Bounded Determinization of Timed Automata with Silent Transitions

Florian Lorber[2], Amnon Rosenmann[1],
Dejan Ničković[1], and Bernhard K. Aichernig[2]

[1] AIT Austrian Institute of Technology GmbH
Vienna, Austria
[2] Institute for Software Technology
Graz University of Technology, Austria

Abstract. Deterministic timed automata are strictly less expressive than their non-deterministic counterparts, which are again less expressive than those with silent transitions. As a consequence, the problem of timed automata determinization is undecidable. This is unfortunate since deterministic automata play a major role in model-based testing, observability and implementability. However, by bounding the length of the traces in the automaton, effective determinization becomes possible. We propose a novel procedure for bounded determinization of timed automata. The procedure unfolds the automata to bounded trees, removes all silent transitions and determinizes via disjunction of guards. The proposed algorithms are optimized to the bounded setting and thus are more efficient and can handle a larger class of timed automata than the general algorithms. The approach is implemented in a prototype tool and evaluated on several examples. To our best knowledge, this is the first implementation of this type of procedure for timed automata.

1 Introduction

The design of modern embedded systems often involves the integration of interacting components I_1 and I_2 that realize some requested behavior. In early stages of the design, I_1 and I_2 are high-level and partial models that allow considerable implementation freedom to the designer. In practice, this freedom is reflected in the non-deterministic choices that are intended to be resolved during subsequent design refinement steps. In addition, the composition of two components involves their synchronization on some shared actions. Typically, the actions over which the two components interact are *hidden* and become unobservable to the user. It follows that the overall specification $I = I_1 \parallel I_2$ can be a *non-deterministic partially observable* model. The passage from a high-level model towards an implementation consists of an iteration of *refinement* steps, often based on trace or alternating trace inclusion. In practice, checking whether I' refines I often requires the *determinization* of I.

Many embedded systems must meet strict real-time requirements. Timed automata (TA) [3] are a formal modeling language that enables specification of

complex real-time systems. In contrast to the classical automata theory, deterministic TA (DTA) are strictly less expressive than the fully observable nondeterministic TA (NTA) [3, 14, 11], whereas the latter are strictly less expressive than TA with silent transitions (eNTA) [5]. This strict hierarchy of TA with respect to determinism and observability has an important direct consequence - NTA are not determinizable in general, hence refinement checking between two TA is undecidable. In addition, due to their complexity, it is rarely the case that exhaustive verification methods are used during the design of modern embedded systems. Lighter and incomplete methods, such as model-based testing [13] and bounded model checking [6] are used in practice in order to gain confidence in the design-under-test and effectively catch bugs.

In this paper, we propose a procedure for *bounded determinization* of eNTA. Given an arbitrary *strongly responsive*[1] eNTA A and a bound k, our algorithm computes a DTA $D(A)$ in the form of a timed tree, such that every timed trace consisting of at most k observable actions is a trace in A if and only if it is a trace in $D(A)$. It provides the basis for effectively implementing bounded refinement checking and test case generation procedures.

The proposed algorithms are performed in three steps: (1) we unfold the original automaton into a finite tree and rename the clocks in a way that only needs one clock reset per transition, (2) we remove the silent transitions from the tree, (3) we determinize it. Our determinization procedure results in a TA description which includes diagonal [8] and disjunctive constraints. Although non-standard, this representation is practical and optimized for the bounded setting – it avoids costly transformation of the TA into its standard form and exploits efficient heuristics in SMT solvers that can directly deal with this type of constraints in verification and testing. In addition, our focus on bounded determinization allows us to consider models, such as TA with loops containing observable as well as silent transitions with reset, that could not be determinized otherwise. We implemented the procedure in a prototype tool and evaluated it on several examples. To our best knowledge, this is the first implementation of this type of procedure for timed automata.

Running example. The different steps of the algorithms will be illustrated on a running example of a coffe-machine shown in Figure 1. After inserting a *coin*, the system heats up for zero to three seconds, followed by a *beep*-tone indicating its readyness. Alternatively, if there is no coffee or water left, the *beep* might occur after exactly two seconds, indicating that the *refunding* process has started and the coin will be returned within four seconds. After successfully heating up, the machine grains the coffee for one or two seconds, before it starts the brewing process and finally releases the *coffee* after one

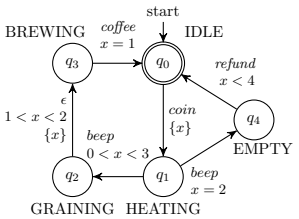

Fig. 1. Running example

[1] In model-based testing, strong responsiveness is the requirement that there are no silent loops, otherwise the tester cannot distinguish between deadlocks and livelocks.

second of brewing. There is no observable signal indicating the transition from graining to brewing, thus this transition is silent.

The rest of the paper is structured as follows: First, we will give the basic definitions and notation of TA with silent transitions (Section 2). Then, we will illustrate the first step of our procedure, the bounded-unfolding of the automaton and the renaming of clocks (Section 3). This will be followed by the second step, the removal of silent transitions (Section 4) and the final step, our determinization approach (Section 5). Section 6 summarizes the complexity of the different steps. In Section 7 we evaluate our prototype implementation and in Section 8 we address related work. Finally, in Section 9 we conclude our work. Complete proofs of the propositions and theorems can be found in the Appendix.

2 Timed Automata with Silent Transitions

We start with some definitions and notation. A timed automaton is an abstract model aiming at capturing the temporal behaviour of systems. It is a finite automaton extended with a set of clocks defined over $\mathbb{R}_{\geq 0}$, the set of non-negative real numbers. We may represent the timed automaton by a graph whose nodes are called *locations*, which are defined through a set of upper bounds put on the clock values. These bounds are restricted to non-negative integer values, which make decidability and computational issues easier and simpler to handle. While being at a location, all clocks progress at the same rate. The edges of the graph are called *transitions*. Each transition may be subject to some constraints, called *guards*, put on clock values in the form of integer inequalities. At each such transition an *action* occurs and some of the clocks may be reset. The actions take values in some finite domain denoted by Σ. Here we are dealing with the class of timed automata with an extended set of actions including also *silent actions*, denoted by ϵ. These are inner actions that are *non-observable* from the outside, to distinguish from the actions that are not silent and called *observable* actions. We call a TA without silent transitions *fully-observable*.

Let $\mathcal{X}$ be a finite set of *clock* variables. A clock *valuation* $v(x)$ is a function $v : \mathcal{X} \to \mathbb{R}_{\geq 0}$ assigning a real value to every clock $x \in \mathcal{X}$. We denote by $\mathcal{V}$ the set of all clock valuations and by $\mathbf{0}$ the valuation assigning 0 to every clock. For a valuation v and $d \in \mathbb{R}_{\geq 0}$ we define $v+d$ to be the valuation $(v+d)(x) = v(x)+d$ for all $x \in \mathcal{X}$. For a subset $\mathcal{X}_{rst}$ of $\mathcal{X}$, we denote by $v[\mathcal{X}_{rst}]$ the valuation such that for every $x \in \mathcal{X}_{rst}$, $v[\mathcal{X}_{rst}](x) = 0$ and for every $x \in \mathcal{X} \setminus \mathcal{X}_{rst}$, $v[\mathcal{X}_{rst}](x) = v(x)$. A *clock constraint* φ is a conjunction of predicates of the form $x \sim n$, where $x \in \mathcal{X}$, $n \in \mathbb{N}$ and $\sim \in \{<, \leq, =, \geq, >\}$. Given a clock valuation v, we write $v \models \varphi$ when v satisfies φ. We give now a formal definition of (non-deterministic) timed automata with silent transitions.

Definition 1 (eNTA). *A (non-deterministic) timed automaton with silent transitions A is a tuple $(\mathcal{Q}, q_{init}, \Sigma_\epsilon, \mathcal{X}, \mathcal{I}, \mathcal{G}, \mathcal{T}, \mathcal{Q}_{accept})$, where $\mathcal{Q}$ is a finite set of* locations *and $q_{init} \in \mathcal{Q}$ is the* initial *location; $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ is a finite set of actions, where Σ are the observable actions and ϵ represents a silent action, that is a non-observable internal action; $\mathcal{X}$ is a finite set of* clock *variables; $\mathcal{I} : L \to LI$*

is a mapping from locations to location invariants*, where each location invariant* $li \in LI$ *is a conjunction of constraints of the form* $true$, $x < n$ *or* $x \leq n$, *with* $x \in \mathcal{X}$ *and* $n \in \mathbb{N}$; $\mathcal{G}$ *is a set of* transition guards*, where each guard is a conjunction of constraints of the form* $x \sim n$, *where* $x \in \mathcal{X}$, $\sim \in \{<, \leq, =, \geq, >\}$ *and* $n \in \mathbb{N}$; $\mathcal{T} \subseteq \mathcal{Q} \times \Sigma_\epsilon \times \mathcal{G} \times \mathcal{P}(\mathcal{X}) \times \mathcal{Q}$ *is a finite set of* transitions *of the form* $(q, \alpha, g, \mathcal{X}_{rst}, q')$, *where* $q, q' \in \mathcal{Q}$ *are the* source *and the* target *locations;* $\alpha \in \Sigma_\epsilon$ *is the transition* action*;* $g \in \mathcal{G}$ *is the transition guard;* $\mathcal{X}_{rst} \subseteq \mathcal{X}$ *is the subset of clocks to be* reset*;* $\mathcal{Q}_{accept} \subseteq \mathcal{Q}$ *is the subset of* accepting *locations.*

Example 1. For the eNTA illustrated in Figure 1 we have $\mathcal{Q} = \{q_0, \ldots, q_4\}$, $q_{init} = q_0$, $\Sigma_\epsilon = \{\epsilon, coin, beep, refund, coffee\}$, $\mathcal{X} = \{x\}$, $\mathcal{I}(q_i) = true | q_i \in \mathcal{Q}$, $\mathcal{G} = \{0 < x < 3, x = 2, x < 4, 1 < x < 2, x = 1\}$, $\mathcal{Q}_{accept} = \{q_0\}$. $\mathcal{T}$ is the set containing all transitions, e.g. the transition from q_2 to q_3, with $\alpha = \epsilon$ (thus, it is a silent transition), $g = 1 < x < 2$ and $\mathcal{X}_{rst} = \{x\}$.

The *semantics* of an eNTA A is given by the *timed transition system* $[[A]] = (S, s_{init}, \mathbb{R}_{\geq 0}, \Sigma_\epsilon, T, S_{accept})$, where $S = \{(q, v) \in \mathcal{Q} \times \mathcal{V} \mid v \models \mathcal{I}(q)\}$; $s_{init} = (q_{init}, \mathbf{0})$; $T \subseteq S \times (\Sigma_\epsilon \cup \mathbb{R}_{\geq 0}) \times S$ is the transition relation consisting of *timed* and *discrete* transitions such that: *Timed transitions (delay):* $((q, v), d, (q, v+d)) \in T$, where $d \in \mathbb{R}_{\geq 0}$, if $v + d \models \mathcal{I}(q)$; *Discrete transitions (jump):* $((q, v), \alpha, (q', v')) \in T$, where $\alpha \in \Sigma$, if there exists a transition $(q, \alpha, g, \mathcal{X}_{rst}, q')$ in $\mathcal{T}$, such that: (1) $v \models g$; (2) $v' = v[\mathcal{X}_{rst}]$ and (3) $v' \models \mathcal{I}(q')$; $S_{accept} \subseteq S$ such that $(q, v) \in S_{accept}$ if and only if $q \in \mathcal{Q}_{accept}$.

A *finite well-behaving run* ρ of an eNTA A is a finite sequence of alternating timed and discrete transitions, that ends in an observable action, of the form $(q_0, v_0) \xrightarrow{d_1} (q_0, v_0 + d_1) \xrightarrow{\tau_1} (q_1, v_1) \xrightarrow{d_2} \cdots \xrightarrow{d_n} (q_{n-1}, v_{n-1} + d_n) \xrightarrow{\tau_n} (q_n, v_n)$, where $q_0 = q_{init}$, $v_0 = \mathbf{0}$, $\tau_i = (q_{i-1}, \alpha_i, g_i, \mathcal{X}_{rst(i)}, q_i) \in \mathcal{T}$ and $\alpha_i \in \Sigma$. In this paper we consider only finite and well-behaving runs. A run ρ is *accepting* if the last location q_n is accepting. The run ρ of A induces the *timed trace* $\sigma = (t_1, \alpha_1), (t_2, \alpha_2), \ldots, (t_n, \alpha_n)$ defined over Σ_ϵ, where $t_i = \Sigma_{j=1}^{i} d_i$. From the latter we can extract the *observable timed trace*, which is obtained by deleting from σ all the pairs containing silent actions. A TA is called *deterministic* if it does not contain silent transitions and whenever two timed traces are the same then they are induced by the same run. Otherwise, the TA is *non-deterministic*. The *language* accepted by an eNTA A, denoted $\mathfrak{L}(A)$, is the set of observable timed traces induced by all accepting runs of A. Note, that the restriction to well-behaving runs is compatible with the definition of the language of the automaton, where silent actions that occur after the last observable action on a finite run are ignored. As a consequence, a location with in-going edges consisting of only silent transitions cannot be an accepting location.

3 *k*-Bounded Unfolding of Timed Automata

Given an eNTA A which is strongly responsive, its k-prefix language $\mathfrak{L}_k(A) \subseteq \mathfrak{L}(A)$ is the set of observable timed traces induced by all accepting runs of A

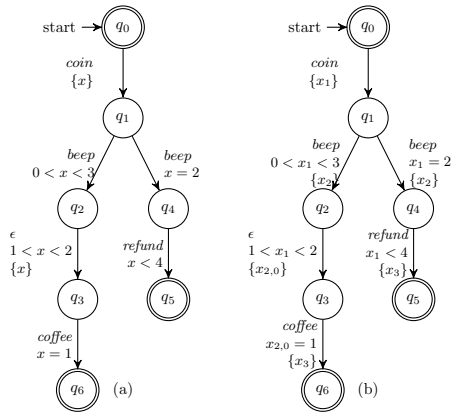

Fig. 2. Unfolding and clock renaming

which are of observable length bounded by k. That is,

$$\mathfrak{L}_k(A) = \{w \in \mathfrak{L}(A) \,|\, |w| \leq k\}. \quad (1)$$

By unfolding A and cutting it at observable level k, the resulting TA, $U_k(A)$, satisfies

$$\mathfrak{L}(U_k(A)) = \mathfrak{L}_k(A). \quad (2)$$

$U_k(A)$ is in the form of a finite tree, where each path that starts at the root ends after at most k observable transitions, and we may also further cut A by requiring that all leaves are accepting locations. Note, that if we reach in $U_k(A)$ a copy of an accepting location q of A by a silent transition then it will not be marked as an accepting location (but another copy might be marked as an accepting location if reached by an observable transition).

Figure 2(a) shows the unfolding of the coffee-machine up to observable depth three. The left branch is longer than the right, as it contains a silent transition.

3.1 Renaming the Clocks

Every unfolded timed automaton can be expressed by an equivalent timed automaton that resets at most one clock per transition. This known normal form [4] crucially simplifies the next stages of our algorithm, where we do not need to bother with multiple clock resets in one transition. The basic idea is to substitute the clocks from the original automaton by new clocks, where multiple old clocks reset at the same transition are replaced by the the same new clock, as they measure the same time until they are reset again. The substitution of the clocks works straight forward: At each path from the root, at the i-th observable transition, a new clock x_i is introduced and reset, and if this transition is followed by $l > 0$ silent transitions then new clocks $x_{i,0}, \ldots, x_{i,l-1}$ are introduced and reset. A clock x that occurs in a guard is substituted by the new clock that was introduced in the transition where the last reset of x happened, or by x_0 if it was never reset. Let τ_i and τ_j be two transitions on the same path in the original automata at observable depth i, j, s.t. $i < j$. Furthermore, a clock x appearing in the guard of τ_j, is reset before in τ_i, but is not reset on any transition in between τ_i and τ_j. Then, x_i is introduced and reset at τ_i and the original clock variable x is substituted by x_i in the guard of τ_j. Figure 2(b) illustrates the clock renaming applied to the coffee machine. In the guards of the two *beep*-transitions starting at q_1, x is replaced by x_1, since the last reset of x in the original automata was at depth one, while in the *coffee*-transition from q_3 it is replaced by $x_{2,0}$, as x was reset in the first silent transition after depth two.

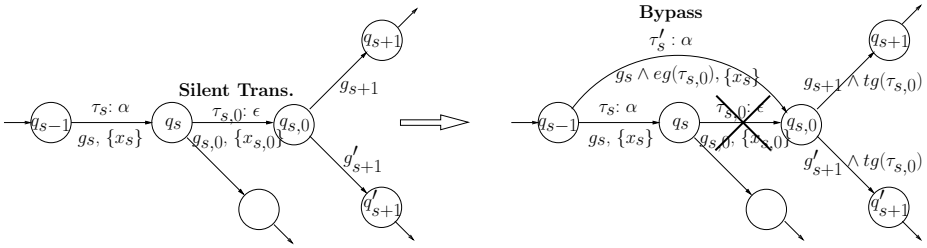

Fig. 3. Bypassing the silent transition

4 Removing the Silent Transitions

In this section we give an algorithm that removes the silent transitions from the eNTA A, which is in the form of a finite tree with renamed clocks. Thus, at each level i there will be a single clock x_i reset on all transitions of that level. Algorithm 1 shows the workflow and Figure 3 illustrates the general idea.

Algorithm 1 Removing the Silent Transitions

Input: $A \in \text{eNTA}_k$ in the form of a tree of observable depth k with renamed clocks
Output: $O(A) \in \text{NTA}_k$, such that $\mathfrak{L}(O(A)) = \mathfrak{L}(A)$

1: **while** there are silent transitions **do**
2: FIND first (from root) silent transition $\tau_{s,0}$ from q_s to $q_{s,0}$
3: SET lower bound to the silent transition
4: CREATE bypass transition with enabling guard
5: AUGMENT transitions from $q_{s,0}$ with taken guard
6: UPDATE guards on paths from $q_{s,0}$
7: REMOVE $\tau_{s,0}$
8: **end while**

We remove the silent transitions one at a time, where at each iteration we remove the first occurrence of a silent transition on some path from the root, until no silent transitions are left. So, let $\tau_{s,0}$ be such a first silent transition found by Line 2 of the algorithm, leading from location q_s to location $q_{s,0}$ with guard $g_{s,0}$ and reset of clock $x_{s,0}$. Let q_s be reached from location q_{s-1} with an observable transition τ_s and with guard g_s. The case where q_s is the initial location is simpler, as it does not require building a bypass transition. In order to remove the silent transition $\tau_{s,0}$ after forming a transition that bypasses it, several steps are carried out, that will be explained in detail in the following subsections. First, we set an auxilliary lower bound on the clock that is reset on the silent transition by updating the guard (Line 3). Then, we create the bypass transition using an *enabling guard* $eg(\tau_{s,0})$ which represents the upper bound until when the silent transition $\tau_{s,0}$ is enabled (Line 4). In Line 5 we construct a *taken guard* $tg(\tau_{s,0})$ that ensures that the transitions from $q_{s,0}$ come after the necessary delay that is forced by the silent transition. The taken guard is added to all transitions leaving $q_{s,0}$. Finally, in Lines 6–7, we remove the silent transition $\tau_{s,0}$ and update all future guards referring to the deleted clock $x_{s,0}$.

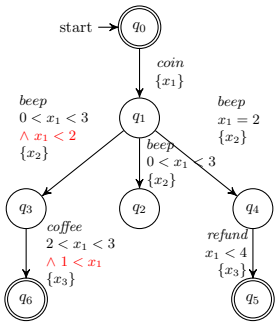

Fig. 4. Fully observable non-deterministic TA

Set a Lower Bound to the Silent Transition. We set a lower bound to the silent transition by augmenting the guard $g_{s,0}$ of $\tau_{s,0}$ to be $g'_{s,0} = g_{s,0} \wedge (0 \leq x_s)$, where x_s is the clock that is reset on the transition τ_s that precedes the silent transition. This additional constraint per definition always evaluates to *true*, but it is used in the next step to compute the unary constraints of the enabling guard. The guard of the silent transition in Figure 2 (b) after setting the lower bound is $1 < x_1 < 2 \wedge 0 \leq x_2$.

Creating a Bypass with the Enabling Guard. The enabling guard $eg(\tau_{s,0})$ guarantees that each clock's constraint that was part of the silent transition is satisfied at some non-negative delay and that these constraints are satisfied simultaneously, thus at some point during the bypass transition the silent transition would have been enabled as well. We describe here how the enabling guards are defined for strict inequalities, as shown in the upper part of Table 1. The other cases are dealt similarly, as seen in the table, and the constraint $x_i = n_i$ is treated as $n_i \leq x_i \leq n_i$. For every pair of a lower bound constraint $m_i < x_i$ and an upper bound constraint $x_j < n_j$, where $i \neq j$ and $x_i, x_j \neq x_s$ (x_s is the clock that is reset at τ_s), that appear in $g'_{s,0}$ we form the enabling guard binary constraint $x_j - x_i < n_j - m_i$ as shown in the first line of Table 1.

The next two lines consider constraints that involve the clock x_s, where x_s will be removed as it is the clock that will be reset on the bypass and is considered of value 0. Note, that for each upper bound constraint $x_j < n_j$ we use the lower bound constraint $0 \leq x_s$ that was added in the previous step of the algorithm to compute the enabling guard unary constraint $x_j < n_j$, which guarantees that at the time of the bypass x_j does not pass its upper bound constraint of the silent transition. An example of such a unary constraint is marked in red in the transition from q_1 to q_3 in Figure 4. The silent transition in the original automata could not have been enabled if x_1 had already been higher than two after the *beep*-transition, thus the bypass can also only be enabled while x_1 is smaller than two. The running example does not contain any binary constraints.

To create the bypass, we split the paths through q_s in the original automaton A into two. Those that do not take the silent transition $\tau_{s,0}$ continue as before from q_{s-1} to q_s and then to some location different from $q_{s,0}$. The paths that went through $\tau_{s,0}$ are directed from q_{s-1} to $q_{s,0}$ and then continue as before. The bypass τ'_s from q_{s-1} to $q_{s,0}$ has the same observable actions as those of τ_s, the same new clock reset x_s, and the guard g'_s which is the guard g_s of τ_s augmented with the *enabling guard* $eg(\tau_{s,0})$ (see Figure 3). Figure 4 shows the removal of the silent transition illustrated on the coffee-machine. The transition from q_1 to q_3 is the bypass and the transition from q_1 to q_2 is the original transition. Since the silent transition was the only transition leaving q_2, q_2 does not contain any outgoing transitions anymore, once the bypass is generated.

Silent Trans. Constraints	Clock Reset	Enabling Guard Constraint
$(m_i < x_i) \wedge (x_j < n_j)$	x_s	$x_j - x_i < n_j - m_i$
$(m_s < x_s) \wedge (x_j < n_j)$	x_s	$x_j < n_j - m_s$
$(m_i < x_i) \wedge (x_s < n_s)$	x_s	$m_i - n_s < x_i$
$(m_i \leq x_i) \wedge (x_j < n_j)$	x_s	$x_j - x_i < n_j - m_i$
$(m_i < x_i) \wedge (x_j \leq n_j)$	x_s	$x_j - x_i < n_j - m_i$
$(m_i \leq x_i) \wedge (x_j \leq n_j)$	x_s	$x_j - x_i \leq n_j - m_i$
$(m_i = x_i) \wedge (x_j = n_j)$	x_s	$x_j - x_i = n_j - m_i$

Table 1. Enabling guard constraints

Augmenting the Taken Guard. For each transition from $q_{s,0}$ to q_{s+1} we augment its guard g_{s+1} by forming $g'_{s+1} = g_{s+1} \wedge tg(\tau_{s,0})$ (see Figure 3), where $tg(\tau_{s,0})$ is the *taken guard*. $tg(\tau_{s,0})$ is composed of a single constraint: $0 \leq x_{s,0}$, where $x_{s,0}$ is the clock that is reset at the silent transition $\tau_{s,0}$. In the next stage of the algorithm of updating the future guards it will be transformed into the conjunction of the lower bound constraints $m_i < x_i$ or $m_i \leq x_i$ that appear in $g'_{s,0}$. These constraints make sure that we spend enough time at $q_{s,0}$ before moving to the next locations, as if we had taken the silent transition. The constraint is also used for synchronization of the future guards in the next step. In Figure 4, the red-marked part of the guard from transition q_3 to q_6 shows the taken guard that has already been updated from $0 \leq x_{2,0}$ to $1 < x_1$.

Updating the Future Guards. The removal of the silent transition $\tau_{s,0}$ enforces updating of the guards in the paths that start at $q_{s,0}$ and that refer to the clock $x_{s,0}$, that is reset on the silent transition. The most simple case is when the the silent transition guard $g'_{s,0}$ contains an exact constraint $x_i = n_i$, because then any future constraint of the form $x_{s,0} \sim l$ can be replaced by $x_i \sim n_i + l$. So, let us assume that the silent transition does not contain an exact constraint. The rules for updating the future guards are summarized in Table 2. Note, that an equality constraint $x_{s,0} = n_{s+j}$ in a future guard may be treated as $n_{s+j} \leq x_{s,0} \leq n_{s+j}$.

Let $g_{s+1}, \ldots, g_{s+p}$ be the ordered list of guards of consecutive transitions $\tau_{s+1}, \ldots, \tau_{s+p}$ along a path that starts at $q_{s,0}$. Then, if g_{s+j} contains the constraint $m_{s+j} < x_{s,0}$, it is replaced by the conjunction of constraints $m_i + m_{s+j} < x_i$, for each constraint $m_i < x_i$ that appear in $g'_{s,0}$. Similarly, for upper bound

Silent Trans. Constr.	Future Constr.	Replaced Constr.
$m_i < x_i$, $\{x_{s,0}\}$	$m_{s+j} < x_{s,0}$ or $m_{s+j} \leq x_{s,0}$	$m_i + m_{s+j} < x_i$
$m_i \leq x_i$, $\{x_{s,0}\}$	$m_{s+j} < x_{s,0}$	$m_i + m_{s+j} < x_i$
$m_i \leq x_i$, $\{x_{s,0}\}$	$m_{s+j} \leq x_{s,0}$	$m_i + m_{s+j} \leq x_i$
$x_i < n_i$, $\{x_{s,0}\}$	$x_{s,0} < n_{s+j}$ or $x_{s,0} \leq n_{s+j}$	$x_i < n_i + n_{s+j}$
$x_i \leq n_i$, $\{x_{s,0}\}$	$x_{s,0} < n_{s+j}$	$x_i < n_i + n_{s+j}$
$x_i \leq n_i$, $\{x_{s,0}\}$	$x_{s,0} \leq n_{s+j}$	$x_i \leq n_i + n_{s+j}$
$x_i = n_i$, $\{x_{s,0}\}$	$x_{s,0} \sim n_{s+j}$	$x_i \sim n_i + n_{s+j}$

Table 2. Update rules for future guards after removing the silent transitions

Constr. of g_{s+j}	**Constr. of g_{s+i}, $\{x_{s+i}\}$, $i < j$**	**Sync. Constr. of g_{s+j}**
$m_{s+j} < x_{s,0}$	$x_{s,0} < n_{s+i}$ or $x_{s,0} \leq n_{s+i}$	$m_{s+j} - n_{s+i} < x_{s+i}$
$m_{s+j} \leq x_{s,0}$	$x_{s,0} < n_{s+i}$	$m_{s+j} - n_{s+i} < x_{s+i}$
$m_{s+j} \leq x_{s,0}$	$x_{s,0} \leq n_{s+i}$	$m_{s+j} - n_{s+i} \leq x_{s+i}$
$x_{s,0} < n_{s+j}$	$m_{s+i} < x_{s,0}$ or $k_i \leq x_{s,0}$	$x_{s+i} < n_{s+j} - m_{s+i}$
$x_{s,0} \leq n_{s+j}$	$m_{s+i} < x_{s,0}$	$x_{s+i} < n_{s+j} - m_{s+i}$
$x_{s,0} \leq n_{s+j}$	$m_{s+i} \leq x_{s,0}$	$x_{s+i} \leq n_{s+j} - m_{s+i}$
$x_{s,0} = n_{s+j}$	$x_{s,0} = n_{s+i}$	$x_{s+i} = n_{s+j} - n_{s+i}$

Table 3. Synchronization constraints for future guards after removing silent transitions

constraints. In Figure 4, one future guard was updated in the transition from q_3 to q_6: The original guard of this transition was $x_{2,0} = 1$ (where $x_{2,0}$ was reset on the silent transition) and the guard of the silent transition was $1 < x_1 < 2$. Thus, according to the update rules, the updated future guard is $2 < x_1 < 3$ (written in black), conjuncted with the taken guard (marked in red).

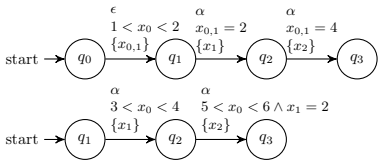

Fig. 5. Guard synchronization

These rules ensure that each future constraint on the clock $x_{s,0}$ separately conforms to and does not deviate from the possible time range of the silent transition. Yet, we need to satisfy a second condition: that along each path that starts at $q_{s,0}$ these future occurrences of $x_{s,0}$ are synchronized. This is achieved by augmenting the future guards with constraints of the form that appear in Table 3. No transition in our running example needs synchronization, hence we use a different example: the upper automaton in Figure 5 shows one silent transition followed by two observable transitions. Using only the previous update rules when removing the silent transition, the first observable transition might occur between three and four seconds, and the second one between five and six seconds. If the first transition occurs after three seconds and the second one after six, this would not conform to the original automaton which required exactly two seconds between them. Thus, applying the last synchronization rule of Table 3, the constraint $x_1 = 4 - 2$ is conjuncted to the second guard. The lower automaton in Figure 5 illustrates the synchronization. Note, we do not need a bypass transition here, since the silent transition starts in the initial state.

Removing the Silent Transition. Finally, we can safely remove the silent transition $\tau_{s,0}$ from q_s to $q_{s,0}$ after forming the bypass from q_{s-1} to $q_{s,0}$ with the necessary modifications to the transition guards.

Theorem 1 (Silent Transitions Removal). $\mathfrak{L}(O(A)) = \mathfrak{L}(A)$.

5 Determinization

Existing determinization algorithms (as e.g. applied in [15]) create the powerset of all transitions to be determinized, and build one transition for each subset in the powerset. We propose an alternative approach, that reduces the amount of

locations and transitions in the deterministic automata, by shifting some complexity towards the guards. Our motivation is the use of SMT solvers for verifying the timed automata models. The larger guards can by directly converted into SMT-LIB formulas, and thus should not pose a problem.

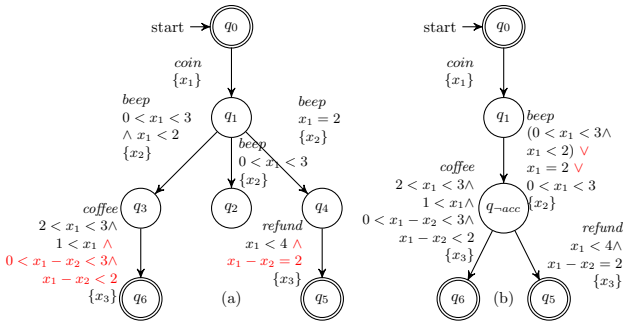

Fig. 6. (a) Modified guards added to future transitions (b) determinization via disjunction

The approach works under the following prerequisites: After the removal of the silent transitions the timed automaton A is in the form of a tree of depth k. At each level i the same new clock x_i is reset on each of the transitions of that level. This is the only clock reset on this level, and no clock is ever reset again.

The basic idea behind the determinization algorithm is to merge all transitions of the same source location and the same action via disjunction, and to push the decision which of them was actually taken to the following transitions. The postponed decision which transition was actually taken can be solved later on by forming diagonal constraints (as in zones) that are invariants of the time progress, and are conjuncted to immediately following transitions.

A pseudo-code description is given in Algorithm 2. The determinization is done in several steps applied to every location q with multiple outgoing transitions with the same action (Line 5), starting at the initial location (Line 1). Let q_i be such a location with multiple α transitions (Line 8). First, we add an accepting and a non-accepting location q_{acc}, $q_{\neg acc}$ replacing the target locations of the multiple α transitions (Line 7). Then, for each τ_i in the α transitions with guard g from q_i to q_{i+1}, let g' be the result of subtracting the clock x_{i+1} that is reset on τ_i from all clocks that appear in g (Lines 9-12). Next, g' is conjuncted to the guards of each transition τ_{i+1} that follows τ_i and the source location of τ_{i+1} is set to either q_{acc} or $q_{\neg acc}$, depending on whether q_{i+1} is accepting or not. Transitions leaving $q_{\neg acc}$ are additionally copied to q_{acc}, in case the guards of α transitions overlap. (Lines 14,16). Note that g' evaluates to *true* in every branch below τ_i if τ_i was enabled, thus the conjunction does not change the language of the automaton. Figure 6(a) illustrates the conjunction of the modified guards on our running example, marked in red. Note that the determinization did not involve any accepting locations, thus there was no splitting into q_{acc} and $q_{\neg acc}$. Next, all the α-transitions from q leading to accepting locations are merged into a transition leading to q_{acc} (Line 24) and all others into a transition leading to $q_{\neg acc}$(Line 25), by disjuncting their guards (Lines 20,21). The guard of the transition leading to $q_{\neg acc}$ is conjuncted to the negation of the other guard, to ensure determinism (Line 25). Finally, all merged τ_i and their target locations can be removed (Line 22). Figure 6(b) shows the determinized coffee-machine.

Algorithm 2 Guard-Oriented Determinization

Input: $A \in \text{NTA}_k$ in the form of a tree of depth k with renamed clocks
Output: $D(A) \in \text{TA}_k$, such that $\mathfrak{L}(D(A)) = \mathfrak{L}(A)$

$P \leftarrow \{(\mathcal{Q}_{init}, 0)\}$
while $P \neq \emptyset$ **do**
  PICK $(q_i, i) \in P$; $P \leftarrow P \backslash (q_i, i)$
  for each $\alpha \in \Sigma$ **do**
    if $\exists\ \tau_1(q_i, \alpha, g_1, \{x_{i+1}\}, q_1) \neq \tau_2(q_i, \alpha, g_2, \{x_{i+1}\}, q_2)$ **then**
      $g_{acc} \leftarrow false$; $g_{\neg acc} \leftarrow false$
      ADD new locations q_{acc}, $q_{\neg acc}$
      for each transition $\tau_i(q_i, \alpha, g_{i+1}, \{x_{i+1}\}, q_{i+1})$ **do**
        $g' \leftarrow g_{i+1}$
        for each clock x_j in g_{i+1} **do**
          $g' \leftarrow g'[x_j := x_j - x_{i+1}]$
        end for
        for each transition $\tau_{i+1}(q_{i+1}, \beta, g_{i+2}, \{x_{i+2}\}, q_{i+2})$ **do**
          ADD $\tau_{acc}(q_{acc}, \beta, (g_{i+2} \wedge g'), \{x_{i+2}\}, q_{i+2})$
          if $\neg$ accepting(q_{i+1}) **then**
            ADD $\tau_{\neg acc}(q_{\neg acc}, \beta, (g_{i+2} \wedge g'), \{x_{i+2}\}, q_{i+2})$
          end if
          REMOVE τ_{i+1}
        end for
        if $accepting(q_{i+1})$ **then** $g_{acc} \leftarrow g_{acc} \vee g_{i+1}$ **end if**
        if $\neg accepting(q_{i+1})$ **then** $g_{\neg acc} \leftarrow g_{\neg acc} \vee g_{i+1}$ **end if**
        REMOVE τ_i and q_{i+1}
      end for
      ADD transition $\tau_{acc}(q_i, \alpha, g_{acc}, \{x_{i+1}\}, q_{acc})$
      ADD transition $\tau_{\neg acc}(q_i, \alpha, (g_{\neg acc} \wedge \neg g_{acc}), \{x_{i+1}\}, q_{\neg acc})$
    end if
  end for
  for each transition $\tau_i(q_i, \alpha, g_{i+1}, \{x_{i+1}\}, q_{i+1})$ **do**
    $P \leftarrow P \cup (q_{i+1}, i+1)$
  end for
end while

Theorem 2 (Determinization). *The determinization algorithm constructs a deterministic timed automaton $D(A)$ such that $\mathfrak{L}(D(A)) = \mathfrak{L}(A)$.*

6 Complexity

Bounded Unfolding. We unfold the timed automaton A into a tree and cut it when reaching observable level k. Let us assume that the tree is of depth K, $K \geq k$, and of size $N = O(d^K)$, with $d \geq 1$ representing the approximate out-degree of the vertices in the graph of A. Since the analysis of the SMT solvers for different applications requires the exploration of all the transitions in the unfolded graph of A, the unfolding stage of our algorithm does not necessarily increase the overall time complexity of the algorithm.

Removing Silent Transitions. Our algorithm does not increase the size of the tree since we only substitute the silent transitions by the bypass transitions. We do add, however, constraints. The number of enabling-guard constraints that we add to each bypass transition is of order $O(K^2)$. Each updated future constraint is of order $O(K)$ (including on-the-fly simplification, so that each clock has at most one lower and one upper bound), and each future transition may be updated at most $O(K)$ times. Hence, the updating step is also of order $O(K^2)$, and the complexity of the whole algorithm is $O(NK^2)$. Note, we do not need to transform the diagonal constraints introduced in the algorithm into unary constraints, nor do they introduce problems in the next algorithm of determinization.
Determinization decreases the size of the unfolded automaton, if non-determinism exists. The complexity gain can be exponential in the number of locations and transitions, but is lost by a proportional larger complexity in the guards.

7 Implementation and Experimental Results

The algorithms were implemented in Scala (Version 2.10.3) and integrated into the test-case generation tool MoMuT::TA[2], providing a significant increase in the capabilities of the tool. MoMuT::TA provides model-based mutation testing algorithms for timed automata [2], using UPPAAL's [12] XML format as input and output. The determinization algorithm use the SMT-solver Z3 [9] for checking satisfiability of guards. All experiments were run on a MacBook Pro with a 2.53 GHz Intel Core 2 Duo Processor and 4 GB RAM.

The implementation is still a prototype and further optimizations are planned. One already implemented optimization is the "on-the-fly" execution of the presented algorithms, allowing the unrolling, clock renaming, silent transition removal and determinization in one single walk through the tree. The combined algorithm does not suffer from the full exponential blow-up of the unfolding: if the automata contains a location that can be reached via different traces, yet with the same clock resets, the unfolding splits it into several, separately processed, locations, while the on-the-fly algorithm only needs to process it once.

The following studies compare the numbers of locations and the runtimes of *a*) the silent transition removal, *b*) a standard determinization algorithm that works by splitting non-deterministic transitions into several transitions that contain each possible combination of their guards, *c*) the new determinization algorithm introduced in Section 5 and *d*) its on-the-fly version.
Study 1. The first example, taken from Diekert et al. [10], is the timed automaton illustrated in Fig. 7 (a), which cannot be determinized. We then added another α-transition (Fig. 7 (b)), which causes non-determinism after removing the silent transition. The test results are shown in Table 4 (before and after modification). **Study 2.** The second example is taken from Baier et al. [4] and is illustrated in Fig. 7(c). We modified the automaton by adding a silent transition (Fig. 7(d)). Table 5 shows the results of the two determinization approaches.

[2] `https://momut.org/?page_id=355`

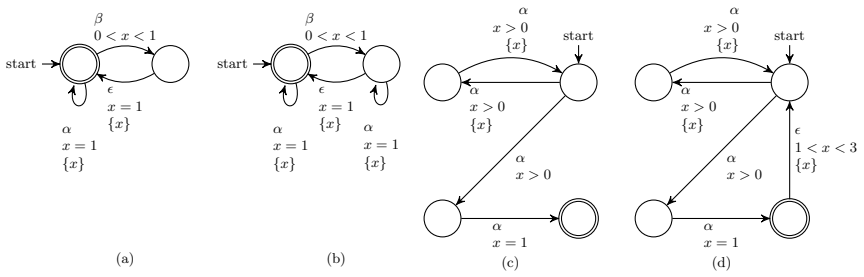

Fig. 7. The four timed automata used in Study 1 and Study 2

Study 3. This study is part of a model of an industrial application: it is based on a car alarm system that was already used as an example in our work on model-based mutation testing from timed automata (see [2] for the whole model). In this evaluation, we introduced a silent transition that adds a non-deterministic delay of up to two seconds before the timer of the alarm starts, and our results are given in Table 6. We were able to perform the removal of silent transitions and the guard-oriented determinization up to depth 12, and the location-oriented determinization up to depth 8. As expected, the studies confirm that the complexity of the different algorithms depends vastly on the input models. The tool and the examples are available[3].

8 Related Work

The first inspiration to our work was the paper of Bérard et al. [5], which shows that silent transitions extend the expressive power of TA and thus cannot be removed in general. The authors identify a class of eNTA that have the same expressive power as NTA, and propose a procedure for removing silent transitions.

The eNTA that are handled in [5] do not include loops containing observable silent transitions with clock reset. This obstacle disappears in the bounded setting, thus a larger class of TA can be handled. From the complexity point of view, our algorithm for removing silent transitions avoids splitting locations and

[3] `https://momut.org/?page_id=385`

Depth	Number of locations				Runtime (sec.)			
	unfolded	std. det.	new det.	on-the-fly	ϵ-removal	std. det.	new det.	on-the-fly
2	8	7	7	7	0.1	0.3	0.1	0.1
5	78	63	63	63	0.4	0.5	0.4	0.2
9	1,278	1,023	1,023	1,023	16,011.2	6.7	7.2	1.0
2	9	8	8	8	0.2	0.2	0.2	0.1
5	177	135	84	63	0.8	0.9	1.3	0.7
9	8,361	4,364	3,609	1,023	20,969.0	71.2	88.3	9.6

Table 4. Runtime and number of locations for the automata of Fig. 7 (a) (first three rows) and Fig. 7 (b) (last three rows)

Depth	Number of locations				Runtime (sec.)			
	unfolded	std. det.	new det.	on-the-fly	ϵ-removal	std. det.	new det.	on-the-fly
2	5	5	4	4	-	0.1	0.1	0.1
5	11	10	8	8	-	0.2	0.3	0.1
10	21	21	16	16	-	0.3	0.3	0.1
25	51	50	38	38	-	0.5	0.9	0.2
50	101	100	76	76	-	0.7	391.6	0.3
2	5	5	4	4	0.1	0.1	0.1	0.01
5	24	26	8	8	0.2	2.1	0.4	0.3
10	140	661	16	16	0.5	1,945.1	2.1	0.5

Table 5. Runtime and number of locations for the automata of Fig. 7 (c) (first three rows) and Fig. 7 (d) (last three rows)

removing diagonal constraints, as needed in [5]. These operations cause exponential blow-up (see [7], [8] for the advantage in keeping diagonal constraints).

The second major inspiration to our work was the paper of Baier et al. [4]. The authors first propose a procedure for translating NTA to *infinite* DTA trees, and then identify several classes of NTA that can be effectively determinized into finite DTA. In contrast to our work, the procedure proposed in [4] passes via the region graph, which makes it quite impractical for implementation. In addition, we also allow in our determinization procedure disjunctive constraints which results in a more succint representation. Both [5] and [4] study theoretical questions such as expressiveness and decidability. In contrast, we adapt the ideas from these papers and propose an effective procedure for the determinization of eNTA tailored to the bounded setting.

Wang et. al [15] use timed automata for language inclusion. Their procedure involves building a tree, renaming the clocks and determinization of the tree. Contrary to our work, they do not restrict themselves to the bounded setting, thus taking the risk that their algorithm does not terminate for some classes of timed automata. Also, they use the "standard" determinization method that involves splitting non-deterministic transitions into a possibly far larger set of deterministic transitions, whereas we join them into one transition.

Depth	Number of locations				Runtime (sec.)			
	unfolded	std. det.	new det.	on-the-fly	ϵ-removal	std. det.	new det.	on-the-fly
2	8	8	8	8	0.108	0.2	0.1	0.0
5	153	139	83	81	0.4	1.0	0.8	0.2
8	2,062	1,973	757	739	4.1	129.0	11.6	0.9
12	78,847	-	14,009	13,545	10,592.3	-	4,832.1	10.2

Table 6. Runtime and number of locations for the Car Alarm System [2], modified by adding a silent transition causing a 0-2 seconds delay.

9 Conclusion

The bounded setting allows the handling of a larger class of TA and in a more efficient way than in the unbounded setting. The extension from standard unary constraints to diagonal and disjuncive constraints has a practical reason: it is more efficient to let the SMT solvers deal with them than to translate them into standard form. In this paper a novel procedure was presented, which transforms bounded, non-deterministic and partially-observable TA into deterministic and fully-observable TA with diagonal and disjunctive constraints. The procedure includes an algorithm for removing the silent transitions and a determinization algorithm. It was implemented, tested and integrated into a model-based test generation tool. Recently [1] we investigated ways of pruning the determinized tree, to reduce the state space of the unfolding. These appoaches look promising for applying the presented work to test-case generation in industrial studies.

References

1. Bernhard K. Aichernig and Florian Lorber. Towards generation of adaptive test cases from partial models of determinized timed automata. In *AMOST*, 2015.
2. Bernhard K. Aichernig, Florian Lorber, and Dejan Nickovic. Time for mutants - model-based mutation testing with timed automata. In *TAP*, pages 20–38, 2013.
3. Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
4. Christel Baier, Nathalie Bertrand, Patricia Bouyer, and Thomas Brihaye. When are timed automata determinizable? In *ICALP*, pages 43–54, 2009.
5. Béatrice Bérard, Antoine Petit, Volker Diekert, and Paul Gastin. Characterization of the expressive power of silent transitions in timed automata. *Fundam. Inform.*, 36(2-3):145–182, 1998.
6. Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Advances in Computers*, 58:117–148, 2003.
7. Patricia Bouyer and Fabrice Chevalier. On conciseness of extensions of timed automata. *Journal of Automata, Lang. and Comb.*, 10(4):393–405, 2005.
8. Patricia Bouyer, François Laroussinie, and Pierre-Alain Reynier. Diagonal constraints in timed automata: Forward analysis of timed systems. In *FORMATS*, pages 112–126, 2005.
9. Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *TACAS*, pages 337–340, 2008.
10. Volker Diekert, Paul Gastin, and Antoine Petit. Removing epsilon-transitions in timed automata. In *STACS*, pages 583–594. Springer, 1997.
11. Olivier Finkel. Undecidable problems about timed automata. In *FORMATS*, pages 187–199, 2006.
12. Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *STTT*, 1(1-2):134–152, 1997.
13. Jan Tretmans. Test generation with inputs, outputs, and quiescence. In *TACAS*, pages 127–146, 1996.
14. Stavros Tripakis. Folk theorems on the determinization and minimization of timed automata. *Inf. Process. Lett.*, 99(6):222–226, 2006.
15. Ting Wang, Jun Sun, Yang Liu, Xinyu Wang, and Shanping Li. Are timed automata bad for a specification language? language inclusion checking for timed automata. In *TACAS*, pages 310–325, 2014.

10 Appendix A - Renaming of Clocks

The concrete algorithm used for renaming of the clocks is presented in pseudo-code in Algorithm 3. The original clocks are $x_0, \ldots, x_{n-1}$. Each new clock has either one index (l_1) in case the transition in which it is reset is observable, or two indices (l_1, l_2) in case of a silent transition. After the removal of the silent transitions stage we will be left with clocks with a single index and the same clock reset for the same level of the tree. The vector $X[0..n-1]$ holds the clock substitution list: $X[i]$ refers to the new clock that substitutes the original clock x_i. The the set of transition with source location q is denoted by $trans(q)$.

Algorithm 3 Renaming the Clocks

Input: $A \in \text{eNTA}_K$, a tree of depth K and observable depth k, clocks $\mathcal{X}$, $|\mathcal{X}| = n$
Output: $A \in \text{eNTA}_K$, clocks $\mathcal{X}'$, $|\mathcal{X}'| = K$, single clock reset per transition, same clock reset at same (observable, silent) level

$l_1 \leftarrow 0$ ▷ observable (primary) level
$l_2 \leftarrow -1$ ▷ silent (secondary) level
for $i \leftarrow 0, .., n-1$ **do**
    $X[i] \leftarrow x_0$ ▷ x_0 is reset at the initial location
end for
RenameClocks(q_0, X, l_1, l_2)
procedure RenameClocks(q, X, l_1, l_2)
    for each $\tau = (q, \alpha, g, \mathcal{X}_{rst}, q') \in trans(q)$ **do**
        for $i \leftarrow 0, .., n-1$ **do**
            $g \leftarrow g[x_i \leftarrow X[i]]$ ▷ renaming the clocks in the guard g
        end for
        if $\alpha = \epsilon$ **then** ▷ silent transition
            $l_2 \leftarrow l_2 + 1$
            $x \leftarrow x_{l_1, l_2}$ ▷ the new reset clock in case of a silent trans.
        else
            $l_1 \leftarrow l_1 + 1$
            $l_2 \leftarrow -1$
            $x \leftarrow x_{l_1}$ ▷ the new reset clock in case of an observable trans.
        end if
        for $i \leftarrow 0, .., n-1$ **do**
            if $x_i \in \mathcal{X}_{rst}$ **then**
                $X[i] \leftarrow x$ ▷ updating the clock substitution list
            end if
        end for
        $\mathcal{X}_{rst} \leftarrow \{x\}$ ▷ updating the reset clocks of τ
        if $l_1 < k$ **then**
            RenameClocks(q', X, l_1, l_2) ▷ recursive call with the target location
        end if
    end for
end procedure

11 Appendix - Proofs

11.1 Proof of Theorem 1 [Silent Transitions Removal]

Given a non-deterministic timed automaton with silent transitions A in the form of a finite tree, we need to show that our algorithm of removing the silent transitions results in an equivalent timed automaton, that is, $\mathfrak{L}(O(A)) = \mathfrak{L}(A)$. That is, we will show that if A' is the result of removing one first silent transition then A and A' are equivalent: for every timed trace of A there is an equivalent timed trace of A' and vice versa, in the sense that the corresponding observable timed traces are identical.

We claim that by induction the proof of equivalence for a single removal of a first silent transition suffices to prove the theorem. First, there are only finitely-many silent transitions in A. Secondly, the removal of a silent transition does not change the form of the guards at the part of the automaton that contains the remaining silent transitions: the introduction of diagonal constraints happens only at the enabling guard and so the algorithm for removal of the next silent transitions remains the same.

So, let $\tau_{s,0}$ be a first silent transition on a path γ that starts at the initial location. Let $\tau_{s,0}$ be from location q_s to location $q_{s,0}$, let q_{s-1} be the location that leads to q_s and let q_{s+1} be a location that follows $q_{s,0}$ on the path. Let A' be the automaton that results after removing τ and performing the steps as in Algorithm 1. Clearly, for every run that does not pass through $\tau_{s,0}$ there is an identical run in the other automaton. Thus, we restrict ourselves to runs though $\tau_{s,0}$.

$\boldsymbol{\mathfrak{L}(A) \subseteq \mathfrak{L}(A')}$**.** Let ρ be a run on A through γ. We need to show that there exists a run ρ' on A' with the same observable trace as of ρ. The run ρ' will go through the same locations and transitions as does ρ, except for the part q_{s-1}, τ_s, q_s, $\tau_{s,0}$, $q_{s,0}$ in A which will be replaced by the bypass q_{s-1}, τ'_s, $q_{s,0}$ in A' as in Fig. 3. The dates of the transitions will also be the same, except for the silent transition that is missing in ρ'. That is, if t_s, $t_{s,0}$ and t_{s+1} are the dates of ρ at the transitions τ_s, $\tau_{s,0}$ (the silent transition) and τ_{s+1} then the corresponding transitions of ρ' will take place at t_s (the time of the bypass) and t_{s+1}.

Since ρ goes through $\tau_{s,0}$, we know that by the time t_s after the reset of clock x_s the guard $g_{s,0}$ of $\tau_{s,0}$ is satisfied in some non-negative time. Thus, we know that each constraint of a clock x_j that appears in $g_{s,0}$ is satisfied at a non-negative delay, and that all these constraints can be satisfied simultaneously. So, first we need to show that the corresponding guard $g'_s = g_s \wedge eg(\tau_{s,0})$ of τ'_s in ρ' is satisfied at the same time, that is, that the enabled guard $eg(\tau_{s,0})$ is satisfied at t_s.

We will mostly restrict ourselves to strict inequalities, as the extension to the other cases (strict inequality versus weak inequality or weak inequality versus weak inequality) is straight forward.

For each clock x_j that is not reset at τ_s (that is, $j \neq s$) and that appears with an upper bound constraint $x_j < n_j$ (or $x_j \leq n_j$) at $g_{s,0}$ clearly the same

constraint holds also at the not-later time t_s. But that part is exactly what we have in $eg(\tau_{s,0})$ when comparing the upper bound constraint of x_j with the lower bound constraint of the reset clock x_s. Here the constraint in $eg(\tau_{s,0})$ is, in general, $x_j - x_s < n_j - m_s$, and since $m_s = 0$ and x_s is reset at τ'_s and replaced by 0 in the inequality the result is indeed $x_j < n_j$.

In addition to the above unary constraints, we know that each upper bound constraint on clock x_j in $g_{s,0}$ refers to a time which is of greater delay than the delay needed to reach each lower bound constraint on clock x_i in $g_{s,0}$, that is, $n_j - x_j > m_i - x_i$ at time $t_{s,0}$, otherwise these constraints couldn't have been satisfied simultaneously in ρ. But this is indeed the constraint $x_j - x_i < n_j - m_i$ that appears in $eg(\tau_{s,0})$.

We have seen that all the constraints of $eg(\tau_{s,0})$ are satisfied at time t_s and so the constraint g'_s of ρ' is satisfied at t_s and the transition τ'_s can be taken.

The next step is to show that the transition τ_{s+1} with guard g'_{s+1} of ρ' from location $q_{s,0}$ to location q_{s+1}, as well as the next transitions τ_{s+j}, $j = 2, \ldots, p$, with guards g'_{s+j} can be taken at the same dates t_{s+j} on which τ_{s+j} are taken in ρ on guards g_{s+j}, $j = 1, \ldots, p$.

If the silent transition happens to be on an exact time: $x_i = n_i$ then the update of the future guards that refer to the clock $x_{s,0}$ that was reset at $\tau_{s,0}$ is clear: each occurrence of $x_{s,0}$ is replaced by $x_i - n_i$, and we are done. So, suppose that there are no exact constraints at the silent transition.

For simplicity we will restrict ourselves mostly to strict inequalities and write the guard $g'_{s,0}$ of the silent transition $\tau_{s,0}$ as:

$$g'_{s,0} = 0 \leq x_s \wedge \bigwedge_{i=2,\ldots,r} m_i < x_i < n_i, \tag{3}$$

where for some of the clocks x_i there may be only a lower bound or only an upper bound constraint.

The constraints on $x_{s,0}$ at the transitions τ_{s+j}, $j = 1, \ldots, p$ contain $0 \leq x_{s,0}$ in τ_{s+1} and are of the general (strict inequalities) form $m_{s+j} < x_{s,0} < n_{s+j}$ in τ_{s+j}. The corresponding updated constraints of A' at time t_{s+j}, $j = 1, \ldots, p$, are

$$\bigwedge_{i=1,\ldots,r} m_i + m_{s+j} < x_i < n_i + n_{s+j}. \tag{4}$$

First, we need to show that the taken guard $tg(\tau_{s,0})$ is satisfied at time t_{s+1}. The taken guard is the constraint $0 \leq x_{s,0}$. After the update of the future guards this constraint is replaced by the conjunction of all the lower bound constraints $m_i < x_i$ of $g'_{s,0}$. But since these lower bound constraints are satisfied at the time $t_{s,0}$ of the silent transition (in ρ) then clearly they are satisfied at t_{s+1}, $t_{s+1} \geq t_{s,0}$, that is, the updated taken guard $tg(\tau_{s,0})$ is satisfied in ρ'.

Let us look at the other updated future constraints. Since at the time of the silent transition $x_{s,0} = 0$ and $m_i < x_i$ then at time t_{s+j} when $m_{s+j} < x_{s,0}$ we have $m_i + m_{s+j} < x_i$. With a similar argument for the upper bound constraints, we see that the constraints of (4) are satisfied in ρ'..

Also the part of the synchronization rules is clear since it refers to the possible minimum and maximum time difference between every two transitions on which $x_{s,0}$ occurs, and since the run ρ goes through these transitions it assures that these constraints can be satisfied. So, for example, the synchronization constraint $m_{s+j} - n_{s+i} < x_{s+i} < n_{s+j} - m_{s+i}$ that is added to the guard g_{s+j} of τ_{s+j}, refers to the time difference $t_{s+j} - t_{s+i}$ between the transition τ_{s+i} and the transition τ_{s+j}, $i < j$.

Note that the synchronization with the constraint $0 \leq x_{s,0}$ of τ_{s+1} results in adding to τ_{s+j}, $j = 1, \ldots, p$ the constraint $x_{s+1} < n_{s+j}$, that is $t_{s+j} - t_{s+1} < n_{s+j}$, which clearly is satisfied since $t_{s+j} - t_{s,0} < n_{s+j}$.

We showed that the observable trace of ρ' is the same as that of ρ and this completes the proof of $\mathfrak{L}(A) \subseteq \mathfrak{L}(A')$.

$\mathfrak{L}(A') \subseteq \mathfrak{L}(A)$. Let ρ' be a run on A' going through the bypass τ'_s. We will show that there exists a run ρ through $\tau_{s,0}$ in A with the same observable trace as of ρ'.

The first thing we need to check is that the silent transition $\tau_{s,0}$ can be taken, given that the enabling guard $eg(\tau_{s,0})$ is satisfied at time t_s. The unary constraints $x_j < n_j$ ($x_j \leq n_j$) of $eg(\tau_{s,0})$ guarantee that each of the constraints in the guard $g'_{s,0}$ of the silent transition $\tau_{s,0}$ can be satisfied separately at some time that is equal or is later than t_s. Then, in order that all the constraints could be satisfied simultaneously, it suffices to show that the minimum upon the time delays to the upper bound constraints of the clocks appearing in $g'_{s,0}$ is greater than the maximum upon the time delays to the lower bound constraints in $g'_{s,0}$ (the 'greater' should be replaced by 'greater or equal' in case both the maximum and minimum come from weak inequalities):

$$\min_j(n_j - x_j) > \max_i(m_i - x_i). \tag{5}$$

But this condition is equivalent to the condition that $n_j - x_j > m_i - x_i$ at time t_s for every i, j, which is exactly the conjunction of diagonal constraints

$$\bigwedge_{i \neq j} x_j - x_i < n_j - m_i \tag{6}$$

of $eg(\tau_{s,0})$.

Thus, we know that the silent transition $\tau_{s,0}$ can be taken in the run ρ at some time $t_{s,0}$ after a delay of $M = \max_i(m_i - x_i)$ from t_s (this delay is not negative since we introduced the constraint $0 \leq x_s$) and before a delay of $N = \min_j(n_j - x_j)$.

It remains to show that the transitions $\tau_{s+1}, \ldots, \tau_{s+p}$ on guards $g_{s+1}, \ldots, g_{s+p}$ of ρ can be taken at the same dates $t_{s+1}, \ldots, t_{s+p}$ as the corresponding transitions on guards $g'_{s+1}, \ldots, g'_{s+p}$ are taken in ρ'.

To be more specific, it suffices to prove that there exists $t_{s,0}$ with the following conditions:

1. $t_s \leq t_{s,0} \leq t_{s+1}$;

2. $g'_{s,0}$ is satisfied at $t_{s,0}$;
3. the constraints on $x_{s,0}$ are satisfied at $t_{s+1}, \ldots, t_{s+p}$, with $x_{s,0}$ reset at $t_{s,0}$.

For condition 2. the constraints of $g'_{s,0}$ that should be satisfied at time $t_{s,0}$ are

$$\bigwedge_{i=1,\ldots,r} m_i < x_i(t_{s,0}) < n_i. \tag{7}$$

Equivalently, at each time t_{s+j}, $j = 1, \ldots, p$:

$$\bigwedge_{i=1,\ldots,r} m_i + t_{s+j} - t_{s,0} < x_i(t_{s+j}) < n_i + t_{s+j} - t_{s,0}, \tag{8}$$

or,

$$\bigwedge_{i=1,\ldots,r} m_i - x_i(t_{s+j}) + t_{s+j} < t_{s,0} < n_i - x_i(t_{s+j}) + t_{s+j}. \tag{9}$$

For condition 3. the constraints on $x_{s,0}$ that should be satisfied at times $t_{s+1}, \ldots, t_{s+p}$ are $m_{s+j} < x_{s,0}(t_{s+j}) < n_{s+j}$ for $j = 1, \ldots, p$. The constraint here at time t_{s+1} is $0 \leq x_{s,0}(t_{s+1})$ possibly conjuncted with other constraints (for convenience we wrote all constraints as strict inequalities). This is equivalent to

$$\bigwedge_{j=1,\ldots,p} m_{s+j} < t_{s+j} - t_{s,0} < n_{s+j} \tag{10}$$

or

$$\bigwedge_{j=1,\ldots,p} -n_{s+j} + t_{s+j} < t_{s,0} < -m_{s+j} + t_{s+j}. \tag{11}$$

We need to show that the constraints on $t_{s,0}$ of (9) and (11) do not define an empty set. This condition is equivalent to showing that the set S_1 of the above expressions to the left of $t_{s,0}$ is smaller than the set S_2 of the expressions to the right of $t_{s,0}$ (equivalently that the maximum of S_1 is smaller than the minimum of S_2), where

$$S_1 = \{m_i - x_i(t_{s+j}) + t_{s+j} \mid i = 1, \ldots, r,\ j = 1, \ldots, p\} \cup \{-n_{s+j} + t_{s+j} \mid j = 1, \ldots, p\}, \tag{12}$$

and

$$S_2 = \{n_i - x_i(t_{s+j}) + t_{s+j} \mid i = 1, \ldots, r,\ j = 1, \ldots, p\} \cup \{-m_{s+j} + t_{s+j} \mid j = 1, \ldots, p\}. \tag{13}$$

There are two types of expressions in S_1 and two types of expressions in S_2, hence we need to check that the following 4 cases are satisfied.

Case 1: $\boldsymbol{m_i - x_i(t_{s+j}) + t_{s+j} < n_{i'} - x_{i'}(t_{s+j'}) + t_{s+j'}}$. This inequality is equivalent to

$$m_i - x_i(t_{s,0}) + t_{s,0} < n_{i'} - x_{i'}(t_{s,0}) + t_{s,0}, \tag{14}$$

or to

$$m_i - x_i(t_{s,0}) < n_{i'} - x_{i'}(t_{s,0}). \tag{15}$$

The latter is equivalent to

$$x_{i'}(t_s) - x_i(t_s) < n_{i'} - m_i, \tag{16}$$

which is (6), the enabling guard $eg(\tau_{s,0})$ that is satisfied at time t_s of the run ρ'.

Case 2: $m_i - x_i(t_{s+j}) + t_{s+j} < -m_{s+j'} + t_{s+j'}$. This inequality is equivalent to

$$m_i - x_i(t_{s+j'}) + t_{s+j'} < -m_{s+j'} + t_{s+j'}, \tag{17}$$

$$m_i - x_i(t_{s+j'}) < -m_{s+j'}, \tag{18}$$

$$m_i + m_{s+j'} < x_i(t_{s+j'}). \tag{19}$$

The last inequality is no other than one of the left inequalities of (4), which are the updated future constraints in A' of the reset clock $x_{s,0}$, and thus are given to be satisfied.

Case 3: $-n_{s+j'} + t_{s+j'} < n_i - x_i(t_{s+j}) + t_{s+j}$. This inequality is equivalent to

$$-n_{s+j'} + t_{s+j'} < n_i - x_i(t_{s+j'}) + t_{s+j'}, \tag{20}$$

$$-n_{s+j'} < n_i - x_i(t_{s+j'}), \tag{21}$$

$$x_i(t_{s+j'}) < n_i + n_{s+j'}. \tag{22}$$

But the last inequality is one of the right inequalities of (4), which are the updated future constraints in A' of the reset clock $x_{s,0}$, and thus are given to be satisfied.

Case 4: $-n_{s+i} + t_{s+i} < -m_{s+j} + t_{s+j}$. This inequality is equivalent to

$$m_{s+j} - n_{s+i} < t_{s+j} - t_{s+i}. \tag{23}$$

The inequality certainly holds when $i = j$. When $i < j$ we can write this inequality with the clock x_{s+i} that is reset at time t_{s+i} in A':

$$m_{s+j} - n_{s+i} < x_{s+i}(t_{s+j}). \tag{24}$$

But the last inequality can be found in the first row of Table 3 which contains the synchronization constraints of the updated future constraints in A' of the reset clock $x_{s,0}$.

Similarly, when $j < i$ we need to satisfy the inequality

$$x_{s+j}(t_{s+i}) = t_{s+i} - t_{s+j} < n_{s+i} - m_{s+j}, \tag{25}$$

which can be found in the forth row of Table 3.

We showed that the set of possible time values $t_{s,0}$ for the silent transition in ρ is not empty, that is, there is a solution to the set of inequalities (9) and (11)

in the indeterminate $t_{s,0}$ (again, the extension to weak inequalities is straight forward).

To complete the proof it remains to show that the solution for $t_{s,0}$ satisfies condition 1., that is that $t_s \leq t_{s,0} \leq t_{s+1}$. Well, the left inequality $t_s \leq t_{s,0}$ comes from satisfying the inequality $m_i - x_i(t_{s+j}) + t_{s+j} \leq t_{s,0}$ of (9) with $x_i = x_s$ and $m_i = m_s = 0$ (it refers to augmenting the silent transition guard with the constraint $0 \leq x_s$). This inequality is equivalent to $0 - x_s(t_s) + t_s \leq t_{s,0}$ or $t_s \leq t_{s,0}$ since x_s was reset at time t_s.

The right inequality comes from satisfying the inequality $t_{s,0} \leq -m_{s+1} + t_{s+1}$ of (11) with $m_{s+1} \geq 0$, that is, $t_{s,0} \leq t_{s+1}$.

11.2 Proof of Theorem 2 [Determinization]

The deterministic property of $D(A)$ follows from the fact that when merging α-transitions into τ_{acc} and $\tau_{\neg acc}$ then the guard of $\tau_{\neg acc}$ is a conjunction of some guard with the negation of the guard of τ_{acc}. Hence, different runs will induce different time traces.

In general, by merging locations of A in $D(A)$ we may only expand the language and conclude that $\mathfrak{L}(A) \subseteq \mathfrak{L}(D(A))$. On the other hand, the new constraints introduced in $D(A)$ may restrict the language. So, let us examine the new transformed constraints and show that they do not impose additional restrictions. Suppose the guard of transition τ contains the constraint $x \sim n$ and that y is reset on τ. Then, at the time t_0 of τ, the constraint $x(t_0) - y(t_0) \sim n$ holds. But also at time $t_1 > t_0$, the constraint $x(t_1) - y(t_1) \sim n$ holds since x and y progress at the same rate. Hence, for any run through τ in A there exists a corresponding run in $D(A)$ with the same trace because the additional constraints of the form $x - y \sim n$ that are added to the future guards are satisfied automatically by all runs in $D(A)$ that satisfy the guard of τ. Thus, it remains $\mathfrak{L}(A) \subseteq \mathfrak{L}(D(A))$.

To show that the language of $D(A)$ does not contain accepting traces that are not in the language of A it suffices to show that when a transition in a merged location of $D(A)$ is enabled then the corresponding original transition in A is enabled. But this is indeed the case since for each transition of $D(A)$ we first copy to its guard the transformed guard of the transition that leads to it, and this transformed guard contains all the history: the transformed guards of the path that leads to this transition. That is, by induction one shows that since the record of paths of level n are passed to paths of level $n+1$ then it holds for every level.