference on tools and algorithms for the construction and analysis of systems (TACAS 2020), LNCS, vol. 12078, pp. 444–462. Springer. https://doi.org/10.1007/978-3-030-45190-5_25
7. Barbuti R, De Francesco N, Tesei L (2001) Timed automata with non-instantaneous actions. Fundam Inf 47(3–4):189–200
8. Bérard B, Gastin P, Petit A (1996) On the power of non-observable actions in timed automata. In: Proceedings of the 13th annual symposium on theoretical aspects of computer science, STACS '96, pp. 257–268. Springer-Verlag, London, UK, UK http://dl.acm.org/citation.cfm?id=646511.759229
9. Bérard B, Gastin P, Petit A (1997) Timed automata with non observable actions : expressive power and refinement. Tech. Rep. LIAFA 97/23, Université Denis Diderot (Paris), Paris. http://opac.inria.fr/record=b1041550
10. Bérard B, Petit A, Diekert V, Gastin P (1998) Characterization of the expressive power of silent transitions in timed automata. Fundam Inf 36(2–3):145–182
11. Bouyer P, Dufourd C, Fleury E, Petit A (2004) Updatable timed automata. Theoret Comput Sci 321(2–3):291–345. https://doi.org/10.1016/j.tcs.2004.04.003
12. Bouyer P, Haddad S, Reynier PA (2009) Undecidability results for timed automata with silent transitions. Fundam Inf 92(1–2):1–25
13. Bresolin D, El-Fakih K, Villa T, Yevtushenko N (2014) Deterministic timed finite state machines: Equivalence checking and expressive power. In: Proc. of the 5th international symposium on games, automata, logics and formal verification, GandALF 2014, Verona, Italy, September 10-12, 2014., EPTCS, vol. 161, pp. 203–216. https://doi.org/10.4204/EPTCS.161.18
14. Brihaye T, Bruyère V, Render E (2010) Formal language properties of hybrid systems with strong resets. RAIRO - Theor Inf Applic 44(1):79–111. https://doi.org/10.1051/ita/2010006
15. Caldwell B, Cardell-Oliver R, French T (2016) Learning time delay Mealy machines from programmable logic controllers. IEEE Trans Autom Sci Eng 13(2):1155–1164. https://doi.org/10.1109/TASE.2015.2496242
16. Chow TS (1978) Testing software design modeled by finite-state machines. IEEE Trans. Softw. Eng. 4(3):178–187
17. Diekert V, Gastin P, Petit A (1997) Removing epsilon-transitions in timed automata. In: Proceedings of the 14th annual symposium on theoretical aspects of computer science, STACS '97, pp. 583–594. Springer-Verlag, London, UK. http://dl.acm.org/citation.cfm?id=646512.695332
18. Dima C (2000) Removing epsilon transitions from event-clock automata. In: Proceedings of the national conference on theoretical computer science and information technology, CITTI 2000. Constanța, Romania, pp 75–81. http://www.inrialpes.fr/bip/people/cdima/work/epsilon.html
19. Dima C (2001) Real-time automata. J Autom Lang Comb 6(1):3–24
20. Dima C, Lanotte R (2009) Removing all silent transitions from timed automata. In: Ouaknine J, Vaandrager FW (eds) FORMATS, vol 5813. Lecture notes in computer science. Springer, Berlin, pp 118–132
21. El-Fakih K, Gromov M, Shabaldina N, Yevtushenko N (2013) Distinguishing experiments for timed nondeterministic finite state machines. Acta Cybern 212(2):205–222
22. El-Fakih K, Yevtushenko N, Simao A (2014) A practical approach for testing timed deterministic finite state machines with single clock. Sci Comput Progr 80:343–355. https://doi.org/10.1016/j.scico.2013.09.008
23. En-Nouaary A, Dssouli R, Khendek F (2002) Timed Wp-method: testing real-time systems. IEEE Trans Softw Eng 28(11):1023–1038. https://doi.org/10.1109/TSE.2002.1049402

24. Fearnley J, Jurdziński M (2015) Reachability in two-clock timed automata is PSPACE-complete. Inf Comput 243:26–36. https://doi.org/10.1016/j.ic.2014.12.004
25. Finkel O (2005) On decision problems for timed automata. Bull EATCS 87:185–190
26. Finkel O (2006) Undecidable problems about timed automata. In: Formal modeling and analysis of timed systems, 4th international conference, FORMATS 2006, Paris, France, September 25-27, 2006, Proceedings, pp. 187–199. https://doi.org/10.1007/11867340_14
27. Fontana P, Cleaveland R (2014) A menagerie of timed automata. ACM Comput Surv 46(3):1–56. https://doi.org/10.1145/2518102
28. Geeraerts G, Raskin JF, Sznajder N (2011) Event clock automata: from theory to practice. In: Fahrenberg U, Tripakis S (eds) Formal modeling and analysis of timed systems. Springer, Berlin Heidelberg, Berlin, Heidelberg, pp 209–224
29. Gromov M, El-Fakih K, Shabaldina N, Yevtushenko N (2009) Distinguishing non-deterministic timed finite state machines. In: Formal Techniques for Distributed Systems, Lecture Notes in Computer Science, Springer, Berlin. vol. 5522, pp. 137–151 https://doi.org/10.1007/978-3-642-02138-1_9
30. Gromov M, Tvardovskii A, Yevtushenko N (2016) Testing components of interacting timed finite state machines. In: 2016 IEEE East-West Design & Test Symposium (EWDTS), vol. 00, pp. 1–4. https://doi.org/10.1109/EWDTS.2016.7807688. doi.ieeecomputersociety.org/10.1109/EWDTS.2016.7807688
31. Hierons RM, Merayo MG, Núñez M (2009) Testing from a stochastic timed system with a fault model. J Logic Algebraic Program 78(2):98–115. https://doi.org/10.1016/j.jlap.2008.06.001
32. Kohavi Z, Jha NK (2009) Switching and Finite Automata Theory. Cambridge University Press, England
33. Kondratyeva O, Kushik N, Cavalli AR, Yevtushenko N (2013) Evaluating web service quality using finite state models. In: 2013 13th international conference on quality software, Najing, China, July 29-30, 2013, pp. 95–102 (2013). https://doi.org/10.1109/QSIC.2013.52
34. Laputenko AV, Petukhov TD, Vasnev NA (2018) Testing microcontroller based physical systems using finite transition models. In: 2018 19th international conference of young specialists on micro/nanotechnologies and electron devices (EDM), pp. 203–206. https://doi.org/10.1109/EDM.2018.8435029
35. Laroussinie F, Markey N, Schnoebelen P (2004) Model checking timed automata with one or two clocks. In: Proceedings of the 15th international conference on concurrency theory (concur'04), lecture notes in computer science, vol. 3170, pp. 387–401. Springer-Verlag, Berlin, Heidelberg
36. Lee D, Yannakakis M (1996) Principles and methods of testing finite state machines-a survey. Proc IEEE 84(8):1090–1123
37. Merayo MG, Núñez M, Rodríguez I (2008) Formal testing from timed finite state machines. Comput Netw 52(2):432–460. https://doi.org/10.1016/j.comnet.2007.10.002
38. Miller JS (2000) Decidability and complexity results for timed automata and semi-linear hybrid automata. In: Proceedings of the third international workshop on hybrid systems: computation and control, HSCC '00, pp. 296–309. Springer-Verlag, London, UK, UK. http://dl.acm.org/citation.cfm?id=646880.710453
39. Nguena Timo O, Prestat D, Rollet A (2019) Multiple Mutation Testing for Timed Finite State Machine with Timed Guards and Timeouts. In: C. Gaston, N. Kosmatov, P.L. Gall (eds.) 31th ifip international conference on testing software and systems (ICTSS), Testing Software and Systems, vol. LNCS-11812. Springer International Publishing, Paris, France. https://hal.inria.fr/hal-02341856
40. Ouaknine J, Worrell J (2004) On the language inclusion problem for timed automata: Closing a decidability gap. In: Proceedings of the 19th Annual IEEE symposium on logic in computer science, LICS '04, pp. 54–63. IEEE Computer Society, Washington, DC, USA. https://doi.org/10.1109/LICS.2004.30
41. Sifakis J, Yovine S (1996) Compositional specification of timed systems. In: Puech C, Reischuk R (eds) STACS 96. Springer, Berlin, pp 345–359
42. Springintveld J, Vaandrager F, D'Argenio PR (2001) Testing timed automata. Theor Comput Sci 254(1–2):225–257. https://doi.org/10.1016/S0304-3975(99)00134-6
43. Stergiou C, Tripakis S, Matsikoudis E, Lee E (2013) On the verification of timed discrete-event models. In: V. Braberman, L. Fribourg (eds.) Formal Modeling and Analysis of Timed Systems, Lecture Notes in Computer Science, Springer, Berlin, vol. 8053, pp. 213–227 https://doi.org/10.1007/978-3-642-40229-6_15
44. Tretmans J (1992) A formal approach to conformance testing. Ph.D. thesis, University of Twente, The Netherlands
45. Tvardovskii A, El-Fakih K, Yevtushenko N (2018) Deriving tests with guaranteed fault coverage for finite state machines with timeouts. In: Medina-Bulo I, Merayo MG, Hierons R (eds) Testing software and system. Springer International Publishing, Cham, pp 149–154
46. Vaandrager F, Bloem R, Ebrahimi M (2021) Learning mealy machines with one timer. In: Leporati A, Martín-Vide C, Shapira D, Zandron C (eds) Language and automata theory and applications. Springer International Publishing, Cham, pp 157–170

47. Verwer S, de Weerdt M, Witteveen C (2009) One-clock deterministic timed automata are efficiently identifiable in the limit. In: Proc. of the 3rd international conference on language and automata theory and Applications (LATA 2009), Lecture notes in computer science, vol. 5457, pp. 740–751. Springer. https://doi.org/10.1007/978-3-642-00982-2_63
48. Verwer S, de Weerdt M, Witteveen C (2011) The efficiency of identifying timed automata and the power of clocks. Inf Comput 209(3):606–625. https://doi.org/10.1016/j.ic.2010.11.023
49. Villa T, Petrenko A, Yevtushenko N, Mishchenko A, Brayton R (2015) Component-based design by solving language equations. Proc IEEE 103(11):2152–2167
50. Villa T, Yevtushenko N, Brayton R, Mishchenko A, Petrenko A, Sangiovanni-Vincentelli A (2012) The Unknown Component Problem: Theory and Applications, Springer, Berlin. https://doi.org/10.1007/978-0-387-68759-9
51. Vinarskii E, López J, Kushik N, Yevtushenko N, Zeghlache D (2019) A model checking based approach for detecting sdn races. In: C. Gaston, N. Kosmatov, P. Le Gall (eds.) Testing Software and Systems, Proceedings of the 31st IFIP WG 6.1 international conference on testing software and systems, ICTSS 2019, pp. 194–211. Springer International Publishing, Cham, Switzerland
52. Waez MTB, Dingel J, Rudie K (2013) A survey of timed automata for the development of real-time systems. Comput Sci Rev 9:1–26. https://doi.org/10.1016/j.cosrev.2013.05.001
53. Zhigulin M, Yevtushenko N, Maag S, Cavalli A (2011) FSM-based test derivation strategies for systems with time-outs. In: Proceedings of the 11th international conference on quality software (QSIC 2011), pp. 141–149. https://doi.org/10.1109/QSIC.2011.30