Minimizable Timed Automata*

Jan Springintveld and Frits Vaandrager

Computing Science Institute
University of Nijmegen
P.O. Box 9010, 6500 GL Nijmegen
The Netherlands
{jans,fvaan}@cs.kun.nl

Abstract. State minimization plays a fundamental role in both classical automata theory and in the theory of reactive systems. Many algorithms and results are based on the fact that for each finite automaton there exists an equivalent minimum state automaton that can be effectively computed and that is unique up to isomorphism.

Timed safety automata (TSA's) [5], finite automata with clocks, have been used extensively for the specification and verification of real-time systems. However, there does not always exist a unique minimum state TSA that is equivalent to a given TSA. This problem occurs irrespective of the selected notions of state (including or excluding clock values) and equivalence on states (language equivalence, bisimulation equivalence, etc.).

Henzinger, Kopke and Wong-Toi [4] convincingly showed that if states do not include clock values, state minimization for timed automata is neither useful nor interesting. In this paper, we discuss state minimization for states that do include clock values, i.e., at the semantic level, and work in bisimulation equivalence. In this setting, a timed automaton is minimal when there does not exist a pair of bisimilar but distinct states in the transition system induced by the timed automaton.

We present a new model of *minimizable timed automata (MTA's)*, a variant of the TSA model, and prove that

1. The MTA and TSA model are equally expressive in the sense that for each MTA there exists a bisimilar TSA and for each TSA there exists a bisimilar MTA.
2. For each MTA there exists a bisimilar minimal MTA that can be effectively computed and that is unique up to isomorphism.

1 Introduction

State minimization plays a fundamental role in both classical automata theory and in the theory of reactive systems. Many algorithms and results are based on the fact that for each finite automaton there exists an equivalent minimum state automaton that can be effectively computed and that is unique up to isomorphism. Timed safety automata (TSA's) [5], finite automata with clocks, have been used extensively for the specification and verification of real-time systems. Despite this success, TSA's suffer from drawbacks. One key problem is that there does not always exist a unique minimum state TSA that is equivalent to a given TSA. This problem occurs irrespective of the selected notions of state (including or excluding clock values) and equivalence on states (language equivalence, bisimulation equivalence, etc.). Henzinger, Kopke and Wong-Toi [4] convincingly showed that if states do not include clock values, state minimization for timed automata is neither useful nor interesting: if time steps of duration 0 are not allowed it is even possible to find for every TSA an equivalent (not uniquely determined) TSA with just one state.

In this paper, we discuss state minimization for states that do include clock values, i.e., at the semantic level, and work in bisimulation equivalence. (For the notion of bisimulation, consult, e.g.,

* Research supported by the Netherlands Organization for Scientific Research (NWO) under contract SION 612-33-006 and by the HCM network EXPRESS.

[8].) In this setting, a timed automaton is minimal when there does not exist a pair of bisimilar but distinct states in the transition system induced by the timed automaton.

We first present a series of examples of TSA's for which no equivalent minimum state TSA exists. This motivates the subsequent definition of our new model of *minimizable timed automata (MTA's)*, a variant of the TSA model. We prove that

1. The MTA and TSA model are equally expressive in the sense that for each MTA there exists a bisimilar TSA and for each TSA there exists a bisimilar MTA.
2. For each MTA there exists a bisimilar minimal MTA that can be effectively computed and that is unique up to isomorphism.

MTA's are defined in two stages. First we introduce timed automata with *bounded time domains* (BTDA's). The boundedness of time domains is itself essential for minimization and in addition makes it possible to introduce more general assignments to clocks, without altering the expressive power of the model. E.g., assignments of the form $x := y+2$ are allowed. Manipulating such general assignments will be a key technique in the minimization. An MTA is defined as a BTDA A together with a family of *relevance formulas*, one for each clock in A, determining when x is relevant (w.r.t. enabling of transitions). These formulas will make it possible to identify states that only differ w.r.t. irrelevant clocks.

Our main motivation for developing the MTA model is that we are currently involved in a project to generalize the classical theory of testing for finite automata [6] to a timed setting. Minimization plays such a central role in the untimed theory that we do not see how one can possibly generalize this to the timed setting without a corresponding notion of minimality. In the testing world, systems are usually assumed to be deterministic. Since it is well-known (see, for instance, [8]) that the linear-time branching time spectrum collapses for deterministic transition systems, this also motivates our choice to work in the setting of bisimulation equivalence: technically this seems to be the simplest equivalence to deal with and for our intended domain of application minimization modulo bisimulation is all we need. An interesting topic of future research will be whether the results of this paper can be generalized to the setting of trace equivalence. Since trace equivalence between timed transition systems is undecidable [3, 2], the construction of a minimal MTA will in any case not be effective.

Apart from being essential for the purpose of minimization, the MTA model provides a nice alternative representation of TSA's that offers insight in their behavior and that may be useful for the efficient implementation of verification procedures. E.g., we obtain for every location of the automaton the minimal dimension of the clock space of that location, in terms of the number of relevant clocks and the size of their domains. We expect that from this information also an estimation of the minimal number of clocks can be derived (see [7] for an algorithm to minimize the number of clocks).

To the best of our knowledge, this is the first paper in which minimization of timed systems is treated at the level of transition systems. The work on minimization of timed systems done in [1, 11] concentrated on minimization of the region graph. For testing timed systems and many other purposes, minimization of the region graph results in a structure that is too course, and the more fundamental operation of minimization of transition systems is required. In [10, 9, 2], bisimulations between timed automata are studied, but minimization up to bisimulation is not dealt with.

The paper is organized as follows. In Section 2, we present some examples that motivate the MTA model. BTDA's and their operational semantics are defined in Section 3. In Section 4, we prove that for every BTDA there exists a bisimilar TSA and in Section 5 we prove the converse. In Section 6, we introduce MTA's and show that they can indeed be minimized and have the same expressive power as BTDA's.

2 Motivating Examples

Timed safety automata are not minimizable for a variety of reasons. In this section we will discuss some examples to explain the problems. These examples also serve as motivation for our new model of minimizable timed automata. We assume the reader to be familiar with the model of timed safety automata (TSA's) as presented in [5]. In Section 3, the definition of TSA's will be recalled along with the definition of some new concepts.

Example 2.1. It is well-known that beyond a certain bound the actual values of clocks do not matter. In fact, this was one of the key insights of Alur and Dill [3] when they defined the region construction. Consider the TSA of Figure 1. This TSA is not minimal since (for instance), for all $t, t' > 2$, the

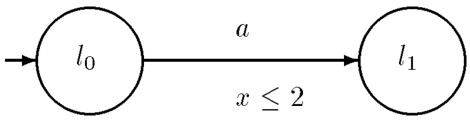

Fig. 1. The need for bounded clock domains.

states (l_0, t) and (l_0, t') are bisimilar. It is not difficult to see that in fact no minimal TSA can be equivalent to the TSA of Figure 1. Therefore, the clocks in our MTA model take values in a finite interval augmented with the single element ∞. This allows us, for instance, to give clock x in the TSA of Figure 1 domain $[0, 2] \cup \{\infty\}$. Beyond a certain point there is no need to record the specific value of a clock, and we only need to know that this value is *large*. *(End example.)*

Example 2.2. Consider the TSA of Figure 2. This TSA represents a switch that can be turned on

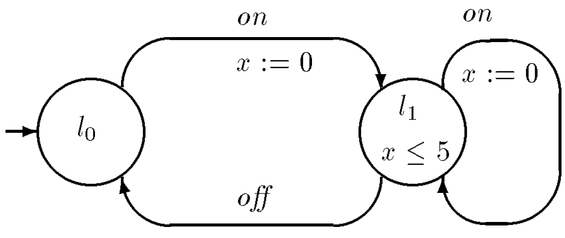

Fig. 2. Clocks are not always relevant.

at any time and switches off automatically 5 time units after the last time it has been turned on. The TSA is not minimal since, for all $t, t' \geq 0$, the states (l_0, t) and (l_0, t') are bisimilar: clock x only matters in location l_1, where it records the time that has elapsed since the previous *on*-event. Again, it is not difficult to prove that no minimal TSA can be equivalent to the TSA of Figure 2.

To deal with this situation, our new model allows one not to record the values of certain clocks in certain locations of the automaton. *(End example.)*

Example 2.3. In the TSA model only two types of assignments are allowed: *resets* of the form $x := 0$ and (implicit) identity assignments of the form $x := x$. More general assignments, such as $x := x + 1$ and $x := x - 1$, are not included in the TSA model for decidability reasons: adding such assignments would make it trivial to encode a two-counter machine and thus render reachability and model checking problems undecidable. The example of Figure 3, however, suggests that assignments that increment variables cannot be avoided if the goal is to minimize timed automata. It is easy to find similar examples that show the use of assignments that decrement variables or assignments of the form $x := n$ with $n \neq 0$. Therefore we decided to allow for such assignments in the MTA model. The main reason why this does not lead to undecidability is that in the MTA model the domains of the clock variables are bounded intervals extended with ∞. This boundedness makes it impossible to encode two-counter machines directly.

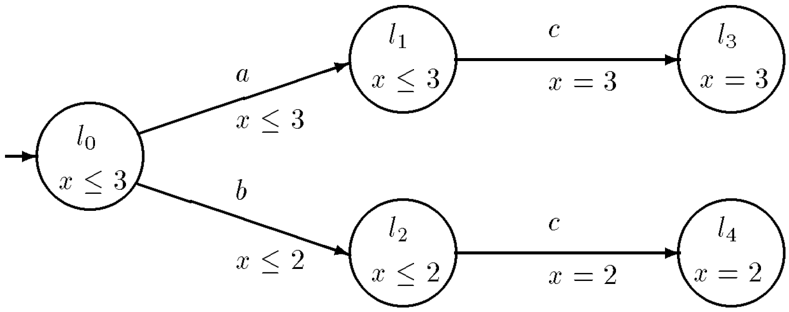

Fig. 3. The need for clock increments.

The TSA of Figure 3 is not minimal since, for all $t \in [1, 3]$, the states (l_1, t) and $(l_2, t-1)$ are bisimilar. Once again, no minimal TSA exists that is equivalent to the TSA of Figure 3.

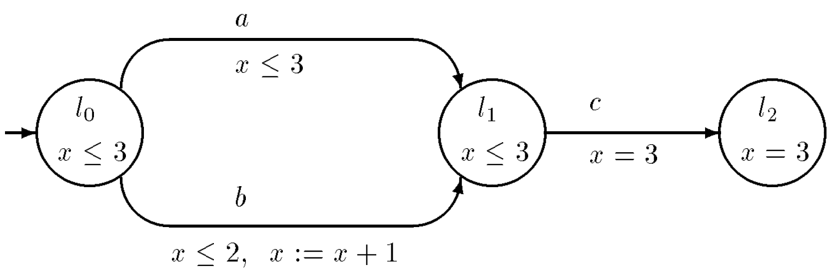

Fig. 4. Using clock increments in MTA's.

Figure 4 indicates how this TSA can be minimized in the MTA model. *(End example.)*

Example 2.4. In order to minimize automata, it is also quite useful to allow for assignments of the form $x := y$. In the TSA of Figure 5, the states $(l_3, x = t, y = t')$ and $(l_3, x = t', y = t)$ are bisimilar for all $t, t' \in [0, 1]$. Figure 6 shows how, by swapping the roles of clocks x and y for one of the incoming edges of l_3 and by strengthening the invariant of this location, this redundancy can be eliminated. *(End example.)*

Example 2.5. Our final example in this section illustrates how the value of one clock may become irrelevant when the value of another clock passes some boundary. In the TSA of Figure 7, the value of clock x in location l_1 becomes irrelevant as soon as clock y reaches a value larger than 1. As long as $y \leq 1$, a b transition is possible from location l_1 to location l_2. Since l_2 has an outgoing c transition that tests x, this means that the value of x is relevant in location l_1 as long as $y \leq 1$. However, as soon as $y > 1$, the b transition gets disabled.

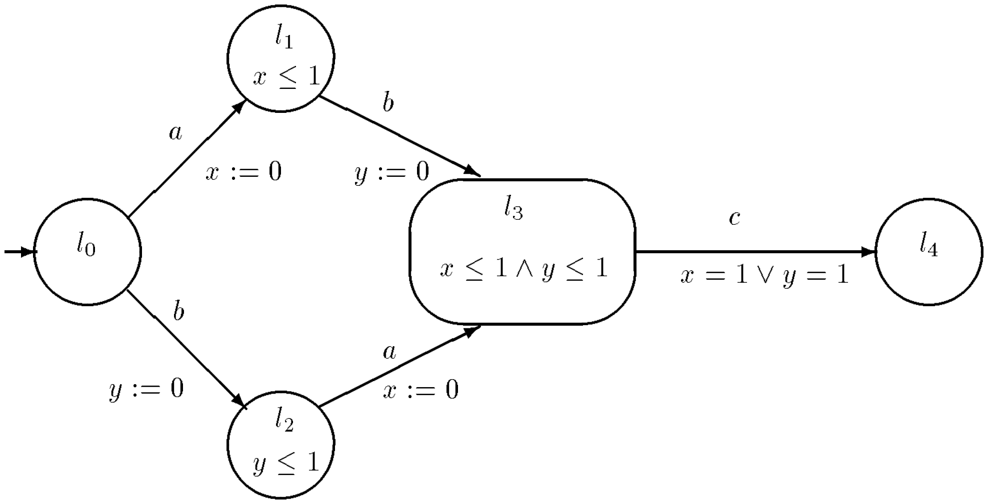

Fig. 5. The need for clock renaming.

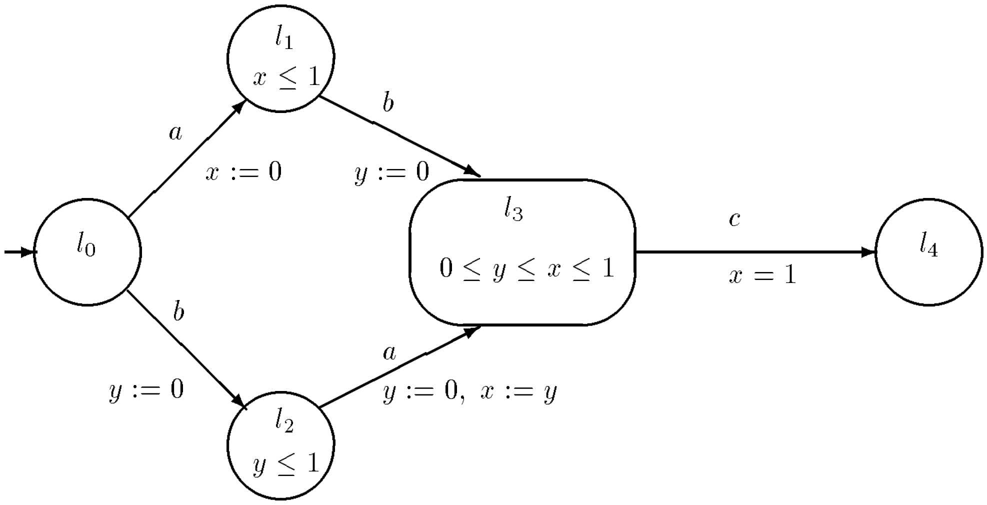

Fig. 6. The use of clock renaming.

Thus, for instance, states $(l_1, x = \frac{1}{2}, y = 1\frac{1}{2})$ and $(l_1, x = \frac{3}{4}, y = 1\frac{1}{2})$ are bisimilar, whereas the states $(l_1, x = \frac{1}{2}, y = \frac{3}{4})$ and $(l_1, x = \frac{3}{4}, y = \frac{3}{4})$ are not. It is not so difficult to prove that there exists no minimal TSA that is equivalent to the TSA of Figure 7. In order to deal with this type of situations, our MTA model incorporates so-called *relevance formulas* that allow one to specify, for each clock, where its value is relevant and should be recorded as part of the state. *(End example.)*

3 Timed Automata

3.1 The model

Let R denote the reals, $\mathsf{R}^{\geq 0}$ the nonnegative reals, and R^{∞} the reals together with the single element ∞. We extend the standard ordering $\leq$ and addition operator $+$ over R to R^{∞} in the usual way: for every $t \in \mathsf{R}^{\infty}$, $t \leq \infty$ and $t + \infty = \infty + t = \infty$. Let Z denote of integers and Z^{∞} the set $\mathsf{Z} \cup \{\infty\}$.

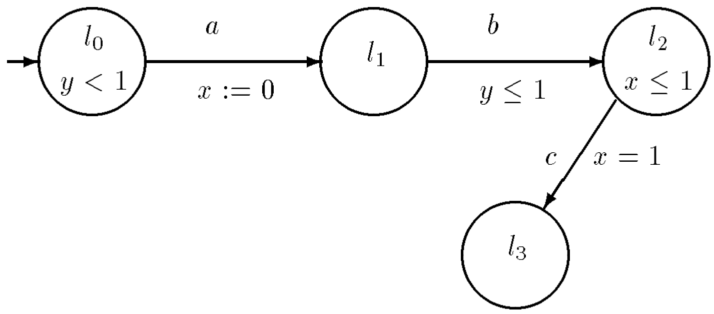

Fig. 7. Relevance of x depends on y.

Definition 3.1 *(Intervals).* An *interval* I is a (possibly empty) convex subset of R. An interval I is *bounded* if both $\sup I$ and $\inf I$ are in R. An *integer interval* is an interval I with both $\inf I$ and $\sup I$ in $\mathsf{Z} \cup \{-\infty, \infty\}$. By convention $\inf \emptyset = \sup \emptyset = 0$, which implies that the empty set is an integer interval. Note that there are countably many integer intervals.

Definition 3.2 *(Variables, domains and actions).* We assume a countable universe $\mathcal{C}$ of *clock* variables (or, just *clocks*). To each clock x we associate a *domain*, $dom(x)$, which is either an integer interval that is unbounded to the right, or the union of a bounded integer interval with the single element ∞. So $[0, \infty)$, $[6, \infty)$ and $(-3, 4] \cup \{\infty\}$ are possible clock domains, whereas R, $[0, \infty]$ and $[0, 5]$ are not. A domain that is obtained by adding ∞ to a bounded interval is called *bounded*. We write $intv(x) \triangleq dom(x) - \{\infty\}$, $lb(x) \triangleq \inf intv(x)$ and $ub(x) \triangleq \sup intv(x)$. We assume that for each domain D there are infinitely many clocks in $\mathcal{C}$ to which D is associated. If D is a clock domain (or interval) and n is an integer, then we write $D - n$ to denote the clock domain (or interval) $\{t - n \mid t \in D\}$.

Beside a universe of clock variables, we also assume a countable universe $\mathcal{P}$ of *propositional* variables disjoint from $\mathcal{C}$. For all propositional variables p the *domain*, $dom(p)$, equals the set $\{\mathsf{T}, \mathsf{F}\}$ of truth values.

Finally, we assume the presence of a universe $\Sigma \supseteq \mathsf{R}^{\geq 0}$ of *actions*, ranged over by $a, \ldots$.

Definition 3.3 *(Constraints, assignments and transition tables).* Let P, P_1, P_2 be finite sets of propositional variables and let C, C_1, C_2 be finite sets of clock variables.

- *Terms over C* are expressions generated by the BNF grammar $e ::= x \mid n \mid e + n$, where $x \in C$ and $n \in \mathsf{Z}^{\infty}$. We denote the set of all such terms by $T(C)$.
- *Inequations over C* are expressions of the form $e \leq e'$ or $e < e'$ with $e, e' \in T(C)$. Inequations that contain two clock variables are also called *clock comparisons.*
- *Constraints over P and C* are Boolean combinations of propositional variables in P and inequations over C. We denote the set of all such formulas by $F(P, C)$. A constraint φ is *simple* if it does not contain clock comparisons, and *finitary* if it does not contain ∞.

  The Boolean constants T and F, denoting truth and falsehood, respectively, as well as equations $x = n$ are definable by simple constraints. In fact, for each integer interval I, the predicate $x \in I$ can be expressed as a simple, finitary constraint $\varphi_I(x)$. In inductive proofs we will often use that each constraint can be rewritten such that it only contains inequations of the form $x \leq n$, $x < n$, $x \leq y + n$ and $x < y + n$.

  Let f be a term or constraint, let e be a term, and let x be a clock. The *substitution* of x by e in f, notation $f[e/x]$, is the term or constraint that is obtained from f by replacing all occurrences of x by e. For $\mathbf{x}$ a list $x_1, \ldots, x_n$ of distinct clocks and $\mathbf{e}$ a list $e_1, \ldots, e_n$ of terms, the *simultaneous substitution* $f[\mathbf{e}/\mathbf{x}]$ denotes the simultaneous replacement in f of the variables of $\mathbf{x}$ by the corresponding terms of $\mathbf{e}$.

- *Assignments from* $C_1 \cup P_1$ *to* $C_2 \cup P_2$ are expressions of the form $p := \varphi$ with $p \in P_2$ and $\varphi \in F(P_1, C_1)$, or of the form $x := e$ with $x \in C_2$ and $e \in T(C_1)$. A *simultaneous assignment* λ *from* $C_1 \cup P_1$ *to* $C_2 \cup P_2$ is a finite set of assignments from $C_1 \cup P_1$ to $C_2 \cup P_2$ such that there is exactly one assignment to each $u \in C_2 \cup P_2$. If an assignment $u := f$ occurs in λ, then we write $\lambda(u) = f$. We define $Cons(\lambda)$ to be the conjunction, for each assignment $x := e$ in λ, of the constraint $e \in dom(x)$. A (simultaneous) assignment is *finitary* if it does not contain ∞.
- *Transition tables over* P *and* C are finite sets of *guarded commands* of the form $a : \varphi \Rightarrow \lambda$, where $a \in \Sigma \setminus \mathsf{R}^{\geq 0}$, $\varphi \in F(P, C)$ and λ is a simultaneous assignment from $P \cup C$ to $P \cup C$.

Definition 3.4 *(States and operations on states).* Let P, C be finite sets of propositional and clock variables, respectively.

- A *state over* P *and* C is a valuation of the variables in $P \cup C$, i.e., a function that maps each variable in $P \cup C$ to an element of its domain. We write $\mathcal{S}(P, C)$ for the set of states over P and C. If s and s' are states and u is a variable, then we write $s =_u s'$ to denote that s and s' agree on all variables except u. Similarly, we write $s =_U s'$ to denote that s and s' agree on all variables except those contained in the set U.
- Given a term or constraint $e \in T(C) \cup F(P, C)$ and a state $s \in \mathcal{S}(P, C)$, we write $eval(s, e)$ to denote the value to which e evaluates under valuation s. If e is a constraint then we write $s \models e$ if $eval(s, e) = \mathsf{T}$. A constraint φ is *satisfiable* if there exists a state s such that $s \models \varphi$; constraint φ *holds* if for all states s, $s \models \varphi$. If λ is a simultaneous assignment and $s \models Cons(\lambda)$, then we define $s[\lambda]$ to be the state satisfying, for all u, $s[\lambda](u) = eval(s, \lambda(u))$. This notation is extended to sets of states S by pointwise extension: $S[\lambda] \triangleq \{s[\lambda] \mid s \in S\}$.
- Let $s \in \mathcal{S}(P, C)$ be a state and $d \in \mathsf{R}^{\geq 0}$. Then $s \oplus d$ is the state given by

$$(s \oplus d)(u) = \begin{cases} s(u) & \text{if } u \in P \\ s(u) + d & \text{if } u \in C \text{ and } s(u) + d \in intv(u) \\ \infty & \text{otherwise} \end{cases}$$

A constraint φ is *past-closed* if, for all states s and all $d \in \mathsf{R}^{\geq 0}$, $s \oplus d \models \varphi$ implies $s \models \varphi$.

Definition 3.5 *(Timed automata).* A *timed automaton* A is a tuple $\langle P, C, Inv, Init, G \rangle$ where

- P is a finite set of propositional variables,
- C is a finite set of clocks,
- Inv is a constraint over P and C,
- $Init$ is a satisfiable constraint over P and C such that $Init \rightarrow Inv$ holds,
- G is a transition table over P and C. We demand that, for each guarded command $a : \varphi \Rightarrow \lambda$ occurring in G, the implication $\varphi \wedge Inv \rightarrow Cons(\lambda)$ holds.

The components of A are denoted by P_A, C_A, etc. We say that A is *finitary* if all constraints and assignments are finitary.

Definition 3.6 *(Special cases).*

- A *timed safety automaton (TSA)* is a timed automaton in which all clocks have domain $\mathsf{R}^{\geq 0}$, all assignments are of the form $x := 0$ or $x := x$, and all constraints are finitary.
- A *bounded time domain automaton (BTDA)* is a timed automaton in which all clocks have a bounded domain.

Our definition of a TSA is essentially the same as the original definition of Henzinger et al. [5], but we made some small changes for technical convenience. Our notion of a TSA corresponds to what [5] call a *real-time program*, with the following differences: (1) we have included the sets of variables as explicit components of a TSA, (2) we added initial states and actions to make behavioral comparison of automata possible, and (3) we removed the requirement from [5] that invariants are past-closed (instead we have an additional requirement in the definition of the operational semantics).

The automata from Figures 1-7 of the previous section can all easily be viewed as timed automata. For this the following notation is useful:

Definition 3.7. Given a finite set P of propositional variables and an element $q \in P$, we write $loc_P = q$ for the simple constraint $q \wedge \bigwedge_{p \in P \setminus \{q\}} \neg p$ and $loc_P := q$ for the simultaneous assignment $\{q := \mathsf{T}\} \cup \{p := \mathsf{F} \mid p \in P \setminus \{q\}\}$. We write $loc = q$ and $loc := q$ when P is clear from the context.

In order to view the automata from Section 2 as timed automata, we introduce a propositional variable for each vertex in the graph and impose as invariant the constraint $Loc_L = \bigvee_{l \in L} loc = l$, where L is the set of all vertices. All clocks have value 0 in the initial states unless specified otherwise, and constraints T and assignments $x := x$ are omitted from the diagrams. Thus, for example, the automaton of Figure 1 corresponds to the TSA A with $P_A = \{l_0, l_1\}$, $C_A = \{x\}$, $Inv_A = Loc_P$, $Init_A = (loc = l_0 \wedge x = 0)$, and $G_A = \{a : (loc = l_0 \wedge x \leq 2) \Rightarrow \{loc := l_1, x := x\}\}$.

3.2 Semantics

In this subsection we will define the semantics of timed automata by showing how to each timed automaton a transition system can be associated.

Definition 3.8 *(Transition systems).* A *transition system* B is a tuple $\langle S, S^0, \rightarrow \rangle$ where

- S is a set of *states*,
- $S^0 \subseteq S$ is a nonempty set of *initial states*,
- $\rightarrow \subseteq S \times \Sigma \times S$ is a *transition relation*.

The components of B are denoted by S_B, S_B^0, etc. We write $s \xrightarrow{a}_B s'$ for $\langle s, a, s' \rangle \in \rightarrow$. When B is clear from the context we just write $s \xrightarrow{a} s'$.

Definition 3.9 *(Operational semantics).* Let A be a timed automaton. Then the *operational semantics of* A, notation $TS(A)$, is the transition system B given by

- $S_B = \{s \in \mathcal{S}(P_A, C_A) \mid s \models Inv_A\}$,
- $S_B^0 = \{s \in \mathcal{S}(P_A, C_A) \mid s \models Init_A\}$,
- $\rightarrow_B$ consists of all triples of the form $\langle s, a, s' \rangle$ with $s, s' \in S_B$ and $a \in \Sigma$ such that
  - If $a \notin \mathsf{R}^{\geq 0}$ then, for some guarded command $a : \varphi \Rightarrow \lambda$ in G_A, $s \models \varphi$ and $s' = s[\lambda]$.
  - If $a \in \mathsf{R}^{\geq 0}$ then $s' = s \oplus a$ and, for all $d \in [0, a]$, $s \oplus d \in S_B$.

We write $\mathcal{S}(A)$ for S_B, $\mathcal{I}(A)$ for S_B^0, and $\mathcal{T}(A)$ for $\rightarrow_B$. Transitions $s \xrightarrow{d} s'$ with $d \in \mathsf{R}^{\geq 0}$ are known as *time steps* or *delays*. A valuation of the variables in P_A is called a *location*. Note that time steps do not change the location of a state and that time can only progress in a location as long as the invariant Inv_A is not violated.

The above operational semantics is essentially the same as the operational semantics defined in [5] but again there are some minor technical differences: (1) we have not included stutter steps, (2) we restrict the set of states to those that satisfy the invariant, and (3) we require that the invariant holds for all intermediate states passed through in a time step; this condition is automatically fulfilled in [5] since there invariants are required to be past-closed.

3.3 Bisimulations and Minimality

Definition 3.10 *(Bisimulation and minimality)*. Let B be a transition system. A relation $R \subseteq S_B \times S_B$ is a *bisimulation on* B iff

- $R(s_1, s_2)$ and $s_1 \xrightarrow{a} s_1'$ implies that there is an $s_2' \in S_B$ such that $s_2 \xrightarrow{a} s_2'$ and $R(s_1', s_2')$,
- $R(s_1, s_2)$ and $s_2 \xrightarrow{a} s_2'$ implies that there is an $s_1' \in S_B$ such that $s_1 \xrightarrow{a} s_1'$ and $R(s_1', s_2')$.

Two states s, s' of B are *bisimilar*, notation $s \simeq_B s'$, if there exists a bisimulation R on B with $R(s, s')$.

We say that B is *minimal (up to bisimulation)* if, for all states s, s' of B, $s \simeq_B s'$ implies $s = s'$. A timed automaton A is *minimal* iff $TS(A)$ is minimal.

Definition 3.11 *(Bisimulation of Transition Systems)*. Two transition systems B_1 and B_2 are *bisimilar*, notation $B_1 \simeq B_2$, if there exists a bisimulation R on the disjoint union of B_1 and B_2 that relates each start state of B_1 to a start state of B_2, and each start state of B_2 to a start state of B_1. Two timed automata A_1 and A_2 are bisimilar, notation $A_1 \simeq A_2$, if the corresponding transition systems $TS(A_1)$ and $TS(A_2)$ are bisimilar.

4 From BTDA's to TSA's

In this section we show that for every BTDA there exists a bisimilar TSA. This is done in two phases. First, in Section 4.2 to Section 4.5, we prove that for every BTDA there exists a bisimilar BTDA satisfying the following three properties: (a) there exists an $N \in \mathsf{N}$ such that for every clock x in $C_{A'}$, $dom(x) = [0, N] \cup \{\infty\}$, (b) assignments are of the form $x := 0$ or $x := x$, and (c) constraints are finitary. Then, in Section 4.6, we show that domains $[0, N] \cup \{\infty\}$ can be modified to $\mathsf{R}^{\geq 0}$.

The first phase consists of several steps, each of which takes up one subsection. First, we show that all constraints and assignments may be assumed to be finitary. Next, we show that time intervals may be assumed to have lower bound 0. After that, we remove assignments of the form $x := n$ or $x := y+n$ with $n \in \mathsf{Z}$, $n \neq 0$. Having removed time shifts altogether, we proceed to show that all clocks may be assumed to have an equal domain. Finally, we show how to remove assignments of the form $x := y$ with $y \neq x$.

4.1 Removing ∞

In this section, we show that infinitary assignments and constraints can be eliminated from BTDA's. The key idea is to introduce, for each clock x, a new propositional variable p_x that records whether clock x has (recently) been subjected to an infinitary assignment. We then adapt all invariants and guarded commands by rewriting them to formulas that do not contain ∞, that do not refer to any of the clocks x for which p_x is true, and that are equivalent to the original formulas under the assumption that all these clocks have value ∞. For instance, if p_x is true while p_y is false, then the formula $z \leq x \wedge y < 5 \wedge u \leq \infty$ rewrites to $\mathsf{T} \wedge y < 5 \wedge \mathsf{T}$. Since clocks x for which p_x is true do not occur in the resulting formulas, their value becomes irrelevant and the removal of the infinitary assignments to these clocks is harmless.

Formally, we prove that BTDA's may be assumed to be finitary, as defined below.

Definition 4.1. A BTDA A is called *finitary* when constraints in A are finitary, and for every guarded command $a : \varphi \Rightarrow \lambda \in G_A$ and every assignment $x := e$ in λ either $e \equiv x$, or the implication $\varphi \wedge Inv_A \rightarrow e \in intv(x)$ holds.

Consider an assignment $x := y + n$ with $n \in \mathsf{N}$. If in a state s this assignment occurs and $s(y) = \infty$, then x is 'implicitly' assigned the value ∞. We make this 'explicit' by replacing such an assignment by the assignment $x := \infty$.

Lemma 4.2. *For every BTDA A there exists a BTDA A' such that $A \simeq A'$ and for every guarded command $a : \varphi \Rightarrow \lambda \in G_A$ and every assignment $x := e$ in λ either $e \equiv \infty$, or the implication $\varphi \wedge Inv_A \rightarrow e \in intv(x)$ holds.*

Proof. Let $A = \langle P, C, Inv, Init, G\rangle$ be given. We define $A' = \langle P, C, Inv, Init, G'\rangle$ as follows. Let $a : \varphi \Rightarrow \lambda \in G$. We define $is_\infty(\lambda)$ as the set of conjunctions ψ that contain for each $x \in C$ precisely one conjunct, which is either of the form $\lambda(x) \in intv(x)$ or of the form $\lambda(x) \notin intv(x)$. For each formula $\psi \in is_\infty(\lambda)$, we write $expl_\infty(\lambda, \psi)$ for the result of replacing each assignment $x := \lambda(x)$ such that the formula $\lambda(x) \notin intv(x)$ occurs in ψ by $x := \infty$. The guarded command $a : \varphi \Rightarrow \lambda$ is replaced by the set of guarded commands $a : \varphi \wedge \psi \Rightarrow expl_\infty(\lambda, \psi)$, for all $\psi \in is_\infty(\lambda)$. It is easy to check that A' satisfies the requirements stated in the lemma. □

Next, we show how to remove occurrences of ∞ from constraints and assignments.

Definition 4.3. Let φ be a constraint over P and C and let $X \subseteq C$. We define $fin(\varphi, X)$ by induction on the structure of φ.

$$fin(x \le n, X) = \begin{cases} \mathsf{T} & \text{if } n = \infty \\ \mathsf{F} & \text{if } n \ne \infty \text{ and } x \in X \\ x \le n & \text{otherwise} \end{cases}$$

$$fin(x < n, X) = \begin{cases} \mathsf{F} & \text{if } x \in X \\ x \in intv(x) & \text{if } x \notin X \text{ and } n = \infty \\ x < n & \text{otherwise} \end{cases}$$

$$fin(x \le y + n, X) = \begin{cases} \mathsf{T} & \text{if } y \in X \text{ or } n = \infty \\ y \notin intv(y) & \text{if } (y \notin X \text{ and } n \ne \infty) \text{ and } x \in X \\ x \le y + n & \text{otherwise} \end{cases}$$

$$fin(x < y + n, X) = \begin{cases} \mathsf{F} & \text{if } x \in X \\ x \in intv(x) & \text{if } x \notin X \text{ and } (y \in X \text{ or } n = \infty) \\ x < y + n & \text{otherwise} \end{cases}$$

$$fin(\varphi_1 \Box \varphi_2, X) = fin(\varphi_1, X) \Box fin(\varphi_2, X) \qquad \Box \in \{\wedge, \vee\}$$

$$fin(\neg\varphi_1, X) = \neg fin(\varphi_1, X)$$

Note that $fin(\varphi, X)$ is finitary and that clocks from X do not occur in $fin(\varphi, X)$.

Lemma 4.4. *Let s, s' be states over P and C, and let $X \subseteq C$ such that, for all $x \in C$,*

$$s(x) = \begin{cases} \infty & \textit{if } x \in X \\ s'(x) & \textit{otherwise} \end{cases}$$

Then we have, for all constraints φ over P and C,

$$s \models \varphi \Leftrightarrow s' \models fin(\varphi, X).$$

Theorem 4.5. *For every BTDA A there exists a finitary BTDA A' such that $A \simeq A'$.*

Proof. Let $A = \langle P, C, Inv, Init, G\rangle$ be given. Let, for each $x \in C$, p_x be a fresh propositional symbol and let, for $X \subseteq C$, p_X abbreviate the formula $\bigwedge_{x\in X} p_x \wedge \bigwedge_{y \in C/X} \neg p_y$. We define $A' = \langle P', C, Inv', Init', G'\rangle$ by:

- $P' = P \cup \{p_x \mid x \in C\}$.
- $Inv' = \bigvee_{X \subseteq C}(p_X \wedge fin(Inv, X))$
- $Init' = (p_\emptyset \wedge fin(Init, \emptyset))$.
- For each guarded command $a : \varphi \Rightarrow \lambda$ in G and for each $X \subseteq C$, G' contains a guarded command $a : fin(\varphi, X) \wedge p_X \Rightarrow \lambda'$, where λ' is the simultaneous assignment from $P' \cup C$ to $P' \cup C$ given by

$$\lambda'(u) = \begin{cases} fin(\lambda(u), X) & \text{if } u \in P \\ \mathsf{T} & \text{if } \exists y \in Y : u = p_y \\ \mathsf{F} & \text{if } \exists y \in C/Y : u = p_y \\ u & \text{if } u \in Y \\ \lambda(u) & \text{otherwise} \end{cases}$$

  where Y is the set of clocks for which $\lambda(y)$ is infinitary or contains a clock from X.

Using Lemma 4.4, it is easily seen that A' is well-defined. For each $s \in \mathcal{S}(A')$, define $h(s)$ to be the state over $P \cup C$ given by

$$h(s)(u) = \begin{cases} \infty & \text{if } u \in C \text{ and } s(p_u) = \mathsf{T} \\ u & \text{otherwise} \end{cases}$$

Define relation R by $R = \{(h(s), s) \mid s \in \mathcal{S}(A')\}$. Using Lemma 4.4, it is routine to verify that R is a bisimulation between A and A'. □

4.2 Changing lower bounds of clock domains to 0

We proceed to show that for every BTDA A there exists a bisimilar BTDA A' with the property that the time domain of every clock in A' has lower bound 0. Intuitively, we *shift* the domain of each clock x and its valuations by $lb(x)$. Since the domain of a clock is hard-wired in the identity of the clock, this is achieved by taking a copy x' of clock x with the new domain. For instance, if the domain of x is $[-4, 7) \cup \{\infty\}$ then the domain of the copy x' will be $[0, 11) \cup \{\infty\}$, and if state s' of A' corresponds to state s of A then $s'(x') = s(x) + 4$. To ensure that the resulting BTDA is well-defined and bisimilar to A, we also have to shift formulas and assignments. E.g., the formula $x \leq 5$ will be shifted to $x' \leq 9$, and the assignment $x := 5$ will be shifted to $x' := 9$.

In the proof of the theorem below and later on in the paper, we use the following notation. Given a function f and vector $\mathbf{x} = x_1, \ldots, x_n$ we write $f(\mathbf{x})$ for the vector $f(x_1), \ldots, f(x_n)$. In a similar way also binary operators are lifted to vectors.

Theorem 4.6. *For every BTDA A there exists a BTDA A' such that $A \simeq A'$ and $lb(x') = 0$ for every clock x' of A'.*

Proof. Let $A = \langle P, C, Inv, Init, G \rangle$ be given. Let $C = \{x_1, \ldots, x_n\}$. Associate to each clock $x \in C$ a fresh clock x' with $dom(x') = dom(x) - lb(x)$. Let $C' = \{x' \mid x \in C\}$. If φ is a constraint over P and C, then let $sh(\varphi)$ denote the constraint $\varphi[\mathbf{x}' + lb(\mathbf{x})/\mathbf{x}]$ over P and C'. Also, if s is a state over $P \cup C$ then let $sh(s)$ be the state over $P \cup C'$ given by

$$sh(s)(u) = \begin{cases} s(u) & \text{if } u \in P \\ s(x) - lb(x) & \text{if } u = x' \in C' \end{cases}$$

(Note that sh is a bijection.) A straightforward induction gives that, for each constraint φ over P and C and for each state s over $P \cup C$,

$$s \models \varphi \Leftrightarrow sh(s) \models sh(\varphi) \tag{1}$$

Now define A' to be equal to $\langle P, C', Inv', Init', G' \rangle$, where

– $Inv' = sh(Inv)$
– $Init' = sh(Init)$
– For each guarded command $a : \varphi \Rightarrow \lambda$ in G, G' contains a guarded command $a : sh(\varphi) \Rightarrow sh(\lambda)$, where $sh(\lambda)$ is the simultaneous assignment from $P \cup C'$ to $P \cup C'$ given by

$$sh(\lambda)(u) = \begin{cases} sh(\lambda(u)) & \text{if } u \in P \\ sh(\lambda(x)) - lb(x) & \text{if } u = x' \in C' \end{cases}$$

where, for $x \in C$

$$sh(\lambda(x)) = \begin{cases} n & \textit{if } \lambda(x) = n \\ y + lb(y) + n & \textit{if } \lambda(x) = y + n \end{cases}$$

Using (1), it is routine to show that A' is a well-defined BTDA: E.g., let $a : sh(\varphi) \Rightarrow sh(\lambda)$ be a guarded command in G' as above. We show that the implication $sh(\varphi) \wedge Inv' \rightarrow Cons(sh(\lambda))$ holds. This is equivalent to showing that, for each state s' over $P \cup C'$ and for each $x' \in C$,

$$s' \models sh(\varphi) \wedge Inv' \rightarrow sh(\lambda(x)) - lb(x) \in dom(x) - lb(x)$$

which in turn is equivalent to

$$s' \models sh(\varphi \wedge Inv \rightarrow \lambda(x) \in dom(x))$$

Using (1), this reduces to

$$sh^{-1}(s') \models \varphi \wedge Inv \rightarrow \lambda(x) \in dom(x)$$

which is directly implied by the assumption that A is a BTDA.

Clearly, $lb(x') = 0$ for each clock x' of A'.

Let R be equal to the mapping sh viewed as a relation, i.e., $R = \{(s, sh(s)) \mid s \in \mathcal{S}(A)\}$. Using (1), it is routine to check that R is a bisimulation (an isomorphism, in fact) between A and A'. Here one can use the observations that, for all $s \in \mathcal{S}(A)$, $d \in \mathsf{R}^{\geq 0}$, and simultaneous assignments λ to variables in $P \cup C$, $sh(s \oplus d) = sh(s) \oplus d$ and $sh(s[\lambda]) = sh(s)[sh(\lambda)]$. □

4.3 Removing finitary time shifts

In this section we show that one can replace assignments of the form $x := n$ and $x := y+n$ with $n \neq 0$ by assignments of the form $x := 0$ and $x := y$, respectively.

The idea is to encode time shifts in the identity of variables. For instance, an assignment $x := n$ is replaced by the assignment $x_n := 0$ and x_n plays the role of x until a new assignment to x occurs (e.g., in formulas, x is replaced by x_n). The fact that the clock x_n is actually the clock x shifted n time units is modeled by putting $ub(x_n) = ub(x) - n$ and shifting formulas n time units at clock x. For each location l we keep track of the current time shifts of clocks by means of functions h that map each clock $x \in C$ to a time shift $h(x)$ (and propositional variables p_h for these functions). So if in location l, h is the current function and $h(x) = n$ then x_n plays in l the role of x.

A crucial property to be established is of course that the set of time shift functions need not be infinite. To prove this, we show that time shifts have to be accumulated only up to a certain point. Consider e.g. an assignment of the form $x := y + n$ with $y + n \not\equiv x + 0$. When $h(y) + n$ lies within a certain range $stretch(x)$, this assignment is replaced by $x_{h(y)+n} := y_{h(y)}$ and the new h value of x is $h(y) + n$.

Definition 4.7. Let A be a BTDA, $x \in C_A$. Define $\mathsf{max}(A) = \max\{ub(x) \mid x \in C_A\}$. Define $stretch(x) = [-\mathsf{max}(A), ub(x)] \cap \mathsf{Z}$.

For a given clock x in C_A, only time shifts $h(y)+n$ in $stretch(x)$ need to be considered. This can be seen as follows. Suppose $h(y)+n \notin stretch(x)$. Roughly, if h is the current time shift function, then the value $s_h(y)$ of a clock y in A equals $h(y)$ plus the value $s(y_{h(y)})$ of $y_{h(y)}$ in A'. Suppose now that there exists a guarded command $a : \varphi \Rightarrow \lambda \in G_A$ with $\lambda(x) = y+n$, such that $s_h \models Inv_A \wedge \varphi$. By Theorem 4.5, $s_h(y)+n \in intv(x)$, i.e., $s(y_{h(y)}) + h(y) + n \in intv(x)$. From this we will be able to infer that $s(y_{h(y)}) \in [0, \mathsf{max}(A)]$. But this is impossible, since $h(y)+n \notin stretch(x)$. So $h(y)+n \notin stretch(x)$ implies that the guarded command $a : \varphi \Rightarrow \lambda$ is not enabled.

Note that the time shift value $h(x)$ of a clock x may be negative, which implies that the domain of $x_{h(x)}$ extends the domain of x and that $s(x_{h(x)}) + h(x)$ may be strictly negative. However, we will maintain as an invariant that integer values of the new clocks $x_{h(x)}$ do not exceed $\mathsf{max}(A)$ and are such that $s(x_{h(x)}) + h(x) \geq 0$. This is reflected in the notion *area*, below.

Definition 4.8. Let C be a set of clocks.

1. To each clock $x \in C$ associate a set of clocks $C_x = \{x_n \mid n \in stretch(x)\}$. For each clock $x_n \in C_x$, $dom(x_n) = dom(x) - n$. Intuitively, $x_0 = x$. Put $sh(C) = \bigcup_{x \in C} C_x$.
2. Define H_C as the (finite) set of functions h which map each clock $x \in C$ to an element $h(x)$ of $stretch(x)$. We let h_0 be the function that maps each clock to 0.
3. For $x \in C$ and $h \in H_C$, put $area(x,h) = [\max\{0, -h(x)\}, \mathsf{max}(A)] \cup \{\infty\}$. A state s over P and $\bigcup_{x \in C} x_{h(x)}$ is called *h-compliant* when for all $x_{h(x)} \in \bigcup_{x \in C} x_{h(x)}$, $s(x_{h(x)}) \in area(x,h)$.
4. A simultaneous assignment λ to clocks in C defines a function from H_C to H_C as follows.
$$\lambda(h)(x) = \begin{cases} n & \text{if } \lambda(x) = n \\ h(y)+n & \text{if } \lambda(x) = y+n \text{ and } h(y)+n \in stretch(x) \\ ub(x) & \text{otherwise} \end{cases}$$
5. Conversely, a function $h \in H_C$ induces a function on simultaneous assignments to variables in $P \cup C$ as follows. To each clock x, h associates a clock $x_{h(x)}$. Given this association, we define $sh(\varphi, h)$ as $\varphi[\mathbf{x}' + h(\mathbf{x})/\mathbf{x}]$ over P and C'. Then $h(\lambda)$ is the result of replacing in λ every assignment of the form $p := \varphi$ by $p := sh(\varphi, h)$, every assignment of the form $x := n$ by $x_n := 0$, and every assignment of the form $x := y+n$ by $x_{\lambda(h)(x)} := y_{h(y)}$.
6. For $h \in H_C$ and an h-compliant state s over P and $\bigcup_{x \in C} x_{h(x)}$, let s_h be the state over P and C defined by
$$s_h(u) = \begin{cases} s(u) & \text{if } u \in P \\ s(u_{h(u)}) + h(u) & \text{if } u \in C \end{cases}$$

Theorem 4.9. *For every BTDA A there exists a BTDA A' such that $A \simeq A'$ and A' contains no assignments of the forms $x := n$ or $x := y+n$ with $n \neq 0$.*

Proof. Let $A = \langle P, C, Inv, Init, G \rangle$ be given. We define $A' = \langle P', C', Inv', Init', G' \rangle$ as follows.

- P' is P extended with, for each function $h \in H_C$, a fresh propositional variable p_h.
- $C' = sh(C)$.
- $Inv' = \bigvee_{h \in H_C} (loc = p_h \wedge sh(Inv, h))$.
- $Init' = (loc = p_{h_0} \wedge sh(Init, h_0))$.
- G' is defined as follows. For every $h \in H_C$ and guarded command $a : \varphi \Rightarrow \lambda \in G$, G' contains a guarded command $a : sh(\varphi, h) \wedge p_h \Rightarrow h(\lambda) \cup \{loc := p_{\lambda(h)}\}$.

Define a bisimulation over $\mathcal{S}(A')$ and $\mathcal{S}(A)$ by $R = \{(s, s_h) \mid s \in \mathcal{S}(A'),\ s \models p_h\}$. $\square$

4.4 Equalizing domains

In this section, we prove that for every BTDA A there exists a bisimilar BTDA A' such that the domains of clocks in A' are all equal. Domains are made equal to the largest domain in A. More precisely, to each clock $x \in C_A$ we associate a clock x' with domain $[0, \mathsf{max}(A)] \cup \{\infty\}$ (for $\mathsf{max}(A)$, see Definition 4.7). Of course we have to compensate for the extension of the domains. We keep track of which clocks x' have values in the original interval $intv(x)$ and which clocks don't. Similar to the proof of Lemma 4.2, this is done by means of conjunctions of formulas $x' \in intv(x)$ and $x' \notin intv(x)$. To each such conjunction ψ we associate the set X_ψ of clocks x' such that the formula $x' \notin intv(x)$ occurs in ψ, i.e., the value of x in A would have been ∞. For every conjunction ψ, we make a local copy of invariants and guarded commands by applying the $fin(\cdot, X_\psi)$ function of Section 4.1 to them.

Next, consider the assignment $x := y$. In the approach outlined above, this assignment would be translated to $x' := y'$. If $y' \notin intv(y)$ then this assignment should have the effect that x' is assigned the value ∞. As in the proof of Theorem 4.5 we do not perform this assignment but simply store the information that x' actually has value ∞ by means of additional sets X of clocks (and propositional variables for them).

Theorem 4.10. *For every BTDA A there exists a BTDA A' such that $A \simeq A'$ and all clocks have the same domain.*

Proof. Let $A = \langle P, C, Inv, Init, G\rangle$ be given. Associate to each clock $x \in C$ a fresh clock x' with domain $[0, \mathsf{max}(A)] \cup \{\infty\}$. Put $C' = \{x' \mid x \in C\}$ and define $act(C')$ as the set of conjunctions ψ of formulas of the form $x' \in intv(x)$ and $x' \notin intv(x)$ such that each clock x' occurs precisely once in ψ. To each formula $\psi \in act(C')$, we associate the set $X_\psi \subseteq C'$ of clocks x' such that the formula $x' \notin intv(x)$ occurs in ψ.

Let, for each $z \in C'$, p_z be a fresh propositional symbol and let, for $X \subseteq C'$, p_X abbreviate the formula $\bigwedge_{z\in X} p_z \wedge \bigwedge_{y \in C/X} \neg p_y$.

We define $A' = \langle P', C', Inv', Init', G'\rangle$ by

- $P' = P \cup \{p_{x'} \mid x \in C\}$.
- $Inv' = \bigvee_{X \subseteq C'} \bigvee_{\psi \in act(C')} (p_X \wedge \psi \wedge fin(Inv, X \cup X_\psi))$.
- $Init' = \bigvee_{\psi \in act(C')} (p_\emptyset \wedge \psi \wedge fin(Init, X_\psi))$.
- G' is defined as follows. Let $a : \varphi \Rightarrow \lambda \in G$ and let μ be the result of replacing in λ each $x \in C$ by x'. For every $X \subseteq C'$ and formula $\psi \in act(C')$, G' contains the guarded command $a : fin(\varphi, X \cup X_\psi) \wedge \psi \wedge p_X \Rightarrow \mu'$, where μ' is defined from μ as λ' is defined from λ in the proof of Theorem 4.5, but with X replaced by $X \cup X_\psi$.

For $s \in \mathcal{S}(A')$ such that $s \models p_X$, define s_X over P and C by

$$s_X(u) = \begin{cases} s(u) & \text{if } u \in P \\ s(u') & \text{if } u \text{ in } C,\ s(u') \in intv(u) \text{ and } u' \notin X \\ \infty & \text{otherwise.} \end{cases}$$

We define the relation R over $\mathcal{S}(A')$ and $\mathcal{S}(A)$ by $\{(s, s_X) \mid s \in \mathcal{S}(A'), s \models p_X\}$. Using a suitable adaptation of Lemma 4.4, it is easy to check that A' and R are well-defined and that R is a bisimulation. □

4.5 Removing clock references from BTDA's

In this section we show how to remove assignments of the form $x := y$ with $x \neq y$ from BTDA's. The constructions involved are somewhat complicated, so we only give an outline. The basic idea is that instead of performing such an assignment in a certain location, we encode, by means of propositional variables, in the location the information that the value of x equals the value of y and we let y play the role of x until another assignment to x occurs (i.e., y is substituted for x in constraints and assignments). For this it is essential that the domains of x and y are equal. A problematic situation is, of course, when y plays the role of x and becomes itself the subject of an assignment $y := e$ with $e \neq y$. If the assignment to y if of the form $y := z$ with $z \neq y$, the problem is easily solved: the assignment to y is not actually performed but instead the information is stored that x refers to y and y refers to z. But if the assignment is of the form $y := 0$, this trick does not work.

We solve this problem in two steps. First, we introduce for each location l a new clock $clock(l)$ and add the assignment $clock(l) := 0$ (in such a way that this assignment is only applicable when entering location l). The new clocks will refer only to themselves. All other assignments of the form $y := 0$ in location l are removed, while storing the information that y refers to $clock(l)$. So the original clocks are no longer reset to 0 and the clocks that are reset, refer only to themselves. This does not solve our problem completely: consider a loop from l to l such that somewhere on the loop x starts referring to $clock(l)$. When passing through l again, the value of $clock(l)$ is lost. This problem is solved by making an extra copy of the loop (and of $clock(l)$): one in which $clock(l,0)$ holds the value of $clock(l)$ from the previous loop and $clock(l,1)$ holds the current value of $clock(l)$, and one in which the situation is reversed. This is done by means of a toggle bit function β which returns for every location a bit $b \in \{0,1\}$, indicating the current loop; every time a loop passes l, the bit is toggled.

There is yet one snag. Consider two loops from l to l. On the first loop x starts referring to a copy of $clock(l)$, on the second loop this does not happen. Consider a walk through the automaton according to the following scenario. Leave l while $\beta(l) = 0$ and loop through l twice by concatenating the second loop after the first loop. It is clear that in the first loop, x starts referring to $clock(l,0)$ and that after the second pass through l the value of $clock(l,0)$ is lost for x. We will show that automata can be put in this form. After that, the construction of the automaton without assignments $x := y$ with $x \neq y$ outlined above will be given.

4.6 To unbounded domains

We have shown that for every BTDA A there exists a bisimilar BTDA A' satisfying the following three properties. There exists an $N \in \mathsf{N}$ such that for every clock x in $C_{A'}$, $intv(x) = intv = [0,N]$. Moreover, assignments to x are of the form $x := 0$ or $x := x$. Finally, constraints are finitary. To construct for A a bisimilar TSA, it remains to deal with the extension of the bounded domains of clocks in A to the domain $\mathsf{R}^{\geq 0}$. This can be done by a simple syntactic operation $(\cdot)^\infty$ on constraints with the property that a state s over the full domain $\mathsf{R}^{\geq 0}$ satisfies φ^∞ iff the restriction s_∞ of s to $[0,N] \cup \{\infty\}$ (mapping all values larger than N to ∞) satisfies φ. E.g., ignoring renaming of clocks, $(x \leq n)^\infty = x \leq n \wedge x \in intv$ and $(x \leq y + n)^\infty = (x \in intv \wedge x \leq y + n) \vee y \notin intv$.

Definition 4.11. Let A be a BTDA and $intv = intv(x)$ for all $x \in C_A$. Associate to each clock $x \in C_A$ a fresh variable x' with $dom(x') = \mathsf{R}^{\geq 0}$. Put $C^\infty = \{x' \mid x \in C_A\}$. Let φ be a finitary inequation over P_A and C_A. We define φ^∞ by induction on the structure of φ.

$$\begin{aligned}
(x \leq n)^\infty &= x' \in intv \wedge x' \leq n \\
(x < n)^\infty &= x' \in intv \wedge x' < n \\
(x \leq y + n)^\infty &= (x' \in intv \wedge x' \leq y' + n) \vee y' \notin intv \\
(x < y + n)^\infty &= x' \in intv \wedge (x' < y' + n \vee y' \notin intv)
\end{aligned}$$

Let s be a state over P_A and C^∞. Define s_∞ over P_A and C_A by

$$s_\infty(u) = \begin{cases} s(u) & \text{if } u \in P_A \\ s(y') & \text{if } u = y' \text{ with } x \in C_A \text{ and } s(y') \in intv \\ \infty & \text{otherwise} \end{cases}$$

The $(\cdot)^\infty$ function is extended to finitary constraints in the expected way.

Theorem 4.12. *For every BTDA A there exists a TSA A' such that $A \simeq A'$.*

5 From TSA's to BTDA's

In this section we show that for every TSA there exists a bisimilar BTDA. The format of TSA's almost immediately fits into the format of BTDA's, except for the boundedness of the domains. Let A be a TSA and let N be the largest integer constant occurring in constraints in A. To change A into a BTDA A' it seems sufficient, at first sight, to change the domain of each clock into $[0, N] \cup \{\infty\}$: then every state s of A corresponds to a state s' that is the same as s except that $s'(x) = \infty$ for all x with $s(x) > N$. This naive approach does not work, however, when A contains clock comparisons. Suppose for instance that A contains a clock comparison $x < y$ and a state s that satisfies $N < x < y$. Then the corresponding state s' does not satisfy $x < y$, and thus A and A' may behave differently. The problem is that in the TSA model the progress of time preserves the validity of clock comparisons, while this is not the case in the BTDA model. We circumvent this problem by proving that for every TSA there exists a bisimilar TSA that only contains simple constraints (i.e., without clock comparisons). The idea is to encode the relative positions of clocks in a certain state in the discrete part (location) of that state. Once all constraints are simple the naive transformation from TSA's to BTDA's can easily be shown correct.

Theorem 5.1. *For every TSA A there exists a TSA A' such that $A \simeq A'$ and constraints in A' are simple.*

Theorem 5.2. *For every TSA A there exists a BTDA A' such that $A \simeq A'$.*

6 Minimizable Timed Automata

Roughly speaking, a minimizable timed automaton is a bounded time domain automaton enriched with a mechanism to identify equivalent states. In order to make this precise, we need the auxiliary concept of a "preMTA".

6.1 Definitions

Definition 6.1 *(preMTA's).* A *preMTA* is a pair $M = \langle A, Rel\rangle$, where A is a BTDA and Rel is a function that associates to each clock in C_A a *relevance formula*, a past-closed simple constraint in $F(P_A, C_A)$.

A relevance formula $Rel(x)$ declares in which states clock x is *relevant*, and may take a value different from ∞. A clock that is not relevant is called *retired*. Since relevance formulas are past-closed, a clock that has retired remains so when time passes. However, after the occurrence of a discrete event a retired clock may get back to work again.

The operational semantics of a preMTA is defined as an abstraction of the operational semantics of the underlying BTDA.

Definition 6.2 *(Operational semantics).* Let $M = \langle A, Rel\rangle$ be a preMTA. For each $s \in \mathcal{S}(P_A, C_A)$, let $\rho(s)$ be the state given by

$$\rho(s)(u) = \begin{cases} \infty & \text{if } u \in C_A \text{ and } s \not\models Rel(u) \\ s(u) & \text{otherwise} \end{cases}$$

Mapping ρ applies to sets of states via pointwise extension. Let $B = TS(A)$. Then the *operational semantics of* M, notation $TS(M)$, is the transition system B' given by

- $S_{B'} = \rho(S_B)$,
- $S^0_{B'} = \rho(S^0_B)$,
- $\rightarrow_{B'}$ is the least relation satisfying $s \xrightarrow{a}_B s' \;\rightarrow\; \rho(s) \xrightarrow{a}_{B'} \rho(s')$.

Two preMTA's M_1 and M_2 are bisimilar, notation $M_1 \simeq M_2$, if the corresponding transition systems $TS(M_1)$ and $TS(M_2)$ are bisimilar. Similarly, we define bisimulation between preMTA's and BTDA's. A preMTA M is *minimal* iff $TS(M)$ is minimal.

Relevance formulas may declare that a clock is not relevant in a state, even though the clock is tested in this state and thus *appears* to be relevant. Consider, for instance, the TSA of Figure 1. We turn this into a BTDA A by giving x domain $[0, 2] \cup \{\infty\}$. Next we build a preMTA M by adding the "problematic" relevance formula $Rel(x) = (loc = l_0 \wedge x \leq 1)$. Since A has a step $(l_0, 2) \xrightarrow{a} (l_1, 2)$, M contains a step $(l_0, \infty) \xrightarrow{a} (l_1, \infty)$. But state (l_0, ∞) of A does not have an outgoing a transition, even though it is mapped by ρ onto the state (l_0, ∞) of M. Thus M and its underlying BTDA A behave essentially different. This type of situations is excluded in the notion of an MTA. Intuitively, an MTA is a preMTA in which the relevance formulas only declare that a clock has retired if it really has retired.

Definition 6.3 *(MTA's).* A *minimizable timed automaton (MTA)* $M = \langle A, Rel\rangle$ is a preMTA with the additional property that the function ρ from Definition 6.2 (viewed as a set of pairs) is a bisimulation between A and M.

The following theorem is a direct corollary of previous results that established the equivalence of timed safety automata and bounded time domain automata.

Theorem 6.4. *For every MTA M there exists a TSA A such that $M \simeq A$, and conversely, for every TSA A there exists an MTA M such that $A \simeq M$.*

Proof. Assume that $M = \langle A', Rel\rangle$ is an MTA. Then by definition M is bisimilar with the underlying BTDA A'. By Theorem 4.12, there exists a TSA A that is bisimilar with A'. Now $M \simeq A$ follows since bisimulation is an equivalence.

Conversely, assume A is a TSA. Then, by Theorem 5.2, there exists a BTDA A' that is bisimilar with A. Let M be the preMTA obtained by pairing A' with the function that associates relevance formula T to each clock of A'. Then it is trivial to see that ρ is an isomorphism from $TS(A')$ to $TS(M)$, which implies that M is an MTA that is bisimilar with A'. Again $M \simeq A$ follows since bisimulation is an equivalence. $\square$

6.2 Regions

In this section we define Alur and Dill's [3] notion of a "region" in the context of bounded time domain automata, and prove some lemmas that will be used later in the proof of our main result that for each MTA there exists a bisimilar minimal MTA.

Definition 6.5 *(Regions)*. Let A be a BTDA. The equivalence relation $\sim$ is defined over the set of all states of $TS(A)$; $s \sim s'$ iff the following conditions hold, for all $p \in P_A$ and $x, y \in C_A$,

1. $s(p) = s'(p)$.
2. $s(x) = \infty$ iff $s'(x) = \infty$.
3. If $s(x) \neq \infty$ then $\lfloor s(x) \rfloor = \lfloor s'(x) \rfloor$ and $[\mathit{fract}(s(x)) = 0$ iff $\mathit{fract}(s'(x)) = 0]$.
4. If $s(x) \neq \infty \neq s(y)$ then $\mathit{fract}(s(x)) \leq \mathit{fract}(s(y))$ iff $\mathit{fract}(s'(x)) \leq \mathit{fract}(s'(y))$.

A *region* for A is an equivalence class of states induced by $\sim$.

The following facts about regions are standard:

1. Each BTDA only has finitely many regions.
2. Each region α can be denoted by constraint χ_α in the sense that, for all states s, $s \in \alpha$ iff $s \models \chi_\alpha$.
3. Each constraint φ either holds for all states in a region α or for none of them: $s, s' \in \alpha$ implies $(s \models \varphi \Leftrightarrow s' \models \varphi)$. We write $\alpha \models \varphi$ iff $s \models \varphi$, for some $s \in \alpha$.
4. If α is a region and $a : \varphi \Rightarrow \lambda$ a guarded command of A such that $\alpha \models \varphi$ then $\alpha[\lambda]$ is also a region.

Definition 6.6. Let A be a BTDA and let s, s' be states of $TS(A)$. The *time successor relation* $\preceq$ is defined over the set of all states of $TS(A)$:

$$s \preceq s' \triangleq \exists d \in \mathsf{R}^{\geq 0} : s \xrightarrow{d} s.$$

We write $s\uparrow$ to denote the set of time successors of s; $s\uparrow \triangleq \{s' \mid s \preceq s'\}$. This notation is extended to sets of states S by pointwise extension; $S\uparrow \triangleq \bigcup\{s\uparrow \mid s \in S\}$.

It is well-known that, for each constraint φ, one can effectively construct a constraint $\varphi\uparrow$ such that the set of time successors of states that satisfy φ equals the set of states that satisfy $\varphi\uparrow$. Moreover, if φ corresponds to a region α then $\varphi\uparrow$ corresponds to the union of a finite number of regions, called the *successor regions* of α. Region α is a *predecessor region* of α' iff α' is a successor region of α.

Lemma 6.7. *Let α be a region of some BTDA A. Suppose that $s \in \alpha$ and $s' \in \alpha\uparrow$ are states such that $s \simeq s'$ and $s =_X s'$, for some set X of clocks. Then, for all states r, r' in $\alpha\uparrow$, $r =_X r'$ implies $r \simeq r'$.*

6.3 Main result

We now come to the main result of this paper.

Theorem 6.8. *For every MTA M there exists a bisimilar minimal MTA M' that can be effectively computed and that is unique up to isomorphism.*

Proof. (Sketch) The proof of this result is quite involved and we only outline its main structure here. Let $M = \langle A, \mathit{Rel} \rangle$ be an MTA. Minimization takes place in five phases:

1. Strengthening of the invariant constraint Inv_A so that all states of A that satisfy the invariant are reachable.
2. Application of a history variable construction: we add propositional variables that record the region entered through the last discrete action. As a result of this "spaghetti string strategy" each location is split into a number of new locations in such a way that the clock spaces of the new location are convex and at most one region wide.
3. Construction of relevance formulas that identify all bisimilar states.
4. Superposition of locations that have (part of) their clock space in common.

Ad 1 This can be done effectively using techniques of [5].

Ad 2 Let $A = \langle P, C, Inv, Init, G\rangle$. We define $A' = \langle P', C', Inv', Init', G'\rangle$ as follows.

- P' is P extended with, for each region α of A, a fresh propositional variable p_α.
- $C' = C$.
- $Inv' = (\bigvee_\alpha loc = p_\alpha \wedge \chi_\alpha \uparrow) \wedge Inv$.
- $Init' = (\bigvee_\alpha loc = p_\alpha \wedge \chi_\alpha) \wedge Init$.
- For every region α and guarded command $a : \varphi \Rightarrow \lambda \in G$ with $\alpha \models \varphi$, G' contains a guarded command $a : \chi_\alpha \Rightarrow \lambda \cup \{loc := p_{\alpha[\lambda]}\}$.

The construction of A' is clearly effective, and it is routine to verify that $A \simeq A'$.

Ad 3 This phase consists of three steps:

1. Say that a clock x is *free* in a region α if α contains two different states s, s' with $s =_x s'$. Take, for each clock x and region α for which x is free, two arbitrary states s and s' and decide whether they are bisimilar. This can be done using the result of Čerāns [10, 9] who proved that bisimulation is decidable for timed automata (this result carries over to our setting). According to Lemma 6.7, the outcome is independent of the choice of s and s'. If s and s' are not bisimilar then we declare that clock x is relevant for α, otherwise we declare that x is irrelevant for α.
2. If clock x has been declared relevant for α then we declare x to be relevant for all predecessors of α. Conversely, if clock x has been declared irrelevant for α then we declare x to be irrelevant for all successor regions of α.
3. If x has neither been declared relevant nor irrelevant for α in steps (1) and (2), then x has been declared relevant for all predecessors of α (except α) and irrelevant for all successors of α (except α). If $s(x) = \infty$ for all $s \in \alpha$ or α is maximal, i.e., if the only time successor of α is α itself, then we declare x to be relevant in α. Otherwise, let X be the set of clocks whose value coincides with that of x for all states of α. We can pick a state s in α and a state s' from a proper successor of α such that $s =_X s'$. If s and s' are not bisimilar then we declare that clock x is relevant for α, otherwise we declare that x is irrelevant for α.

Define $Rel(x)$ as the disjunction, for all regions α for which x has been declared relevant, of the constraint χ_α. Let ρ be the abstraction function on states induced by Rel according to Definition 6.2. Using Lemma 6.7, one can show that, for all states s, s' with the same location ($s =_{P_A} s'$), $\rho(s) = \rho(s')$ iff $s \simeq s'$.

Ad 4 This is the most technical part of the proof. After obtaining minimality of the state space for a fixed location, we now have to superimpose the state spaces of different locations. If two states from (minimized) regions with a different location are bisimilar, then in fact there exists an isomorphism between the two regions that is a bisimulation. Together with the result of Čerāns [10, 9] this allows us to decide which regions in the automaton are bisimilar. If two regions with a different location are bisimilar then this means that, from some point, the two state spaces ("spaghetti strings") of these locations are bisimilar and have to be merged. W.l.o.g. we may assume that the relevant clocks of the two state spaces S and S' are disjoint (this can be achieved through renamings). This disjointness allows us merge the two locations. Next we add copies of the clocks of S to both S and S' and ensure that (1) outside the parts that have to be merged the values of the copies are fully determined by the values of the originals, (2) inside the parts that have to be merged the copies take the same values for bisimilar states, (3) the original clocks become irrelevant as soon as a region that has to be merged is entered. $\square$

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. Minimization of timed transition systems. In W.R. Cleaveland, editor, *Proceedings CONCUR 92,* Stony Brook, NY, USA, volume 630 of *Lecture Notes in Computer Science*, pages 340–354. Springer-Verlag, 1992.
2. R. Alur, C. Courcoubetis, and T.A. Henzinger. The observational power of clocks. In B. Jonsson and J. Parrow, editors, *Proceedings CONCUR 94,* Uppsala, Sweden, volume 836 of *Lecture Notes in Computer Science*, pages 162–177. Springer-Verlag, 1994.
3. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
4. T.A. Henzinger, P.W. Kopke, and H. Wong-Toi. The expressive power of clocks. In *Proceedings* 22^{nd} *ICALP*, volume 944 of *Lecture Notes in Computer Science*, pages 417–428. Springer-Verlag, 1995.
5. T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111:193–244, 1994.
6. Z. Kohavi. *Switching and Finite Automata Theory.* McGraw-Hill, Inc., 1970.
7. K.G. Larsen, F. Laroussinie, and C. Weise. From timed automata to logic — and back. In *Proceedings 20th International Symposium on Mathematical Foundations of Computer Science (MFCS'95),* Prague, Czech Republic, volume 969 of *Lecture Notes in Computer Science.* Springer-Verlag, 1995.
8. R. Milner. *Communication and Concurrency.* Prentice-Hall International, Englewood Cliffs, 1989.
9. K. Čerāns. *Algorithmic Problems in Analysis of Real Time System Specifications.* Dr.sc.comp. thesis, University of Latvia, Rīga, 1992.
10. K. Čerāns. Decidability of bisimulation equivalences for parallel timer processes. In G. v. Bochmann and D.K. Probst, editors, *Proceedings of the 4th International Workshop on Computer Aided Verification,* Montreal, Canada, volume 663 of *Lecture Notes in Computer Science*, pages 302–315. Springer-Verlag, 1992.
11. M. Yannakakis and D. Lee. An efficient algorithm for minimizing real-time transition systems. In C. Courcoubetis, editor, *Proceedings of the 5th International Conference on Computer Aided Verification,* Elounda, Greece, volume 697 of *Lecture Notes in Computer Science*, pages 210–224. Springer-Verlag, 1993.