Parameterized Verification of Timed Networks with Clock Invariants

Étienne André
Université Sorbonne Paris Nord, LIPN, CNRS UMR 7030, F-93430 Villetaneuse, France
Institut Universitaire de France (IUF), Paris, France

Swen Jacobs
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Shyam Lal Karra
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Ocan Sankur
Université de Rennes, CNRS, Inria, Rennes, France

Abstract

We consider parameterized verification problems for networks of timed automata (TAs) based on different communication primitives. To this end, we first consider disjunctive timed networks (DTNs), i.e., networks of TAs that communicate via location guards that enable a transition only if there is another process in a certain location. We solve for the first time the case with unrestricted clock invariants, and establish that the parameterized model checking problem (PMCP) over finite local traces can be reduced to the corresponding model checking problem on a single TA. Moreover, we prove that the PMCP for networks that communicate via lossy broadcast can be reduced to the PMCP for DTNs. Finally, we show that for networks with k-wise synchronization, and therefore also for timed Petri nets, location reachability can be reduced to location reachability in DTNs. As a consequence we can answer positively the open problem from Abdulla et al. (2018) whether the universal safety problem for timed Petri nets with multiple clocks is decidable.

2012 ACM Subject Classification Theory of computation → Concurrency

Keywords and phrases Networks of Timed Automata, Parameterized Verification, Timed Petri Nets

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.8

Related Version *Full Version*: `https://arxiv.org/abs/2408.05190` [13]

Funding *Étienne André*: Partially supported by ANR BisoUS (ANR-22-CE48-0012).
Shyam Lal Karra: Carried out this work as a member of the Saarbrücken Graduate School of Computer Science.

1 Introduction

Formally reasoning about concurrent systems is difficult, in particular if correctness guarantees should hold regardless of the number of interacting processes – a problem also known as *parameterized verification* [3, 7], since the number of processes is considered a parameter of the system. Parameterized verification is undecidable in general [14] and even in very restricted settings, e.g., for safety properties of finite-state processes with rather weak communication primitives, such as token-passing or transition guards [32, 23]. A long line of research has identified classes of systems and properties for which parameterized verification is decidable [23, 28, 24, 25, 22, 18, 9, 26], usually with finite-state processes.

Timed automata (TAs) [8] provide a computational model that combines real-time constraints with concurrency, and are therefore an expressive and widely used formalism to model real-time systems. However, TAs are usually used to model a constant and *fixed* number of system components. When the number n of components is very large or unknown,

considering their static combination becomes highly impractical, or even impossible if n is unbounded. However, there are several lines of research studying networks with a parametric number of timed components (see e.g., [6, 17, 4, 11, 1, 10]).

One of these lines considers different variants of timed Petri nets (we here consider the version defined in [2]), and networks of timed automata with k-wise synchronization [6, 5], a closely related model. Due to the expressiveness of the synchronization primitive, results for these models are often negative or limited to severely restricted cases. For example, in networks of timed automata with a controller process and multiple clocks per process, location reachability is undecidable (even in the absence of clock invariants that could force a process to leave a location) [5]. The problem is decidable with a single clock per process and without clock invariants [6]. Decidability remains open for location reachability in networks *without* a controller process and with multiple clocks (with or without clock invariants), which is equivalent to the universal safety problem for timed Petri nets that is mentioned as an open problem in [2].

Another model that has received attention recently and is very important for the work we present is that of *Disjunctive Timed Networks* (DTNs) [31, 12]. It combines the expressive formalism of TAs with the relatively weak communication primitive of disjunctive guards [23]: transitions can be guarded with a location (called "guard location"), and such a transition can only be taken by a TA in the network if another process is in that location upon firing. Consider the example in Figure 1 which illustrates a process's behavior within an asynchronous communication system, where tasks can be dynamically posted and data is read through shared input channels. The transition from init to reading is guarded by location post: for a process to take this transition, at least one other process must be in post. Parameterized model checking of DTNs was first studied in [31], who considered local trace

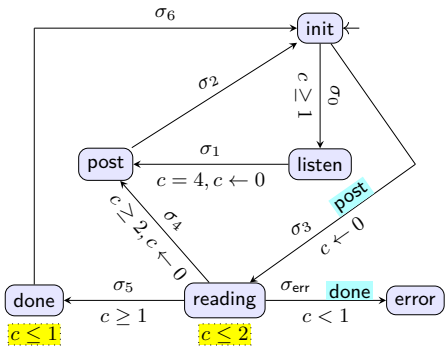

Figure 1 Asynchronous data read example

properties in the temporal logic MITL, and showed that the problem can be solved with a cutoff, i.e., a number of processes that is sufficient to determine satisfaction in networks of any size. However, their result is restricted to the case when guard locations do not have clock invariants. This restriction is crucial to their proof, and they furthermore showed that statically computable cutoffs do not exist for the case when TAs can have clock invariants on all locations.

However, the non-existence of cutoffs does not imply that the problem is undecidable. In [12], the authors improved the aforementioned results by avoiding the construction of a cutoff system and instead using a modified zone graph algorithm. Moreover, they gave sufficient conditions on the TAs to make the problem decidable even in the presence of clock invariants on guard locations. However, these conditions are semantic, and it is not obvious

how to build models that satisfy them; for instance, our motivating example in Figure 1 does not satisfy them. The decidability of the case without restrictions on clock invariants thus remained open.

In this paper, we show that properties of finite local traces (and therefore also location reachability) are decidable for DTNs without restrictions on clock invariants. Moreover, we show that checking local trace properties of systems with lossy broadcast communication [22, 11] or with k-wise synchronization can be reduced to checking local trace properties of DTNs. Note that our simulation of these systems by DTNs crucially relies on the power of clock invariants, and would not be possible in the previous restricted variants of DTNs.

To see why checking local trace properties of DTNs with invariants is technically difficult, consider first the easy case from [31], where guard locations cannot have invariants. In this case, it is enough to determine for every guard location q the minimal time δ at which it can be reached: since a process cannot be forced to leave, q can be occupied at any time in $[\delta, \infty)$, and transitions guarded by q can be assumed to be enabled at any time later than δ. This is already the underlying insight of [31], and in [12] it is embedded into a technique that replaces location guards with clock guards $t \geq \delta$, where t is a clock denoting the time elapsed since the beginning. In contrast, if guard locations can have invariants, a process in q can be forced to leave after some time. Therefore, the set of global times where q can be occupied is an arbitrary set of timepoints, and it is not obvious how it can be finitely represented.

Detailed Example. We introduce an example that motivates the importance of clock invariants in modeling concurrent timed systems, and will be used as a running example. It is inspired by the verification of asynchronous programs [27]. In this setting, processes can be "posted" at runtime to solve a task, and will terminate upon completing the task. Our example in Figure 1 features one clock c per process; symbols σ_i and σ_{err} are transition labels. An unbounded number of processes start in the initial location init. In the inner loop, a process can move to location listen in order to see whether an input channel carries data. Once it determines that this is the case, it moves to post, thereby giving the command to post a process that reads the data, and then can return to init. In the outer loop, if another process gives the command to read data, i.e., is in post, then another process can accept that command and move to reading. After some time, the process will either determine that all the data has been read and move to done, or it will timeout and move to post to ask another process to carry on reading. However, this scheme may run into an error if there are processes in done and reading at the same time, modeled by a transition from reading to error that can only be taken if done is occupied.

While this example is relatively simple, checking reachability of location error (in a network with arbitrarily many processes) is not supported by any existing technique. This is because clock invariants on guard locations are not supported at all by [31], and are supported only in special cases (that do not include this example) by [12]. Also other results that could simulate DTNs do not support clock invariants at all [6, 4].

Moreover, note that clock invariants may be essential for correctness of such systems: in a system A^3, consisting of three copies of the automaton in Figure 1, location error is reachable; a computation that reaches error is given in Figure 2. However, if we add a clock invariant $c \leq 0$ to location post (forcing processes that enter post to immediately leave it again), it becomes unreachable[1].

[1] To see this, consider the intervals of global time in which the different locations can be occupied: first observe that post in the inner loop can now only be occupied in intervals $[4k, 4k]$ (for $k \in \mathbb{N}$), and therefore processes can only move into reading at these times. From there, they might move into post after two time units, so overall post can be occupied in intervals $[2k, 2k]$ for $k \geq 2$, and reading in any

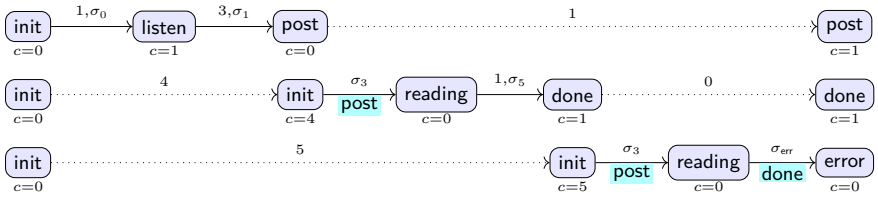

Figure 2 Example of a computation in A^3 for A as depicted by Figure 1.

Contributions. We present new decidability results for parameterized verification problems with respect to three different system models as outlined below.

- **DTNs (Section 3).** For DTNs, we show that, surprisingly, and despite the absence of cutoffs [31], the parameterized model checking problem for finite local traces is decidable in the general case, without any restriction on clock invariants. Our technique circumvents the non-existence of cutoffs by constructing a modified region automaton, a well-known data structure in timed automata literature, such that communication via disjunctive guards is directly taken into account. In particular, we focus on analyzing the traces of a single (or a finite number of) process(es) in a network of arbitrary size.

  While our algorithm uses some techniques from [12], there are fundamental differences: in particular, we introduce a novel abstraction of global time into a finite number of "slots", which are elementary intervals with integer bounds, designed to capture the information necessary for disjunctive guard communication. When a transition with a location guard is to be taken at a given slot, we check whether the given guard location appears in the same slot. It turns out that such an abstract treatment of the global time is sound: we prove that in this case, one can find a computation that enables the location guard at *any* real time instant inside the given slot. Thus, the infinite set of points at which a location guard is enabled is a computable union of intervals; and we rely on this property to build a finite-state abstraction to solve our problem.

- **Lossy Broadcast Timed Networks (Section 4).** We investigate the relation between communication with disjunctive guards and with *lossy broadcast* [22, 11]. For finite-state processes, it is known that lossy broadcast can simulate disjunctive guards wrt. reachability [15], but the other direction is unknown.[2] As our second contribution, we establish the decidability of the parameterized model checking problem for local trace properties in timed lossy broadcast networks. This result is obtained by proving that communication by lossy broadcast is equivalent to communication by disjunctive guards for networks of timed automata *with* clock invariants for local trace properties.

- **Synchronizing Timed Networks and Timed Petri Nets (Section 5).** Finally, we show that the location reachability problem for controllerless multi-clock timed networks with k-wise synchronization can be reduced to the location reachability problem for DTNs with clock invariants.

---

interval $[2k, 2(k+1)]$. Since clock c is always reset upon entering reading, done can only be occupied in intervals $[2k+1, 2k+1]$ for $k \geq 2$, whereas for a process in reading the clock constraint on the transition to error can only be satisfied in intervals $[2k, 2k+1)$. Therefore, error is not reachable with the additional clock invariant on post.

[2] [15] considers IO nets which are equivalent to systems with disjunctive guards. It gives a negative result for a specific simulation relation, but does not prove that simulation is impossible in general.

As a consequence, it follows that the universal safety problem for timed Petri nets with multiple clocks, stated as an open problem in [2], is also decidable.

The proofs of the last two points above involve constructions that require clock invariants on guard locations. This is why clock invariants are crucial in our formalism, which is a nontrivial extension of [12]. Note that in both cases we get decidability even for variants of the respective system models with clock invariants, not considered in [2, 11].

For all of the above systems, location reachability can be decided in EXPSPACE.

Due to space constraints, some details of the definitions and full proofs of some of our results are deferred to the appendix of the extended version [13].

2 Preliminaries

Let $\mathcal{C}$ be a set of clock variables, also called *clocks*. A *clock valuation* is a mapping $v : \mathcal{C} \to \mathbb{R}_{\geq 0}$. For a valuation v and a clock c, we denote the fractional and integral parts of $v(c)$ by $\mathsf{frac}(v(c))$ and $\lfloor v(c) \rfloor$ respectively. We denote by $\mathbf{0}$ the clock valuation that assigns 0 to every clock, and by $v + \delta$ for $\delta \in \mathbb{R}_{\geq 0}$ the valuation s.t. $(v + \delta)(c) = v(c) + \delta$ for all $c \in \mathcal{C}$. Given a subset $\mathcal{C}_r \subseteq \mathcal{C}$ of clocks and a valuation v, $v[\mathcal{C}_r \leftarrow 0]$ denotes the valuation v' such that $v'(c) = 0$ if $c \in \mathcal{C}_r$ and $v'(c) = v(c)$ otherwise. We call *clock constraints* $\Psi(\mathcal{C})$ the terms of the following grammar: $\psi ::= \top \mid \psi \wedge \psi \mid c \sim d \mid c \sim c' + d$ with $d \in \mathbb{N}$, $c, c' \in \mathcal{C}$ and $\sim \in \{<, \leq, =, \geq, >\}$.

A clock valuation v *satisfies* a clock constraint ψ, denoted by $v \models \psi$, if ψ evaluates to $\top$ after replacing every $c \in \mathcal{C}$ with its value $v(c)$.

▶ **Definition 1.** *A* timed automaton (TA) *A is a tuple $(Q, \hat{q}, \mathcal{C}, \Sigma, \mathcal{T}, Inv)$ where Q is a finite set of locations with* initial location *$\hat{q}$, $\mathcal{C}$ is a finite set of clocks, Σ is a finite alphabet that contains a subset Σ^- of special symbols, including a distinguished symbol $\epsilon \in \Sigma^-$, $\mathcal{T} \subseteq Q \times \Psi(\mathcal{C}) \times 2^{\mathcal{C}} \times \Sigma \times Q$ is a transition relation, and $Inv : Q \to \Psi(\mathcal{C})$ assigns to every location q a* clock invariant *$Inv(q)$.*

TAs were introduced in [8] and clock invariants, also simply called invariants, in [29]. We assume w.l.o.g. that invariants only contain upper bounds on clocks (as lower bounds can be moved into the guards of incoming transitions). Σ^- will be used to label silent transitions and unless explicitly specified otherwise (in Sections 4 and 5), we assume that $\Sigma^- = \{\epsilon\}$.

▶ **Example 2.** If we ignore the location guards post (from init to reading) and done (from reading to error), then the automaton in Figure 1 is a classical TA with one clock c. For example, the invariant of done is $c \leq 1$ and the transition from init to reading resets clock c.

A *configuration* of a TA A is a pair (q, v), where $q \in Q$ and $v : \mathcal{C} \to \mathbb{R}_{\geq 0}$ is a clock valuation. A *delay transition* is of the form $(q, v) \xrightarrow{\delta} (q, v + \delta)$ for some delay $\delta \in \mathbb{R}_{\geq 0}$ such that $v + \delta \models Inv(q)$. A *discrete transition* is of the form $(q, v) \xrightarrow{\sigma} (q', v')$, where $\tau = (q, g, \mathcal{C}_r, \sigma, q') \in \mathcal{T}$, $v \models g$, $v' = v[\mathcal{C}_r \leftarrow 0]$ and $v' \models Inv(q')$. A transition $(q, v) \xrightarrow{\epsilon} (q', v')$ is called an *ϵ-transition*. We write $(q, v) \xrightarrow{\delta,\sigma} (q', v')$ if there is a delay transition $(q, v) \xrightarrow{\delta} (q, v + \delta)$ followed by a discrete transition $(q, v + \delta) \xrightarrow{\sigma} (q', v')$.

A *timed path* of A is a finite sequence of transitions $\rho = (q_0, v_0) \xrightarrow{\delta_0,\sigma_0} \dots \xrightarrow{\delta_{l-1},\sigma_{l-1}} (q_l, v_l)$. For a timed path $\rho = (q_0, v_0) \xrightarrow{\delta_0,\sigma_0} \dots \xrightarrow{\delta_{l-1},\sigma_{l-1}} (q_l, v_l)$, let $\delta(\rho) = \sum_{0 \leq i < l} \delta_i$ be the *total time delay* of ρ. The *length* of ρ is $2l$. A configuration (q, v) has a *timelock* if there is $b \in \mathbb{R}_{\geq 0}$ s.t. $\delta(\rho) \leq b$ for every timed path ρ starting in (q, v). We write $(q_0, v_0) \to^* (q_l, v_l)$ if there is a timed path $\rho = (q_0, v_0) \xrightarrow{\delta_0,\sigma_0} \dots \xrightarrow{\delta_{l-1},\sigma_{l-1}} (q_l, v_l)$; ρ is a *computation* if $q_0 = \hat{q}$ and $v_0 = \mathbf{0}$.

The *trace* of the timed path ρ is the sequence of pairs of delays and labels obtained by removing transitions with a label from Σ^- and adding the delays of these to the following transition. The *language* of A, denoted $\mathcal{L}(A)$, is the set of traces of all of its computations.

We now recall guarded timed automata as an extension of timed automata with location guards, that will allow, in a network, to test whether some other process is in a given location in order to pass the guard.

▶ **Definition 3** (Guarded Timed Automaton (GTA)). *A* GTA A *is a tuple* $(Q, \hat{q}, \mathcal{C}, \Sigma, \mathcal{T}, Inv)$*, where* Q *is a finite set of locations with* initial location $\hat{q}$*,* $\mathcal{C}$ *is a finite set of clocks,* Σ *is a finite alphabet that contains a subset* Σ^- *of special symbols, including a distinguished symbol* $\epsilon \in \Sigma^-$*,* $\mathcal{T} \subseteq Q \times \Psi(\mathcal{C}) \times 2^{\mathcal{C}} \times \Sigma \times (Q \cup \{\top\}) \times Q$ *is a transition relation, and* $Inv : Q \to \Psi(\mathcal{C})$ *assigns to every location* q *an* invariant $Inv(q)$*.*

Intuitively, a transition $\tau = (q, g, \mathcal{C}_r, \sigma, \gamma, q') \in \mathcal{T}$ takes the automaton from location q to q'; τ can only be taken if *clock guard* g and *location guard* γ are both satisfied, and it resets all clocks in $\mathcal{C}_r$. Note that satisfaction of location guards is only meaningful in a *network* of TAs (defined below). Intuitively, a location guard γ is satisfied if it is $\top$ or if another automaton in the network currently occupies location γ. We say that γ is *trivial* if $\gamma = \top$. We say location q *has no invariant* if $Inv(q) = \top$.

▶ **Example 4.** In the GTA in Figure 1, the transition from init to reading is guarded by location guard post. The transition from init to listen has a trivial location guard (trivial location guards are not depicted in our figures). Location init has no invariant.

▶ **Definition 5** (Unguarded Timed Automaton). *Given a GTA* A*, we denote by* $\mathsf{UG}(A)$ *the* unguarded *version of* A*, which is the TA obtained from* A *by removing location guards, and adding a fresh clock* t*, called the global clock, that does not appear in the guards or resets. Formally,* $\mathsf{UG}(A) = (Q, \hat{q}, \mathcal{C} \cup \{t\}, \mathcal{T}', Inv)$ *with* $\mathcal{T}' = \big\{(q, g, \mathcal{C}_r, \sigma, q') \mid (q, g, \mathcal{C}_r, \sigma, \gamma, q') \in \mathcal{T}\big\}$*.*

For a GTA A, let A^n denote a *network of guarded timed automata* (NGTA), consisting of n copies of A. Each copy of A in A^n is called a *process*.

A *configuration* $\mathfrak{c}$ of an NGTA A^n is a tuple $\big((q_1, v_1), \ldots, (q_n, v_n)\big)$, where every (q_i, v_i) is a configuration of A. The semantics of A^n can be defined as a *timed transition system* $(\mathfrak{C}, \hat{\mathfrak{c}}, T)$, where $\mathfrak{C}$ denotes the set of all configurations of A^n, $\hat{\mathfrak{c}}$ is the unique initial configuration $(\hat{q}, \mathbf{0})^n$, and the transition relation T is the union of the following delay and discrete transitions:

delay transition $\big((q_1, v_1), \ldots, (q_n, v_n)\big) \xrightarrow{\delta} \big((q_1, v_1 + \delta), \ldots, (q_n, v_n + \delta)\big)$ if $\forall i \in \{1, \ldots, n\}$: $v_i + \delta \models Inv(q_i)$, i.e., we can delay $\delta \in \mathbb{R}_{\geq 0}$ units of time if all clock invariants are satisfied at the end of the delay.

discrete transition $\big((q_1, v_1), \ldots, (q_n, v_n)\big) \xrightarrow{(i,\sigma)} \big((q_1', v_1'), \ldots, (q_n', v_n')\big)$ for some $i \in \{1, \ldots, n\}$ if

1) $(q_i, v_i) \xrightarrow{\sigma} (q_i', v_i')$ is a discrete transition of A with $\tau = (q_i, g, \mathcal{C}_r, \sigma, \gamma, q_i')$,

2) $\gamma = \top$ or $q_j = \gamma$ for some $j \in \{1, \ldots, n\} \setminus \{i\}$, and

3) $q_j' = q_j$ and $v_j' = v_j$ for all $j \in \{1, \ldots, n\} \setminus \{i\}$.

That is, location guards γ are interpreted as disjunctive guards: unless $\gamma = \top$, at least one other process needs to occupy location γ in order for process i to pass this guard.

We write $\mathfrak{c} \xrightarrow{\delta,(i,\sigma)} \mathfrak{c}''$ for a delay transition $\mathfrak{c} \xrightarrow{\delta} \mathfrak{c}'$ followed by a discrete transition $\mathfrak{c}' \xrightarrow{(i,\sigma)} \mathfrak{c}''$. Then, a *timed path* of A^n is a finite sequence $\pi = \mathfrak{c}_0 \xrightarrow{\delta_0,(i_0,\sigma_0)} \cdots \xrightarrow{\delta_{l-1},(i_{l-1},\sigma_{l-1})} \mathfrak{c}_l$.

For a timed path $\pi = \mathfrak{c}_0 \xrightarrow{\delta_0,(i_0,\sigma_0)} \cdots \xrightarrow{\delta_{l-1},(i_{l-1},\sigma_{l-1})} \mathfrak{c}_l$, let $\delta(\pi) = \sum_{0 \le i < l} \delta_i$ be the total time delay of π. The definition of timelocks extends naturally to configurations of NGTAs. A timed path π of A^n is a *computation* if $\mathfrak{c}_0 = \hat{\mathfrak{c}}$. Its *length* is equal to $2l$.

We write $q \in \mathfrak{c}$ if $\mathfrak{c} = \big((q_1, v_1), \ldots, (q_n, v_n)\big)$ and $q = q_i$ for some $i \in \{1, \ldots, n\}$, and similarly $(q, v) \in \mathfrak{c}$. We say that a location q is *reachable* in A^n if there exists a reachable configuration $\mathfrak{c}$ s.t. $q \in \mathfrak{c}$.

▶ **Example 6.** Consider the NGTA A^3 where A is the GTA shown in Figure 1. A computation π of this network is depicted in Figure 2, in which a process reaches error with $\delta(\pi) = 5$. The computation is $\big((\mathsf{init}, c = 0), (\mathsf{init}, c = 0), (\mathsf{init}, c = 0)\big) \xrightarrow{1,(1,\sigma_0)} \big((\mathsf{listen}, c = 1), (\mathsf{init}, c = 1), (\mathsf{init}, c = 1)\big) \xrightarrow{3,(1,\sigma_1)} \big((\mathsf{post}, c = 0), (\mathsf{init}, c = 4), (\mathsf{init}, c = 4)\big) \xrightarrow{0,(2,\sigma_3)} \big((\mathsf{post}, c = 0), (\mathsf{reading}, c = 0), (\mathsf{init}, c = 4)\big) \xrightarrow{1,(2,\sigma_5)} \big((\mathsf{post}, c = 1), (\mathsf{done}, c = 1), (\mathsf{init}, c = 5)\big) \xrightarrow{0,(3,\sigma_3)} \big((\mathsf{post}, c = 1), (\mathsf{done}, c = 1), (\mathsf{reading}, c = 0)\big) \xrightarrow{0,(3,\sigma_{\mathsf{err}})} \big((\mathsf{post}, c = 1), (\mathsf{done}, c = 1), (\mathsf{error}, c = 0)\big)$. Therefore, error is reachable in A^3.

The *trace* of the timed path π is a sequence $\mathsf{trace}(\pi) = (\delta'_0, (i'_0, \sigma'_0)) \ldots (\delta'_{l-1}, (i'_{l-1}, \sigma'_{l-1}))$ obtained by removing all discrete transitions (j, σ_j) of π with $\sigma_j \in \Sigma^-$, and adding all delays of these transitions to the following discrete transition, yielding the δ'_j. The *language* of A^n, denoted $\mathcal{L}(A^n)$, is the set of traces of all of its computations.

▶ **Example 7.** For the computation π in Example 6, $\mathsf{trace}(\pi) = \big(1, (1, \sigma_0)\big), \big(3, (1, \sigma_1)\big), \big(0, (2, \sigma_3)\big), \big(1, (2, \sigma_5)\big), \big(0, (3, \sigma_3)\big), \big(0, (3, \sigma_{\mathsf{err}})\big)$.

We will also use *projections* of these global objects onto subsets of the processes. That is, if $\mathfrak{c} = \big((q_1, v_1), \ldots, (q_n, v_n)\big)$ and $\mathcal{I} = \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$, then $\mathfrak{c}{\downarrow_{\mathcal{I}}}$ is the tuple $\big((q_{i_1}, v_{i_1}), \ldots, (q_{i_k}, v_{i_k})\big)$, and we extend this notation to computations $\pi{\downarrow_{\mathcal{I}}}$ by keeping only the discrete transitions of $\mathcal{I}$ and by adding the delays of the removed discrete transitions to the delay of the following discrete transition of $\mathcal{I}$.

We introduce a special notation for projecting to a single process and define, for any natural number $1 \le a \le n$, $\pi{\downarrow_a}$ a computation of $\mathsf{UG}(A)$, obtained from $\pi{\downarrow_{\{a\}}}$ by discarding the index a from all transitions; that is, $\pi{\downarrow_a}$ has the form $(q_0, v_0) \xrightarrow{(\delta'_{k_0}, \sigma_{k_0})} \ldots \xrightarrow{(\delta'_{k_m}, \sigma_{k_m})} (q_{m+1}, v_{m+1})$. We also extend this to traces; that is, $\mathsf{trace}(\pi){\downarrow_a} = (\delta'_{k_0}, \sigma_{k_0}) \ldots (\delta'_{k_m}, \sigma_{k_m})$, which is a trace of $\mathsf{UG}(A)$. For a set of traces L, and set $\mathcal{I}$ of processes, we write $L{\downarrow_{\mathcal{I}}} = \{tt{\downarrow_{\mathcal{I}}} \mid tt \in L\}$.

Note that the projection of a computation is not necessarily a computation itself, since location guards may not be satisfied.

▶ **Example 8.** For the computation π in Example 6, $\pi{\downarrow_3} = (\mathsf{init}, c = 0) \xrightarrow{(5,\sigma_3)} (\mathsf{reading}, c = 0) \xrightarrow{(0,\sigma_{\mathsf{err}})} (\mathsf{error}, c = 0)$ and $\mathsf{trace}(\pi){\downarrow_3} = (5, \sigma_3), (0, \sigma_{\mathsf{err}})$.

A *prefix* of a computation $\pi = \mathfrak{c}_0 \xrightarrow{\delta_0,(i_0,\sigma_0)} \cdots \xrightarrow{\delta_{l-1},(i_{l-1},\sigma_{l-1})} \mathfrak{c}_l$, is a sequence $\mathfrak{c}_0 \xrightarrow{\delta_0,(i_0,\sigma_0)} \cdots \xrightarrow{\delta_{l'},(i_{l'},\sigma_{l'})} \mathfrak{c}_{l'}$ with $l' \le l - 1$. If π is a timed path and $d \in \mathbb{R}_{\ge 0}$, then $\pi^{\le d}$ denotes the maximal prefix of π with $\delta(\pi^{\le d}) \le d$, and similarly for timed paths $\rho^{\le d}$ of a single GTA. For timed paths π_1 of A^{n_1} and π_2 of A^{n_2} with $\delta(\pi_1) = \delta(\pi_2)$, we denote by $\pi_1 \parallel \pi_2$ their *composition* into a timed path of $A^{n_1+n_2}$ whose projection to the first n_1 processes is π_1, and whose projection to the last n_2 processes is π_2.

▶ **Definition 9** (Disjunctive Timed Network)**.** *A given GTA A induces a* disjunctive timed network (DTN) *A^∞, defined as the following family of NGTAs: $A^\infty = \{A^n \mid n \in \mathbb{N}_{>0}\}$ (we follow the terminology and use abbreviations of [31]). We define $\mathcal{L}(A^\infty) = \bigcup_{n \in \mathbb{N}_{>0}} \mathcal{L}(A^n)$ and consider $\mathcal{L}(A^\infty){\downarrow_I} = \bigcup_{n \in \mathbb{N}_{>0}} \mathcal{L}(A^n){\downarrow_I}$.*

2.1 The Parameterized Model Checking Problem

We formalize properties of DTNs as sets of traces that describe the intended behavior of a fixed number of processes running in a system with arbitrarily many processes. That is, a *local property* φ of k processes, also called a *k-indexed property*, is a subset of $(\mathbb{R}_{\geq 0} \times ([1,k] \times \Sigma))^*$. For $k = 1$, for simplicity, we consider it as a subset of $(\mathbb{R}_{\geq 0} \times \Sigma)^*$. We say that A^n *satisfies* a k-indexed local property φ, denoted $A^n \models \varphi$, if $\mathcal{L}(A^n)\downarrow_{[1,k]} \subseteq \varphi$. Note that, due to the symmetry of the system, it does not matter *which* k processes we project $\mathcal{L}(A^n)$ onto, so we always project onto the first k.

Parameterized model checking problem (PMCP):
INPUT: a GTA A and a k-indexed local property φ
PROBLEM: Decide whether $A^n \models \varphi$ holds $\forall n \geq k$.

Local trace properties allow to specify for instance any local safety property of a single process (with $I = [1,1]$), as well as mutual exclusion properties (with $I = [1,2]$) and variants of such properties for larger I.

PMCP can be solved by checking whether $\mathcal{L}(A^\infty)\downarrow_{[1,k]} \subseteq \varphi$. Our solution consists in building a TA that recognizes $\mathcal{L}(A^\infty)\downarrow_{[1,k]}$. Note that language inclusion is undecidable on TAs [8], but many interesting problems are decidable. These include MITL model checking [21] and simpler problems such as reachability: given symbol $\sigma_0 \in \Sigma$, the *reachability PMCP* is the PMCP where φ is the set of traces that contain an occurrence of σ_0. Reachability of a *location* of A can be solved by PMCP by choosing appropriate transition labels.

▶ **Example 10.** In the example of Figure 1, a natural local property we are interested in is the reachability of the label σ_{err}. Formally, the local property for process 1 can be written as a 1-indexed property: $(\mathbb{R}_{\geq 0} \times ([1,1] \times \Sigma))^* \cdot \{(d, (1, \sigma_{\mathsf{err}})) \mid d \in \mathbb{R}_{\geq 0}\} \cdot (\mathbb{R}_{\geq 0} \times ([1,1] \times \Sigma))^*$.

3 Model Checking DTNs

3.1 Definitions

We recall here the standard notions of regions and region automata, and introduce the *slots* of regions which refer to the intervals of possible valuations of a global clock.

Regions. Given A, for all $c \in \mathcal{C}$, let $M(c)$ denote the maximal bound that c is compared to: $M(c) = \max\{d \in \mathbb{Z} \mid$ "$c \sim d$", "$c - c' \sim d$", "$c' - c \sim d$" appears in a guard or invariant of $A\}$(we set $M(c) = 0$ if this set is empty). M is called the *maximal bound function for* A. Define $M_{\max} = \max\{M(c) \mid c \in \mathcal{C}\}$. We say that two valuations v and v' are equivalent w.r.t. M, denoted by $v \simeq_M v'$, if the following conditions hold for any clocks c, c' [19, 20]:

1. either $\lfloor v(c) \rfloor = \lfloor v'(c) \rfloor$ or $v(c) > M(c)$ and $v'(c) > M(c)$;
2. if $v(c), v'(c) \leq M(c)$ then $\mathsf{frac}(v(c)) = 0 \iff \mathsf{frac}(v'(c)) = 0$;
3. if $v(c) \leq M(c), v(c') \leq M(c')$ then $\mathsf{frac}(v(c)) \leq \mathsf{frac}(v(c')) \iff \mathsf{frac}(v'(c)) \leq \mathsf{frac}(v'(c'))$;
4. for any interval I among $(-\infty, -M(c')), [-M(c'), -M(c')], (-M(c'), -M(c') + 1), \ldots, [M(c), M(c)], (M(c), \infty)$, we have $v(c) - v(c') \in I \iff v'(c) - v'(c') \in I$.

An *M-region* is an equivalence class of valuations induced by $\simeq_M$. We denote by $[v]_M$ the region to which v belongs. We omit M when it is clear from context.

For an M-region r, if a valuation $v \in r$ satisfies a clock guard g, then every valuation in r satisfies g. We write $r \models g$ to mean that every valuation in r satisfies g.

Given an M-region r and a clock c, let $r{\downarrow_c}$ denote the projection of the valuations of r to c, i.e., $r{\downarrow_c} = \{v(c) \mid v \in r\}$. Given a valuation v and a clock $c \in \mathcal{C}$, let $v{\downarrow_{-c}}$ denote *the projection of v to the clocks other than c*, i.e., $v{\downarrow_{-c}} \colon \mathcal{C} \setminus \{c\} \to \mathbb{R}_{\geq 0}$ is defined by $v{\downarrow_{-c}}(c') = v(c')$ for all $c' \in \mathcal{C} \setminus \{c\}$. By extension, given a region r and a clock c, let $r{\downarrow_{-c}} = \{v{\downarrow_{-c}} \mid v \in r\}$.

Region Automaton. The *region automaton* of a TA A is a finite automaton with alphabet $\Sigma \cup \{\mathsf{delay}\}$, denoted by $\mathcal{R}_M(A)$, defined as follows.

The *region states* are pairs (q, r) where $q \in Q$ and r is an M-region. The *initial region state* is $(\hat{q}, \hat{r})$ where $\hat{q}$ is the initial location of A and $\hat{r}$ is the singleton region containing $\mathbf{0}$.

There is a transition $(q, r) \xrightarrow{\mathsf{delay}} (q, r')$ in $\mathcal{R}_M(A)$ iff there is a transition $(q, v) \xrightarrow{\delta} (q, v')$ in A for some $\delta \in \mathbb{R}_{\geq 0}, v \in r$ and $v' \in r'$. We say that r' is a *time successor* of r. Note that we can have $r' = r$. Furthermore, (q, r') is the *immediate time successor* of (q, r) if $(q, r) \xrightarrow{\mathsf{delay}} (q, r')$, $r' \neq r$, and whenever $(q, r) \xrightarrow{\mathsf{delay}} (q, r'')$, we have $(q, r') \xrightarrow{\mathsf{delay}} (q, r'')$.

There is a transition $(q, r) \xrightarrow{\sigma} (q', r')$ in $\mathcal{R}_M(A)$ iff there is a transition $(q, v) \xrightarrow{\sigma} (q', v')$ with label σ in A for some $v \in r$ and $v' \in r'$. We write $(q, r) \to (q', r')$ if either $(q, r) \xrightarrow{\mathsf{delay}} (q', r')$ or $(q, r) \xrightarrow{\sigma} (q', r')$ for some $\sigma \in \Sigma$.

A *path* in $\mathcal{R}_M(A)$ is a finite sequence of transitions $\rho_r = (q_0, r_0) \xrightarrow{\sigma_0} \dots \xrightarrow{\sigma_{n-1}} (q_n, r_n)$ for some $n \geq 0$ where $\sigma_i \in \Sigma \cup \{\mathsf{delay}\}$. A path of $\mathcal{R}_M(A)$ is a *computation* if it starts from the initial region state.

It is known that $\mathcal{R}_M(A)$ captures the *untimed* traces of A, i.e., the projection of the traces of A to Σ [8].

Slots. Now, we can introduce slots. We will show later that slots are a sufficiently precise abstraction of time for DTNs. In this paragraph, we assume that TAs have a distinguished *global clock* t which is never reset and does not appear in clock guards. We will thus consider a clock set $\mathcal{C} \cup \{t\}$ (making t appear explicitly for clarity).

Let N_A denote the number of pairs (q, r) where $q \in Q$ and r is an M-region (thus a region on the clock set $\mathcal{C}$ without t). Recall that N_A is exponential in $|\mathcal{C}|$ [8, 20]. Let us consider a bound function $M^{\nearrow} : \mathcal{C} \cup \{t\} \to \mathbb{N}$ for A such that for $c \in \mathcal{C} \setminus \{t\}$, $M^{\nearrow}(c) = M(c)$, and $M^{\nearrow}(t) = 2^{N_A+1}$. Throughout the paper, the bound functions will be denoted by $M^{\nearrow}(\cdot)$ whenever the clock set contains the distinguished global clock t, and $M(\cdot)$ otherwise. The former will be referred to as $M^{\nearrow}$-regions, and the latter as M-regions.

We define the *slot of an $M^{\nearrow}$-region r* as $\mathsf{slot}(r) = r{\downarrow_t}$. It is known that for any region r (with any bound function) and clock c, $r{\downarrow_c}$ is an interval [30]. Moreover, if $v(c)$ for every $v \in r$ is below the maximal constant $M^{\nearrow}(c)$, then $r{\downarrow_c}$ is either a singleton interval of the form $[k, k]$, or an open interval of the form $(k, k+1)$ for some $k \in \mathbb{N}$.

For a slot s, let us define $\mathsf{next}(s)$ as follows.

1. if $s = (k, k+1)$ for some $k \in \mathbb{N}$, then $\mathsf{next}(s) = [k+1, k+1]$;
2. if $s = [k, k]$ and $k < M^{\nearrow}(t)$, then $\mathsf{next}(s) = (k, k+1)$;
3. if $s = [M^{\nearrow}(t), M^{\nearrow}(t)]$, then $\mathsf{next}(s) = (M^{\nearrow}(t), \infty)$.
4. if $s = (M^{\nearrow}(t), \infty)$ then $\mathsf{next}(s) = s$.

We define the *slot of a valuation v* on $\mathcal{C} \cup \{t\}$ as $\mathsf{slot}(r)$ where r is the (unique) $M^{\nearrow}$-region s.t. $v \in r$. Slots, seen as intervals, can be bounded or unbounded.

▶ **Example 11.** Consider the clock set $\{x, y, t\}$ and the region r defined by $\lfloor x \rfloor = \lfloor y \rfloor = 1$, $\lfloor t \rfloor = 2$, $0 < \mathsf{frac}(x) < \mathsf{frac}(y) < \mathsf{frac}(t) < 1$ (with $M^{\nearrow}(\cdot) = 4$ for all clocks). Then, $\mathsf{slot}(r) = (2, 3)$.

As a second example, assume $M^{\nearrow}(x) = 2$, $M^{\nearrow}(y) = 3$ and $M^{\nearrow}(t)$ is, say, 2048. Consider the region r' defined by $x > 2 \wedge \lfloor y \rfloor = 1 \wedge 0 < \mathsf{frac}(y) < 1 \wedge \lfloor t \rfloor = 2048 \wedge \mathsf{frac}(t) = 0$. Then, $\mathsf{slot}(r) = [2048, 2048]$. In addition, $\mathsf{next}(\mathsf{slot}(r)) = (2048, \infty)$.

We now introduce the *shifting* operation which consists of increasing the global clock value, without changing the values of other clocks.

▶ **Lemma 12.** *Given any $M^{\nearrow}$-region r and $k \in \mathbb{Z}$ such that $\sup(\mathsf{slot}(r)) + k \leq M^{\nearrow}(t)$, and $\inf(\mathsf{slot}(r)) + k \geq 0$, there exists a $M^{\nearrow}$-region r' which satisfies $\mathsf{slot}(r') = \mathsf{slot}(r) + k$ and $r'{\downarrow_{-t}} = r{\downarrow_{-t}}$, and r' can be computed in polynomial time in the number of clocks.*

The region r' in Lemma 12 will be denoted by $r_{\mathsf{slot}+k}$. We say that it is obtained by *shifting the slot by k in r*. We extend this notation to sets of regions and sets of region states, that is, $W_{\mathsf{slot}+k} = \{(q, r_{\mathsf{slot}+k}) \mid (q, r) \in W\}$ where W is a set of region states. For a set of region states W, we define $W{\downarrow_{-t}} = \{(q, r{\downarrow_{-t}}) \mid (q, r) \in W\}$.

▶ **Example 13.** Consider the clock set $\{x, y, t\}$ and the region r defined in Example 11 satisfying $\lfloor x \rfloor = \lfloor y \rfloor = 1$, $\lfloor t \rfloor = 2$, $0 < \mathsf{frac}(x) < \mathsf{frac}(y) < \mathsf{frac}(t) < 1$ (with $M^{\nearrow}_{max} = 4$). Then, $\mathsf{slot}(r) = (2, 3)$, and $r_{\mathsf{slot}+1}$ is defined by the same constraints as above except that $\lfloor t \rfloor = 3$, and $\mathsf{slot}(r_{\mathsf{slot}+1}) = (3, 4)$.

▶ Remark 14. Recall that given a bound function, the number of regions is $O(|\mathcal{C}|!2^{|\mathcal{C}|}{M_{\max}}^{|\mathcal{C}|})$ since regions determine an order of the fractional values of clocks, the subsets of clocks that have integer values, and an integral value for each clock [19]. The number of $M^{\nearrow}$-regions is $O\left(|\mathcal{C}|!2^{|\mathcal{C}|}(M^{\nearrow}(t))^{|\mathcal{C}|}\right)$, which is doubly exponential in $|\mathcal{C}|$ since $M^{\nearrow}(t)$ is.

Crucial to our paper, however, is that the set of *projections* $r{\downarrow_{-t}}$ of the set of $M^{\nearrow}$-regions r has size exponential only. This can be seen as follows: our definition of regions from [19] uses a distinct maximum bound function for each clock. Thus, when constraints on t are eliminated, there only remain constraints on clocks $c \in \mathcal{C} \setminus \{t\}$, with maximal constants $M(c)$ as in the original GTA A. We thus fall back to the set of regions of A of size $O(|\mathcal{C}|!2^{|\mathcal{C}|}{M_{\max}}^{|\mathcal{C}|})$.

3.2 Layer-based Algorithm for the DTN Region Automaton

We describe here an algorithm to compute a TA $\mathcal{S}(A)$ that recognizes the language $\mathcal{L}(A^{\infty}){\downarrow_1}$. We explain at the end of the section how to generalize the algorithm to compute $\mathcal{L}(A^{\infty}){\downarrow_I}$ for an interval $I = [1, a]$ for $a \geq 1$.

▶ **Assumption 1.** *We assume that the given GTA A is* timelock-free*, regardless of location guards. Formally, let A' be obtained from A by removing all transitions with non-trivial location guards. We require that no configuration of A' has a timelock.*

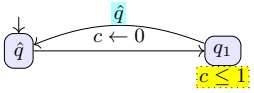

Figure 3 A GTA with timelock due to location guards.

Note that this assumption guarantees that A^n will be timelock-free for all n. Assuming timelock-freeness is not restrictive since a protocol cannot possibly block the physical time: time will elapse regardless of the restrictions of the design. A network with a timelock is thus a design artifact, and just means the model is incomplete. An incomplete model can be completed by adding a sink location to which processes that would cause a timelock can move, and regarding reachability the resulting model is equivalent to the original one.

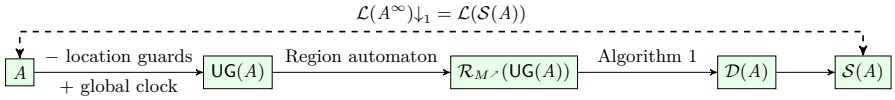

Figure 4 An overview of data structures in the paper.

▶ **Example 15.** The GTA in Figure 3 does not satisfy Assumption 1, since A' (where we remove transitions with non-trivial location guards) has a timelock at $(q_1, c = 1)$.

The following assumption simplifies the proofs:

▶ **Assumption 2.** *Each transition of GTA A is labeled by a unique label different from ϵ.*

Consider a GTA A. Our algorithm builds a TA capturing the language $\mathcal{L}(A^\infty)\downarrow_1$. The construction is based on $M^\nearrow$-region states of $\mathsf{UG}(A)$; however, not all transitions of the region automaton of $\mathsf{UG}(A)$ are to be added since location guards mean that some transitions are not enabled at a given region. Unless otherwise stated, by region states we mean $M^\nearrow$-region states. The steps of the construction are illustrated in Figure 4. From A, we first obtain $\mathsf{UG}(A)$, and build the region automaton for $\mathsf{UG}(A)$, denoted by $\mathcal{R}_{M^\nearrow}(\mathsf{UG}(A))$. Then Algorithm 1 builds the so-called *DTN region automaton* $\mathcal{D}(A)$ which is a finite automaton. Finally we construct $\mathcal{S}(A)$ which we refer to as the *summary timed automaton*, a timed automaton derived from $\mathcal{D}(A)$ by adding clocks and clock guards to $\mathcal{D}(A)$. Our main result is that $\mathcal{S}(A)$ recognizes the language $\mathcal{L}(A^\infty)\downarrow_1$.

Intuitively, Algorithm 1 computes region states reachable by a single process within the context of a network of arbitrary size. These region states are partitioned according to their slots. More precisely, Algorithm 1 computes (lines 3-4) the sequence $(W_i, E_i)_{i\geq 0}$, where W_i is a set of $M^\nearrow$-region states of $\mathsf{UG}(A)$ having the same slot (written $\mathsf{slot}(W_i)$), and E_i is a set of transitions from region states of W_i to either W_i or W_{i+1}. These transitions include ϵ-transitions which correspond to delay transitions: if the slot does not change during the delay transition, then the transition goes to a region-state which is also in W_i; otherwise, it leaves to the next slot and the successor is in W_{i+1}. During discrete transitions from W_i, the slot does not change, so the successor region-states are always inside W_i. In order to check if a discrete transition with location guard γ must be considered, the algorithm checks if some region-state (γ, r_γ) was previously added to the *same* layer W_i. This means that some (other) process can be at γ somewhere at a global time that belongs to $\mathsf{slot}(W_i)$. This is the nontrivial part of the algorithm: the proof will establish that if a process can be at location γ at *some* time in a given slot s, then it can also be at γ at *any* time within s.

For two sets W_i, W_j of region states of $\mathsf{UG}(A)$, let us define $W_i \approx W_j$ iff W_j can be obtained from W_i by shifting the slot, that is, if there exists $k \in \mathbb{Z}$ such that $(W_i)_{\mathsf{slot}+k} = W_j$. Recall that $(W_i)_{\mathsf{slot}+k} = W_j$ means that both sets contain the same regions when projected to the local clocks $\mathcal{C} \setminus \{t\}$. This definition is of course symmetric.

Algorithm 1 stops (line 5) when $W_{i_0} \approx W_{l_0}$ for some $i_0 < l_0$ with both layers having singleton slots (this requirement could be relaxed but this simplifies the proofs and only increases the number of iterations by a factor of 2).

The algorithm returns the DTN region automaton $\mathcal{D}(A) = (W, (\hat{q}, \hat{r}), \Sigma, E)$, where W is the set of explored region states, and E is the set of transitions that were added; except that transitions leaving W_{l_0-1} are redirected back to W_{i_0} (lines 7–9). Redirecting such transitions means that whenever $\mathcal{R}_{M^\nearrow}(\mathsf{UG}(A))$ has a delay transition from (q, r) to (q, r') with $\mathsf{slot}(r) = \mathsf{slot}(W_{l_0-1})$ and $\mathsf{slot}(r') = \mathsf{slot}(W_{l_0})$, then we actually add a transition from (q, r) to (q, r''), where r'' is obtained from r' by shifting the slot to that of $\mathsf{slot}(W_{i_0})$; this means that $r'\downarrow_{-t} = r''\downarrow_{-t}$, so these define the same clock valuations except with a shifted slot. The property $W_{i_0} \approx W_{l_0}$ ensures that $(q, r'') \in W_{i_0}$.

Algorithm 1 Algorithm to compute DTN region automaton of GTA A.

input : GTA $A = (Q, \hat{q}, \mathcal{C}, \Sigma, \mathcal{T}, Inv)$ and $\mathcal{R}_{M^{\nearrow}}(\mathsf{UG}(A))$
output : The DTN region automaton of A

Initialize $s \leftarrow [0,0]$, $W_0 \leftarrow \{(\hat{q}, \hat{r})\}$, $E_0 \leftarrow \emptyset$, $l \leftarrow -1$
repeat
  $l \leftarrow l + 1$;
  Compute (W_l, E_l) by applying the following rules until a fixed point is reached:
  - Rule 1: For any $(q,r) \in W_l$, let $(q,r) \xrightarrow{\mathsf{delay}} (q,r')$ s.t. $\mathsf{slot}(r') = s$, do
    $W_l \leftarrow W_l \cup \{(q,r')\}$, and $E_l \leftarrow E_l \cup \left\{\big((q,r), \epsilon, (q,r')\big)\right\}$.
  - Rule 2: For any $(q,r) \in W_l$ and $\tau = (q, g, \mathcal{C}_r, \sigma, \gamma, q')$ s.t. $(q,r) \xrightarrow{\sigma} (q',r')$, if $\gamma = \top$, or if there exists $(\gamma, r_\gamma) \in W_l$,
    then do $W_l \leftarrow W_l \cup \{(q',r')\}$, and $E_l \leftarrow E_l \cup \left\{\big((q,r), \sigma, (q',r')\big)\right\}$

  $s \leftarrow \mathsf{next}(s)$;
  $W_{l+1} \leftarrow \{(q,r') \mid (q,r) \in W_l, (q,r) \xrightarrow{\mathsf{delay}} (q,r') \text{ and } \mathsf{slot}(r') = s\}$;
  $E_l \leftarrow E_l \cup \left\{\big((q,r), \epsilon, (q,r')\big) \mid (q,r) \in W_l,\ (q,r) \xrightarrow{\mathsf{delay}} (q,r') \wedge \mathsf{slot}(r') = s\right\}$;
until $\exists i_0 < l : W_l \approx W_{i_0}$, *and* $\mathsf{slot}(W_{i_0})$ *is a singleton*;
$l_0 \leftarrow l$;
$E_{l_0-1} \leftarrow E_{l_0-1} \cap \big(W_{l_0-1} \times \Sigma \times W_{l_0-1}\big) \cup$
$\left\{\big((q,r), \epsilon, (q,r')\big) \,\middle|\, (q,r') \in W_{i_0}, \exists r'', \exists k \in \mathbb{N}, \big((q,r), \epsilon, (q,r'')\big) \in E_{l_0-1} \cap \big(W_{l_0-1} \times \Sigma \times W_{l_0}\big), r'_{\mathsf{slot}+k} = r''\right\}$
$W \leftarrow \cup_{0 \le i \le l_0-1} W_i$, $E \leftarrow \cup_{0 \le i \le l_0-1} E_i$
return $(W, (\hat{q}, \hat{r}), \Sigma, E)$

We write $(q,r) \xRightarrow{\sigma} (q',r')$ iff $\big((q,r), \sigma, (q',r')\big) \in E$. Paths and computations are defined for the DTN region automaton analogously to region automata.

We now show how to construct the summary timed automaton $\mathcal{S}(A)$ (the step from $\mathcal{D}(A)$ to $\mathcal{S}(A)$ in Figure 4). We define $\mathcal{S}(A)$ by extending $\mathcal{D}(A)$ with the clocks of A. Moreover, each transition $((q,r), \epsilon, (q,r'))$ has the guard $r'{\downarrow_{-t}}$ and no reset; and each transition $((q,r), \sigma, (q',r'))$ with $\sigma \neq \epsilon$ has the guard $r{\downarrow_{-t}}$, and resets the clocks that are equal to 0 in r'. The intuition is that $\mathcal{S}(A)$ ensures by construction that any valuation that is to take a discrete transition ($\sigma \neq \epsilon$) at location (q,r) belongs to r. Notice that we omit invariants here. Because transitions are derived from those of $\mathcal{R}_{M^{\nearrow}}(\mathsf{UG}(A))$, the only additional behavior we can have in $\mathcal{S}(A)$ due to the absence of invariants is a computation delaying in a location (q,r) and reaching outside of r (without taking an ϵ-transition), while no discrete transitions can be taken afterwards. Because traces end with a discrete transition, this does not add any trace not possible in $\mathcal{L}{\downarrow_1}(A^\infty)$.

3.2.1 Properties of Algorithm 1

We explain the overview of the correctness argument for Algorithm 1 and some of its consequences.

Let us first prove the termination of the algorithm, which also yields a bound on the number of iterations of the main loop (thus on l_0 and i_0). Recall that for a given A, N_A denotes the number of pairs (q,r) where $q \in Q$ and r is an M-region (see Section 3.1).

▶ **Lemma 16.** *Let $\mathcal{D}(A)$ be a DTN region automaton returned by Algorithm 1. Then the slots of all region states in $\mathcal{D}(A)$ are bounded. Consequently, the number of iterations of Algorithm 1 is bounded by 2^{N_A+1}.*

The region automaton is of exponential size. Each iteration of Algorithm 1 takes exponential time since one might have to go through all region states in the worst case. By Lemma 16, the number of iterations is bounded by 2^{N_A}, which is doubly exponential in $|\mathcal{C}|$. Theorem 21 will show how to decide the reachability PMCP in exponential space.

We now prove the correctness of the algorithm in the following sense.

▶ **Theorem 17.** *Let A be a GTA, $\mathcal{D}(A) = (W, (\hat{q}, \hat{r}), \Sigma, E)$ its DTN region automaton, $\mathcal{S}(A)$ be the summary timed automaton. Then we have $\mathcal{L}(A^\infty)\downarrow_1 = \mathcal{L}(\mathcal{S}(A))$.*

To prove this, we need the following lemma that states a nontrivial property on which we rely: if a process can reach a given location q at global time t', then it can also reach q at any global time within the slot of t'. It follows that the set of global times at which a location can be occupied by at least one process is a union of intervals. Intuitively, this is why partitioning the region states by slots is a good enough abstraction in our setting.

▶ **Lemma 18.** *Consider a GTA A with bound function $M^\nearrow$. Let $\rho_r = (q_0, r_0) \xrightarrow{\sigma_0} \dots \xrightarrow{\sigma_{l-1}} (q_l, r_l)$ such that $(q_0, r_0) = (\hat{q}, \hat{r})$ be a computation in $\mathcal{R}_{M^\nearrow}(\mathsf{UG}(A))$ such that $\mathsf{slot}(r_l)$ is bounded. For all $t' \in \mathsf{slot}(r_l)$, there exists a timed computation $(q_0, v_0) \to \dots \to (q_l, v_l)$ in $\mathsf{UG}(A)$ such that $v_i \in r_i$ for $0 \le i \le l$, and $v_l(t) = t'$.*

The following lemma proves one direction of Theorem 17.

▶ **Lemma 19.** *Consider a trace $tt = (\delta_0, \sigma_0) \dots (\delta_{l-1}, \sigma_{l-1}) \in \mathcal{L}(\mathcal{S}(A))$. Let I be the unique interval of the form $[k, k]$ or $(k, k+1)$ with $k \in \mathbb{N}$ that contains $\delta_0 + \dots + \delta_{l-1}$.*

1. *For all $t' \in I$, there exists $n \in \mathbb{N}$, and a computation π of A^n such that $\mathsf{trace}(\pi)\downarrow_1 = (\delta'_0, \sigma_0) \dots (\delta'_{l-1}, \sigma_{l-1})$ for some $\delta'_i \ge 0$, and $\delta(\pi\downarrow_1) = t'$.*
2. *$tt \in \mathcal{L}(A^\infty)\downarrow_1$.*

The following lemma establishes the inclusion in the other direction. Given a computation π in A^n, we build a timed computation in $\mathcal{S}(A)$ on the same trace. Because the total time delay of π can be larger than the bound 2^{N_A}, we need to carefully calculate the slot in which they will end when projected to $\mathcal{S}(A)$.

▶ **Lemma 20.** *For any computation π of A^n with $n \in \mathbb{N}$, $\mathsf{trace}(\pi)\downarrow_1 \in \mathcal{L}(\mathcal{S}(A))$.*

Deciding the Reachability PMCP. It follows from Algorithm 1 that the reachability case can be decided in exponential space. This basically consists of running the main loop of Algorithm 1 without storing the whole list of all W_i, but only the last one. The loop needs to be repeated up to 2^{N_A+1} times (or until the target label σ_0 is found).

▶ **Theorem 21.** *The reachability PMCP for DTNs is decidable in EXPSPACE.*

Local Properties Involving Several Processes. The algorithm described above can be extended to compute $\mathcal{L}(A^\infty)\downarrow_{[1,a]}$. We define the *product* of k timed automata A_i, written $\otimes_{1 \le i \le k} A_i$, as the standard product of timed automata (see e.g., [16]) applied to A_i after replacing each label σ appearing in A_i by (i, σ).

▶ **Lemma 22.** *Given GTA A, and interval $I = [1, a]$, let $\mathcal{S}(A)$ be the summary automaton computed as above. Then $\mathcal{L}(A^\infty)\downarrow_I = \mathcal{L}(\otimes_{1 \le i \le a} \mathcal{S}(A))$.*

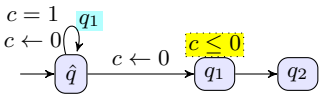

Figure 5 An example of GTA for which liveness is not preserved by our abstraction.

Limitations. Liveness properties (e.g., checking whether a transition can be taken an infinite number of times) are not preserved by our abstraction; since an infinite loop in the DTN region automaton may not correspond to a concrete computation in any A^n. In fact, consider the GTA in Figure 5. While there is an infinite loop on $\hat{q}$ in the DTN region automaton, no concrete execution takes the loop on $\hat{q}$ indefinitely, as each firing of this loop needs one more process to visit q_1, and then to leave it forever, due to the invariant $c \leq 0$.

4 Timed Lossy Broadcast Networks

Systems with lossy broadcast (a.k.a. "reconfigurable broadcast networks", where the underlying network topology might change arbitrarily at runtime) have received attention in the parameterized verification literature [22]. In the setting with finite-state processes, lossy broadcast is known to be at least as powerful as disjunctive guards, but it is unknown if it is strictly more powerful [15, Section 6]. We show that in our *timed* setting the two models are equally powerful, i.e., they simulate each other.

Lossy Broadcast Networks. Let Σ be a set of labels. A lossy broadcast timed automaton (LBTA) B is a tuple $(Q, \hat{q}, \mathcal{C}, \Sigma, \Lambda, \mathcal{T}, Inv)$ where $Q, \hat{q}, \mathcal{C}, Inv$ are as for TAs, and a transition is of the form $(q, g, \mathcal{C}_r, \sigma, \lambda, q')$, where $\lambda \in \{a!!, a?? \mid a \in \Lambda\}$. The *synchronization label* λ is used for defining global transitions. A transition with $\lambda = a!!$ is called a *sending transition*, and a transition with $\lambda = a??$ is called a *receiving transition*.

We also make Assumption 1 and Assumption 2 for LTBAs. The former means that the LBTA is timelock-free when all receiving transitions are removed. The semantics of a network of LBTAs is a timed transition system defined similarly as for NGTAs, except for *discrete transitions* which now induce a sequence of local transitions separated by 0 delays, as follows. Given $n > 0$, configurations of B^n are defined as for DTNs. Let $\mathfrak{c} = \big((q_1, v_1), \ldots, (q_n, v_n)\big)$ be a configuration of B^n. Consider indices $1 \leq i \leq n$ and $J \subseteq \{1, \ldots, n\} \setminus \{i\}$, and labels $\sigma, \sigma_j \in \Sigma$ for $j \in J$, such that

i) $(q_i, v_i) \xrightarrow{a!!,\sigma} (q_i', v_i')$ is a sending transition of B, and

ii) for all $j \in J$: $(q_j, v_j) \xrightarrow{a??,\sigma_j} (q_j', v_j')$ is a receiving transition of B.

Then the timed transition system of B^n contains the transition sequence $\mathfrak{c} \xrightarrow{(0,(i,\sigma))} \mathfrak{c}' \xrightarrow{(0,(j_1,\sigma_{j_1}))} \mathfrak{c}_1' \xrightarrow{(0,(j_2,\sigma_{j_2}))} \ldots \xrightarrow{(0,(j_m,\sigma_{j_m}))} \mathfrak{c}_m'$ for all possible sequences $j_1, \ldots, j_m$ where $J = \{j_1, \ldots, j_m\}$. Non-zero delays only occur outside of these chains of 0-delay transitions. For a given LBTA B, the family of systems B^∞ is called a *lossy broadcast timed network* (LBTN).

Simulating LBTN by DTN (and vice versa). The following theorem states that LBTAs and GTAs are inter-reducible.

▶ **Theorem 23.** *For all GTA A, there exists an LBTA B s.t. $\forall k{\geq}1$, $\mathcal{L}(A^\infty){\downarrow_{[1,k]}} \equiv \mathcal{L}(B^\infty){\downarrow_{[1,k]}}$. For all LBTA B, there exists a GTA A s.t. $\forall k{\geq}1, \mathcal{L}(A^\infty){\downarrow_{[1,k]}} = \mathcal{L}(B^\infty){\downarrow_{[1,k]}}$.*

Sketch. Simulation of disjunctive guards by lossy broadcast is simple: a transition from q to q' with location guard γ is simulated in lossy broadcast by the sender taking a self-loop transition on γ, and the receiver having a synchronizing transition from q to q'.

The other direction is where we need the power of clock invariants: to simulate a lossy broadcast where the sender moves from q to q' and a receiver moves from q_{rcv} to q'_{rcv}, in the DTN we first let the sender move to an auxiliary location q_σ (from which it can later move on to q'), and have a transition from q_{rcv} to q'_{rcv} that is guarded with q_σ. To ensure that no time passes between the steps of sender and receiver, we add an auxiliary clock c_{snd} that is reset when moving into q_σ, and q_σ has clock invariant $c_{\mathsf{snd}} = 0$.

In both directions, auxiliary transitions that are only needed for the simulation are labeled with fresh symbols in Σ^- such that they do not appear in the language of the system. ◀

Because the reduction to DTNs is in linear-time, we get the following.

▶ **Corollary 24.** *The reachability PMCP for LBTN is decidable in EXPSPACE.*

5 Synchronizing Timed Networks and Timed Petri Nets

We first introduce synchronizing timed networks. Our definitions follow [6, 5], except that their model considers systems with a controller process, whereas we assume (like in our previous models) that all processes execute the same automaton.

Synchronizing Timed Network. A *synchronizing timed automaton* (STA) $\mathcal{S}$ is a tuple $(Q, \hat{q}, \mathcal{C}, Inv, \mathcal{R})$ where Q, $\hat{q}$, $\mathcal{C}$, Inv are as for TAs, and $\mathcal{R}$ is a finite set of *rules*, where each rule $r \in \mathcal{R}$ is of the form $\langle q_{r,1} \xrightarrow{g_{r,1},\mathcal{C}_{r,1},\sigma_{r,1}} q'_{r,1}, \cdots, q_{r,m} \xrightarrow{g_{r,m},\mathcal{C}_{r,m},\sigma_{r,m}} q'_{r,m} \rangle$ for some $m \in \mathbb{N}$ and with $(q_i, g_{r,i}, \mathcal{C}_{r,i}, \sigma_{r,i}, q'_i) \in Q \times \Psi(\mathcal{C}) \times 2^{\mathcal{C}} \times \Sigma \times Q$ for $1 \leq i \leq m$.

The semantics of a *network of STAs* (NSTA) is defined as for NGTAs, except for *discrete transitions*, which now synchronize a subset of all processes in the following way: Let $r \in \mathcal{R}$ be a rule (of the form described above) and $\mathfrak{c} = \big((q_1, v_1), \ldots, (q_n, v_n)\big)$ a configuration of $\mathcal{S}^n$. Assume

i) there exists an injection $h : \{1, \ldots, m\} \to \{1 \ldots n\}$ such that for each $1 \leq i \leq m$, $q_{h(i)} = q_{r,i}$, $q_{h(i)} \xrightarrow{g_{r,i},\mathcal{C}_{r,i},\sigma_{r,i}} q'_{h(i)}$ is an element of r, $v_{h(i)} \models g_{r,i}$ and $v'_{h(i)} = v_{h(i)}[\mathcal{C}_{r,i} \leftarrow 0]$, and

ii) $j \notin \mathrm{range}(h)$, $q'_j = q_j$ and $v'_j = v_j$.

Then the timed transition system of $\mathcal{S}^n$ contains the transition sequence $\mathfrak{c} \xrightarrow{(0,(h(1),\sigma_{r,1}))} \mathfrak{c}_1 \xrightarrow{(0,(h(2),\sigma_{r,2}))} \ldots \xrightarrow{(0,(h(m),\sigma_{r,m}))} \mathfrak{c}_m$. That is, m distinct processes take individual transitions according to the rule without delay, and the configurations of the non-participating processes remain unchanged.

Again, we also make Assumptions 1 and 2 for STAs. The former means here that $\mathcal{S}$ is timelock-free when all transitions of rules with $m > 1$ are removed. All other notions follow in the natural way. Given an STA $\mathcal{S}$, the family of systems $\mathcal{S}^\infty$ is called a *synchronizing timed network* (STN).

▶ **Theorem 25.** *For all GTA A with set of locations Q, there exists an STA $\mathcal{S}$ with set of locations Q such that for every $q \in Q$: q is reachable in A iff q is reachable in $\mathcal{S}$. For all STA $\mathcal{S}$ with set of locations $Q_\mathcal{S}$, there exists a GTA A with set of locations $Q_A \supseteq Q_\mathcal{S}$ such that for every $q \in Q_\mathcal{S}$: q is reachable in A iff q is reachable in $\mathcal{S}$.*

Sketch. Simulation of disjunctive guards by STAs is simple: a transition from q to q' with location guard γ is simulated by a pairwise synchronization, where one process takes a self-loop on γ, and the other moves from q to q'.

Table 1 Existing and new decidability results for location reachability (Reach) and local trace properties (Trace) for DTN, LBTN, and STN with a single ($|\mathcal{C}| = 1$) or multiple clocks ($|\mathcal{C}| > 1$), and with (*Inv*) or without invariants (~~*Inv*~~). Entries with ✓* need to satisfy Assumption 1.

	DTN			LBTN			STN		
	$\|\mathcal{C}\|=1$	$\|\mathcal{C}\|>1$	$\|\mathcal{C}\|>1$	$\|\mathcal{C}\|=1$	$\|\mathcal{C}\|>1$	$\|\mathcal{C}\|>1$	$\|\mathcal{C}\|=1$	$\|\mathcal{C}\|>1$	$\|\mathcal{C}\|>1$
	~~*Inv*~~	~~*Inv*~~	*Inv*	~~*Inv*~~	~~*Inv*~~	*Inv*	~~*Inv*~~	~~*Inv*~~	*Inv*
Reach	✓[31]	✓[31]	✓	✓[6, 11]	✓	✓	✓[6]	✓	✓
Trace	✓[31]	✓[31]	✓*	✓	✓	✓*	?	?	?

Conversely, to simulate a rule r of the STA with m participating processes, we add auxiliary locations $p_{r,i}$, for $1 \leq i \leq m$, each with a clock invariant (on an additional clock only used for the simulation) that ensures that no time passes during simulation. For each element $q_{r,i} \xrightarrow{g_{r,i},\mathcal{C}_{r,i},\sigma_{r,i}} q'_{r,i}$ of r, we have a transition from $q_{r,i}$ to $p_{r,i}$, and from there to $q'_{r,i}$. A transition to $p_{r,i}$ is guarded with $p_{r,i-1}$ (except when $i = 1$), and with the clock constraint $g_{r,i}$, and all transitions to $q'_{r,i}$ are guarded with $p_{r,m}$. This ensures that any $q'_{r,i}$ is reachable through this construction if and only if the global configuration at the beginning would allow the STA to execute rule r. To avoid introducing timelocks, each of the $p_{r,i}$ has an additional transition with a trivial location guard and no clock guard to a new sink location $q_\perp$ that does not have an invariant. I.e., if simulation of a rule is started but cannot be completed (because there are processes in some but not all of the locations $q_{r,i}$), then processes can (and have to) move to $q_\perp$. ◀

▶ **Corollary 26.** *The reachability PMCP for STN is decidable in EXPSPACE.*

Note that the construction in our proof is in general not suitable for language equivalence, i.e., $\mathcal{L}(A^\infty)\downarrow_{[1,k]}$ might contain traces that are not in $\mathcal{L}(\mathcal{S}^\infty)\downarrow_{[1,k]}$.

Abdulla et al. [2] considered the *universal safety problem* of *timed Petri nets* – that is, whether a given transition can eventually be fired for any number of tokens in the initial place – and solved it for the case where each token has a single clock. The question whether the problem is decidable for tokens with multiple clocks remained open. This problem, in the multi-clock setting, can be reduced to the PCMP of STNs. The reduction is conceptually straightforward and computable in polynomial time in the size of the input.

▶ **Corollary 27.** *The universal safety problem for timed Petri nets with an arbitrary number of clocks is decidable in EXPSPACE.*

6 Conclusion

In this paper, we solved positively the parameterized model checking problem (PMCP) for finite local trace properties of disjunctive timed networks (DTNs) with invariants. We also proved that the PMCP for networks that communicate via lossy broadcast can be reduced to the PMCP for DTNs, and is therefore decidable. Additional results also allowed us to solve positively the open problem from [2] whether the universal safety problem for timed Petri nets with multiple clocks is decidable. Table 1 gives an overview of our results, compared to existing results for the classes of systems we consider.

In addition to the results presented here, we believe that our proof techniques can be extended to support timed networks with more powerful communication primitives, and in some cases to networks with controllers.

Future work will include tightening the complexity bounds for the problems considered here, as well as the development of zone-based algorithms that can be more efficient in practice than a direct implementation of the algorithms presented here.

References

1 Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Jonathan Cederberg. Timed lossy channel systems. In Deepak D'Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan, editors, *FSTTCS*, volume 18 of *LIPIcs*, pages 374–386. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPICS.FSTTCS.2012.374.

2 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Radu Ciobanu, Richard Mayr, and Patrick Totzke. Universal safety for timed Petri nets is PSPACE-complete. In Sven Schewe and Lijun Zhang, editors, *CONCUR*, volume 118 of *LIPIcs*, pages 6:1–6:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.CONCUR.2018.6.

3 Parosh Aziz Abdulla and Giorgio Delzanno. Parameterized verification. *International Journal on Software Tools for Technology Transfer*, 18(5):469–473, 2016. doi:10.1007/s10009-016-0424-3.

4 Parosh Aziz Abdulla, Giorgio Delzanno, Othmane Rezine, Arnaud Sangnier, and Riccardo Traverso. Parameterized verification of time-sensitive models of ad hoc network protocols. *Theoretical Computer Science*, 612:1–22, 2016. doi:10.1016/j.tcs.2015.07.048.

5 Parosh Aziz Abdulla, Johann Deneux, and Pritha Mahata. Multi-clock timed networks. In *LiCS*, pages 345–354. IEEE Computer Society, 2004. doi:10.1109/LICS.2004.1319629.

6 Parosh Aziz Abdulla and Bengt Jonsson. Model checking of systems with many identical timed processes. *Theoretical Computer Science*, 290(1):241–264, 2003. doi:10.1016/S0304-3975(01)00330-9.

7 Parosh Aziz Abdulla, A. Prasad Sistla, and Muralidhar Talupur. Model checking parameterized systems. In Edmund M Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 685–725. Springer, 2018. doi:10.1007/978-3-319-10575-8_21.

8 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, April 1994. doi:10.1016/0304-3975(94)90010-8.

9 Benjamin Aminof, Tomer Kotek, Sasha Rubin, Francesco Spegni, and Helmut Veith. Parameterized model checking of rendezvous systems. *Distributed Computing*, 31(3):187–222, 2018. doi:10.1007/s00446-017-0302-6.

10 Benjamin Aminof, Sasha Rubin, Florian Zuleger, and Francesco Spegni. Liveness of parameterized timed networks. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *ICALP, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 375–387. Springer, 2015. doi:10.1007/978-3-662-47666-6_30.

11 Étienne André, Benoît Delahaye, Paulin Fournier, and Didier Lime. Parametric timed broadcast protocols. In Constantin Enea and Ruzica Piskac, editors, *VMCAI*, volume 11388 of *Lecture Notes in Computer Science*, pages 491–512. Springer, 2019. doi:10.1007/978-3-030-11245-5_23.

12 Étienne André, Paul Eichler, Swen Jacobs, and Shyam Lal Karra. Parameterized verification of disjunctive timed networks. In Rayna Dimitrova and Ori Lahav, editors, *VMCAI*, volume 14499 of *Lecture Notes in Computer Science*, pages 124–146. Springer, 2024. doi:10.1007/978-3-031-50524-9_6.

13 E. André, S. Jacobs, S.L. Karra, and O. Sankur. Parameterized verification of timed networks with clock invariants (extended version), 2024. arXiv:2408.05190. URL: https://arxiv.org/abs/2408.05190.

14 Krzysztof R. Apt and Dexter Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 22(6):307–309, 1986. doi:10.1016/0020-0190(86)90071-2.

15 A. R. Balasubramanian and Chana Weil-Kennedy. Reconfigurable broadcast networks and asynchronous shared-memory systems are equivalent. In Pierre Ganty and Davide Bresolin, editors, *GandALF*, volume 346 of *EPTCS*, pages 18–34, 2021. doi:10.4204/EPTCS.346.2.

16 Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2003. doi:10.1007/978-3-540-27755-2_3.

17 Nathalie Bertrand and Paulin Fournier. Parameterized verification of many identical probabilistic timed processes. In Anil Seth and Nisheeth K. Vishnoi, editors, *FSTTCS*, volume 24 of *LIPIcs*, pages 501–513. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPIcs.FSTTCS.2013.501.

18 Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. *Decidability of Parameterized Verification*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015. doi:10.2200/S00658ED1V01Y201508DCT013.

19 Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Are timed automata updatable? In E. Allen Emerson and A. Prasad Sistla, editors, *CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 464–479. Springer, 2000. doi:10.1007/10722167_35.

20 Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2-3):291–345, August 2004. doi:10.1016/j.tcs.2004.04.003.

21 Patricia Bouyer, Uli Fahrenberg, Kim Guldstrand Larsen, Nicolas Markey, Joël Ouaknine, and James Worrell. Model checking real-time systems. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 1001–1046. Springer, 2018. doi:10.1007/978-3-319-10575-8_29.

22 Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Parameterized verification of ad hoc networks. In Paul Gastin and François Laroussinie, editors, *CONCUR*, volume 6269 of *Lecture Notes in Computer Science*, pages 313–327. Springer, 2010. doi:10.1007/978-3-642-15375-4_22.

23 E. Allen Emerson and Vineet Kahlon. Reducing model checking of the many to the few. In David A. McAllester, editor, *CADE*, volume 1831 of *Lecture Notes in Computer Science*, pages 236–254. Springer, 2000. doi:10.1007/10721959_19.

24 E. Allen Emerson and Kedar S. Namjoshi. On reasoning about rings. *International Journal of Foundations of Computer Science*, 14(4):527–550, 2003. doi:10.1142/S0129054103001881.

25 Javier Esparza, Alain Finkel, and Richard Mayr. On the verification of broadcast protocols. In *LiCS*, pages 352–359. IEEE Computer Society, 1999. doi:10.1109/LICS.1999.782630.

26 Javier Esparza, Stefan Jaax, Mikhail A. Raskin, and Chana Weil-Kennedy. The complexity of verifying population protocols. *Distributed Computing*, 34(2):133–177, 2021. doi:10.1007/s00446-021-00390-x.

27 Pierre Ganty and Rupak Majumdar. Algorithmic verification of asynchronous programs. *ACM Transactions on Programming Languages and Systems*, 34(1):6:1–6:48, 2012. doi:10.1145/2160910.2160915.

28 Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, 1992. doi:10.1145/146637.146681.

29 Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994. doi:10.1006/inco.1994.1045.

30 Frédéric Herbreteau, Dileep Kini, B. Srivathsan, and Igor Walukiewicz. Using non-convex approximations for efficient analysis of timed automata. In Supratik Chakraborty and Amit Kumar, editors, *FSTTCS*, volume 13 of *LIPIcs*, pages 78–89. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPIcs.FSTTCS.2011.78.

31 Luca Spalazzi and Francesco Spegni. Parameterized model checking of networks of timed automata with Boolean guards. *Theoretical Computer Science*, 813:248–269, 2020. doi:10.1016/j.tcs.2019.12.026.

32 Ichiro Suzuki. Proving properties of a ring of finite-state machines. *Information Processing Letters*, 28(4):213–214, 1988. doi:10.1016/0020-0190(88)90211-6.