Revisiting local time semantics for networks of timed automata

R Govind, Frédéric Herbreteau, B. Srivathsan, Igor Walukiewicz

Revisiting local time semantics for networks of timed automata

R. Govind
Chennai Mathematical Institute, India
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, F-33400, Talence, France
govindr@cmi.ac.in

Frédéric Herbreteau
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, F-33400, Talence, France
fh@labri.fr

B. Srivathsan
Chennai Mathematical Institute, India
sri@cmi.ac.in

Igor Walukiewicz
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, F-33400, Talence, France
igw@labri.fr

Abstract

We investigate a zone based approach for the reachability problem in timed automata. The challenge is to alleviate the size explosion of the search space when considering networks of timed automata working in parallel. In the timed setting this explosion is particularly visible as even different interleavings of local actions of processes may lead to different zones. Salah *et al.* in 2006 have shown that the union of all these different zones is also a zone. This observation was used in an algorithm which from time to time detects and aggregates these zones into a single zone.

We show that such aggregated zones can be calculated more efficiently using the local time semantics and the related notion of local zones proposed by Bengtsson *et al.* in 1998. Next, we point out a flaw in the existing method to ensure termination of the local zone graph computation. We fix this with a new algorithm that builds the local zone graph and uses abstraction techniques over (standard) zones for termination. We evaluate our algorithm on standard examples. On various examples, we observe an order of magnitude decrease in the search space. On the other examples, the algorithm performs like the standard zone algorithm.

2012 ACM Subject Classification Theory of computation → Verification by model checking

Keywords and phrases Timed automata, verification, local-time semantics, abstraction

Funding Work supported by UMI 2000 ReLaX and project IoTTTA - CEFIPRA Indo-French program in ICST - DST/CNRS ref. 2016-01. Author B. Srivathsan is partially funded by grants from Infosys Foundation, India and Tata Consultancy Services, India

1 Introduction

Timed automata [1] are a popular model for real-time systems. They extend finite state automata with real valued variables called *clocks*. Constraints on clock values can be used as guards for transitions, and clocks can be reset to zero during transitions. Often, it is more natural to use a network of timed automata which operate concurrently and synchronize on joint actions. We study the reachability problem for networks of timed automata: given a state of the timed automaton network, is there a run from the initial state to the given state.

A widely used technique to solve the reachability problem constructs a *zone graph* [7] whose nodes are *(state, zone)* pairs consisting of a state of the automaton and a *zone* representing a set of clock valuations [8]. This graph may not be finite, so in order to

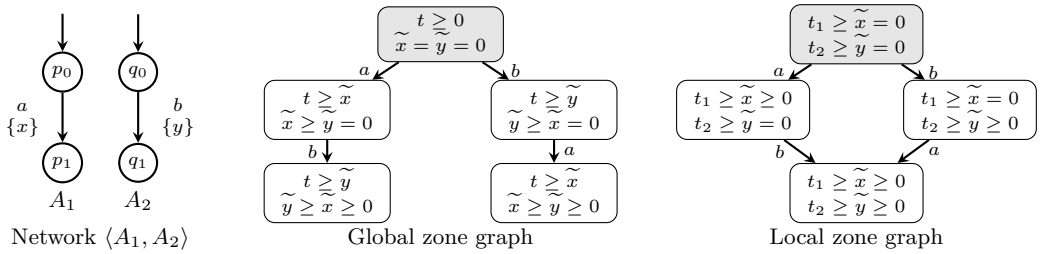

Figure 1 Illustration of commutativity in the local zone graph

guarantee termination of an exploration algorithm, various sound and complete abstraction techniques are used [7, 2, 11, 9].

Dealing with automata operating in parallel poses the usual state-space explosion problem arising due to different interleavings. Consider an example of a network with two processes in Figure 1. Actions a and b are local to each process. Variables x, y are clocks and $\{x\}$ denotes that clock x is reset in transition a. The (global) zone graph maintains the set of configurations reached after each sequence of actions. Although a and b are local actions, there is an intrinsic dependence between the two processes happening due to time. Hence, the zone reached after executing sequence ab contains configurations where x is reset before y and the zone after ba contains configurations where y is reset before x. So the sequences ab and ba lead to different zones. The number of different interleavings of sequences of local actions increases exponentially when their length grows, or when more processes get involved. Every interleaving can potentially lead to a different zone.

Salah et al. [16] have shown a surprising property that the *union* of all zones reached by the interleavings of a sequence of actions is also a zone. We call it an *aggregated zone*. Their argument is based on the fact that for a given sequence one can write a zone-like constraint defining all the runs on the interleavings of the sequence. Then the aggregated zone is obtained simply by projecting this big constraint on relevant components. This approach requires to work with sets of constraints whose size grows with the length of a sequence. This is both inefficient and limited to finite sequences. They use this observation in an algorithm where, when all the interleavings of a sequence σ have been explored, the resulting zones are aggregated to a single zone and further exploration is restricted to this aggregated zone. This requires detecting from time to time whether aggregation can happen. This is an obstacle in using aggregated zones in efficient reachability-checking algorithms. Another limitation of this approach is that it works only for acyclic automata.

Another approach by Bengtsson et al. [4] involves making time local to each process. This local time approach is based on a very elegant idea: make time in every process progress independently, and synchronize local times of processes when they need to perform a common action. In consequence, the semantics has the desired property: two actions whose process domains are disjoint are commutative. In the example above, depending on the local time in processes A_1 and A_2, the sequence ab may result, on a global time scale, in a occurring before b, as well as b occurring before a. As a result, the set of valuations reached after ab does not remember the order in which a and b occurred. Thus, sequences ab and ba lead to the same set of configurations: those obtained after doing a and b concurrently. Similar to standard zones, we now have local zones and a local zone graph having *(state, local zone)* pairs. Due to the above argument, the local zone graph has a nice property — if a run σ from an initial zone reaches a local zone, then all the runs equivalent to σ lead to the same local zone. Local zone graphs are therefore ideal for handling interleavings. However,

substantially more involved abstraction techniques are needed for local zones to make the local zone graph finite. A *subsumption relation* between local zones was defined in [4] but no algorithm was proposed and there was no effective way to use local time semantics. Later Minea [14] proposed a widening operator on local zones to construct finite local zone graphs.

Summary of results in the paper:

- We show that the aggregated zone of interleavings of σ in the standard zone graph is obtained by synchronizing valuations in the local zone obtained after σ (Theorem 24). Hence computing the local zone graph gives a more direct and efficient algorithm than Salah et al. to compute aggregated zones. Moreover, this algorithm is not restricted to acyclic timed automata.
- We point out a flaw in the abstraction procedure of Minea to get a finite local zone graph (Section 5).
- We propose a different algorithm to get a finite local zone graph. This gives a new reachability algorithm for networks of timed automata, which works with local zones but uses subsumption on standard zones (Definition 25 and Theorem 26). Instead of subsumptions between local zones, we use subsumptions between the synchronized valuations inside these local zones. This helps us to exploit the (well-studied) subsumptions over standard zones [7, 2, 11, 9]. Moreover, this subsumption is much more aggressive than the standard one since, thanks to local-time semantics, the (aggregated) zone used in the subsumption represents all the valuations reachable not only by the execution that we are exploring but also by all the executions equivalent to it.
- We report on experiments performed with a prototype implementation of the algorithm (Table 1). The algorithm performs surprisingly well on some examples, and it is never worse than the standard zone graph algorithm.

Related work: The basis of this work is local-time semantics and local zones developed by Bengtsson et. al. [4]. The authors have left open how to use this semantics to effectively compute the local zone graph since no efficient procedure was provided to ensure its finiteness. To that purpose, Minea [15, 14] has proposed a subsumption operation on local zones, and an algorithm using this operation. Unfortunately, as we exhibit here, the algorithm has a flaw that is not evident to repair. Lugiez et. al. [13] also use local time but their method is different. They use constraints to check if local clocks of an execution can be synchronized sufficiently to obtain a standard run.

Aggregated zones are crucial to obtain an efficient verification procedure for networks of timed automata. Coming to the same state with different zones inflicts a huge blowup in the zone graph, since the same paths are explored independently from each of these zones. This has also been observed in the context of multi-threaded program verification in [17]. Solving this problem in the context of program analysis requires to over approximate the aggregated state. Fortunately, in the context of timed automata, the result of aggregating these zones in still a zone [16]. Efficient computation of aggregated zones is thus an important advance in timed automata verification as demonstrated by our experimental results in Section 6.

2 Networks of timed automata

We start by defining networks of timed automata and two semantics for them: a global-time semantics (the usual one) and a local-time semantics (introduced in [4]). Then, we recall the

fact that they are equivalent, and state some interesting properties of local-time semantics w.r.t. concurrency that were observed in [4].

Let $\mathbb{N}$ denote the set of natural numbers and $\mathbb{R}_{\geq 0}$ the set of non-negative reals. Let X be a finite set of variables called *clocks*. Let $\phi(X)$ denote a set of clock constraints generated by the following grammar: $\phi := x \sim c \mid \phi \wedge \phi$ where $x \in X$, $c \in \mathbb{N}$, and $\sim \in \{<, \leq, =, \geq, >\}$.

▶ **Definition 1** (Network of timed automata). *A network of timed automata with k processes, is a k-tuple of timed automata $A_1, \ldots, A_k$. Each process $A_p = \langle Q_p, \Sigma_p, X_p, q_p^{init}, T_p \rangle$ has a finite set of states Q_p, a finite alphabet of actions Σ_p, a finite set of clocks X_p, an initial state q_p^{init}, and transitions $T_p \subseteq \Sigma_p \times Q_p \times \phi(X_p) \times 2^{X_p} \times Q_p$. We require that the sets of states, and the sets of clocks are pairwise disjoint: $Q_{p_1} \cap Q_{p_2} = \emptyset$, and $X_{p_1} \cap X_{p_2} = \emptyset$ for $p_1 \neq p_2$. We write Proc for the set of all processes.*

A network is a parallel composition of timed automata. Its semantics is that of the timed automaton obtained as the "synchronized product" of the processes. For an action b, a b-transition of a process p is an element of T_p with b in the first component. Synchronization happens on two levels: (i) via time that advances the same way in all the processes, and (ii) via common actions, for example if $b \in \Sigma_1 \cap \Sigma_2$, then processes 1 and 2 need to synchronize by doing a b-transition. We define the domain of an action b: $dom(b) = \{p : b \in \Sigma_p\}$ as the set of processes that must synchronize to do b. We will use some abbreviations: $Q = \Pi_{p=1}^{k} Q_p$, $\Sigma = \bigcup_{p=1}^{k} \Sigma_p$ and $X = \bigcup_{p=1}^{k} X_p$.

The semantics of a network is governed by the value of clocks at each instant. We choose to represent these values using offsets as this allows a uniform presentation of the global-time semantics below and the local-time semantics in Section 2.1. For every clock x, we introduce an offset variable $\widetilde{x}$. The value of $\widetilde{x}$ is the time-stamp at which x was last reset. In addition, we consider a variable t which tracks the global time: essentially t is a clock that is never reset. We now make this notion precise. Let $\widetilde{X}_p = \{\widetilde{x} \mid x \in X_p\}$ and $\widetilde{X} = \bigcup_{p=1}^{k} \widetilde{X}_p$. A *global valuation* v is a function $v : \widetilde{X} \cup \{t\} \mapsto \mathbb{R}_{\geq 0}$ such that $v(t) \geq v(\widetilde{x})$ for all variables $\widetilde{x}$. In this representation, the value of clock x corresponds to $v(t) - v(\widetilde{x})$, denoted $\overline{v}(x)$[1].

Recall that offset variable $\widetilde{x}$ stores the last time-stamp at which clock x has been reset. Hence, a delay in offset representation increases the value of reference clock t and leaves offset variables $\widetilde{x}$ unchanged. Formally: for $\delta \in \mathbb{R}_{\geq 0}$, we denote by $v + \delta$ the valuation defined by: $(v + \delta)(t) = v(t) + \delta$ and $(v + \delta)(\widetilde{x}) = v(\widetilde{x})$ for all $\widetilde{x}$. Similarly, resetting the clocks in $R \subseteq X$, yields a global valuation $[R]v$ defined by: $([R]v)(t) = v(t)$, and $([R]v)(\widetilde{x})$ is $v(t)$ if $x \in R$, and $v(\widetilde{x})$ otherwise. Given a global valuation v and a clock constraint g, we write $v \models g$ if every constraint in g holds after replacing x with its value $\overline{v}(x) = v(t) - v(\widetilde{x})$.

A configuration of the network is a pair (q, v) where $q \in Q$ is a global state, and v is a global valuation. We will write $q(p)$ to refer to the p-th component of the state q.

▶ **Definition 2** (Global-time semantics). *The semantics of a network $\mathcal{N}$ is given by a transition system whose states are configurations (q, v). The initial configuration is (q^{init}, v^{init}) where $q^{init}(p) = q_p^{init}$ is the tuple of initial states, and $v^{init}(y) = 0$ for $y \in \widetilde{X} \cup \{t\}$.*

There are two kinds of transitions, which we call steps: global delay, and action steps. A global delay *by the amount $\delta \in \mathbb{R}_{\geq 0}$ gives a step $(q, v) \xRightarrow{\delta}_{st} (q, v + \delta)$. An* action step *on action b gives $(q, v) \xRightarrow{b}_{st} (q', v')$ if there is a set of b-transitions $\{(q_p, g_p, R_p, q'_p)\}_{p \in dom(b)}$ of the respective processes such that:*

[1] Usually, semantics of timed automata is described using valuations of the form $\overline{v}$. Here, we have chosen v which uses offsets since it extends naturally to the local-time setting. Translations between these two kinds of valuations is straightforward, as shown.

- *processes from* $dom(b)$ *change states:* $q_p = q(p)$, $q'_p = q'(p)$, *for* $p \in dom(b)$, *and* $q(p) = q'(p)$ *for* $p \notin dom(b)$;
- *all the guards are satisfied:* $v \vDash g_p$, *for* $p \in dom(b)$;
- *all resets are performed:* $v' = [\bigcup_{p \in dom(b)} R_p]v$;

A *run of an automaton* from a configuration (q_0, v_0) is a sequence of steps starting in (q_0, v_0). For a sequence $u = b_1 \dots b_n$ of actions, we write $(q_0, v_0) \overset{u}{\Longrightarrow} (q_n, v'_n)$ if there is a run

$$(q_0, v_0) \overset{\delta_0}{\Longrightarrow}_{st} (q_0, v'_0) \overset{b_1}{\Longrightarrow}_{st} (q_1, v_1) \overset{\delta_1}{\Longrightarrow}_{st} (q_1, v'_1) \dots \overset{b_n}{\Longrightarrow}_{st} (q_n, v_n) \overset{\delta_n}{\Longrightarrow}_{st} (q_n, v'_n)$$

for some delays $\delta_0, \dots, \delta_n \in \mathbb{R}_{\geq 0}$.

▶ **Definition 3** (Reachability problem). *The* reachability problem *is to decide, given a network* $\mathcal{N}$ *and a state* q, *whether there is a run reaching* q; *or in other words, whether there exists a sequence of transitions* u *such that* $(q^{init}, v^{init}) \overset{u}{\Longrightarrow} (q, v)$ *for some valuation* v.

The reachability problem for networks of timed automata is PSPACE-complete [1].

2.1 Local-time semantics

The definition of a network of automata suggests an independence relation between actions: a pair of actions with disjoint domains (i.e. involving distinct processes) should commute. We say that two sequences of actions are *equivalent*, written $u \sim w$ if one can be obtained from the other by repeatedly permuting adjacent actions with disjoint domains.

▶ **Lemma 4.** *For two equivalent sequences* $u \sim w$: *if there are two runs* $(q, v) \overset{u}{\Longrightarrow} (q_u, v_u)$, *and* $(q, v) \overset{w}{\Longrightarrow} (q_w, v_w)$ *then* $q_u = q_w$.

Proof. Consider the basic case when there are two actions a and b with disjoint domains and $u = ab$, $w = ba$. Since a and b are on disjoint processes, executing ab or ba (whenever possible) leads to the same (discrete) state by definition.

Consider a general u. Sequence w is obtained by repeatedly permuting adjacent actions. From the basic case of the lemma, each permutation preserves the source and target (discrete) states, if it is feasible. ◀

Observe that in the above lemma we cannot assert that $v_u = v_w$. Even further, the existence of $(q, v) \overset{u}{\Longrightarrow} (q_u, v_u)$ does not imply that a run from (q, v) on w is feasible. This happens due to global time delays, i.e., delays that involve all the processes. For example, consider actions a and b on disjoint processes with a having guard $x \leq 1$ and b having guard $y \geq 2$. Then from the initial valuation one can execute ab but not ba.

One solution to get commutativity between actions with disjoint domains is to consider local-time semantics [4]. In this semantics, time elapses independently in every process, and time elapse is synchronized before executing a synchronized action. This way, two actions with disjoint domains become commutative. In the example from the previous paragraph, while the process executing b elapses 2 time units, the other process is allowed to not elapse time at all and hence ba becomes possible. Moreover, for the reachability problem, local-time semantics is equivalent to the standard one.

In the local-time semantics, we replace the clock t which was tracking the global time, with individual *reference clocks* t_p for each process A_p which track the local time of each process. We set $\widetilde{X}'_p = \widetilde{X}_p \cup \{t_p\}$ and $\widetilde{X}' = \bigcup_p \widetilde{X}'_p$. A *local valuation* v is a valuation over the set of clocks $\widetilde{X}'$ such that $\mathsf{v}(t_p) \geq \mathsf{v}(\widetilde{x})$ for all processes $p \in Proc$ and all clocks $\widetilde{x} \in \widetilde{X}_p$. This restriction captures the intuition that t_p is a reference clock for process p, and it is never

reset. In this setting, the value $\mathsf{v}(t_p) - \mathsf{v}(\widetilde{x})$ of clock x is defined relative to the reference clock t_p of process p that owns x, i.e. $x \in X_p$. We will use the notation v for local valuations to distinguish from global valuations v.

We introduce a new operation of local time elapse. For a process $p \in \mathit{Proc}$ and $\delta \in \mathbb{R}_{\geq 0}$, operation $\mathsf{v} +_p \delta$ adds δ to $\mathsf{v}(t_p)$, the value of the reference clock t_p of process p, and leaves the other variables unchanged. Formally, $(\mathsf{v} +_p \delta)(t_p) = \mathsf{v}(t_p) + \delta$ and $(\mathsf{v} +_p \delta)(y) = \mathsf{v}(y)$ for all $y \in \widetilde{X}' \setminus \{t_p\}$. A local valuation v satisfies a clock constraint g, denoted $\mathsf{v} \models g$ if every constraint in g holds after replacing x by its value $\mathsf{v}(t_p) - \mathsf{v}(\widetilde{x})$ where p is the process such that $x \in X_p$. We denote by $[R]\mathsf{v}$ the valuation obtained after resetting the clocks in $R \subseteq X$ and defined by: $([R]\mathsf{v})(t_p) = \mathsf{v}(t_p)$ for every reference clock t_p, $([R]\mathsf{v})(\widetilde{x}) = \mathsf{v}(\widetilde{x})$ if $x \notin R$, and $([R]\mathsf{v})(\widetilde{x}) = \mathsf{v}(t_p)$ if $x \in R$ and p is the process such that $x \in X_p$.

▶ **Definition 5** (Local steps of a timed automata network). *There are two kinds of local steps in a network $\mathcal{N}$:* local delay*, and* local action*. A local delay $\delta \in \mathbb{R}_{\geq 0}$ in process $p \in \mathit{Proc}$ is a step $(q, \mathsf{v}) \xrightarrow{p,\delta}_{st} (q, \mathsf{v} +_p \delta)$. For an action b, we have a step $(q, \mathsf{v}) \xrightarrow{b}_{st} (q', \mathsf{v}')$ if there is a set of b-transitions of respective processes $\{(q_p, g_p, R_p, q'_p)\}_{p \in dom(b)}$ such that:*

- *$q_p = q(p)$, $q'_p = q'(p)$, for $p \in dom(b)$, and $q(p) = q'(p)$ for $p \notin dom(b)$;*
- *start times are synchronized: $\mathsf{v}(t_{p_1}) = \mathsf{v}(t_{p_2})$, for every $p_1, p_2 \in dom(b)$;*
- *guards are satisfied: $\mathsf{v} \vDash g_p$, for every $p \in dom(b)$;*
- *resets are performed: $\mathsf{v}' = [\bigcup_{p \in dom(b)} R_p]\mathsf{v}$;*

The main difference with respect to global semantics is the presence of local time delay. As a result, every process can be in a different local time as emphasized by the reference clocks in each process. In consequence, in local action steps we require that when processes do a common action their local times should be the same. Of course a standard delay δ on all processes can be simulated by a sequence of delays on every process separately, as $\xrightarrow{1,\delta}_{st} \cdots \xrightarrow{k,\delta}_{st}$. For a sequence of local delays $\Delta = (p_1, \delta_1) \dots (p_n, \delta_n)$ we will write $(q, \mathsf{v}) \xrightarrow{\Delta}_{st} (q, \mathsf{v}')$ to mean $(q, \mathsf{v}) \xrightarrow{p_1,\delta_1}_{st} (q, \mathsf{v}_1) \xrightarrow{p_2,\delta_2}_{st} \cdots \xrightarrow{(p_n,\delta_n)}_{st} (q, \mathsf{v}')$.

▶ **Definition 6** (Local run). *A* local run *from a configuration (q_0, v_0) is a sequence of local steps. For a sequence of actions $u = b_1 \dots b_n$, we write $(q_0, \mathsf{v}_0) \xrightarrow{u} (q_n, \mathsf{v}'_n)$ if for some sequences of local delays $\Delta_0, \dots, \Delta_n$ there is a local run*

$$(q_0, \mathsf{v}_0) \xrightarrow{\Delta_0}_{st} (q_0, \mathsf{v}'_0) \xrightarrow{b_1}_{st} (q_1, \mathsf{v}_1) \xrightarrow{\Delta_1}_{st} \cdots \xrightarrow{b_n}_{st} (q_n, \mathsf{v}_n) \xrightarrow{\Delta_n}_{st} (q_n, \mathsf{v}'_n)$$

Observe that a run may start and end with a sequence of delays. In the next section we will make a link between local and global runs. For this we will first examine independence properties of local runs which are much better than for global runs (cf. Lemma 4).

▶ **Lemma 7** (Independence). *Suppose $dom(a) \cap dom(b) = \emptyset$. If $(q, \mathsf{v}) \xrightarrow{ab} (q', \mathsf{v}')$ then $(q, \mathsf{v}) \xrightarrow{ba} (q', \mathsf{v}')$. If $(q, \mathsf{v}) \xrightarrow{a} (q_a, \mathsf{v}_a)$ and $(q, \mathsf{v}) \xrightarrow{b} (q_b, \mathsf{v}_b)$ then $(q, \mathsf{v}) \xrightarrow{ab} (q_{ab}, \mathsf{v}_{ab})$ for some q_{ab} and v_{ab}.*

Proof. Take a run $(q, \mathsf{v}) \xrightarrow{\Delta_a} (q, \mathsf{v}_a) \xrightarrow{a} (q_a, \mathsf{v}'_a) \xrightarrow{\Delta_b} (q_a, \mathsf{v}_b) \xrightarrow{b} (q', \mathsf{v}'_b) \xrightarrow{\Delta} (q', \mathsf{v}')$. Let Δ'_b be a sequence of delays from Δ_a or Δ_b involving a process $p \in dom(b)$, i.e., pairs (p, δ_p) from Δ_a or Δ_b such that $p \in dom(b)$. Let Δ'_a be a sequence of delays in Δ_a involving processes in $dom(a)$; and finally let Δ' be the delays in $\Delta_a \cup \Delta_b \cup \Delta$ which were not counted in Δ'_a or Δ'_b. Since $dom(a) \cap dom(b) = \emptyset$ we get that the following sequence is a run: $(q, \mathsf{v}) \xrightarrow{\Delta'_b} (q, \mathsf{v}''_b) \xrightarrow{b} (q_b, \mathsf{v}'''_b) \xrightarrow{\Delta'_a} (q', \mathsf{v}''_a) \xrightarrow{a} (q', \mathsf{v}'''_a) \xrightarrow{\Delta'} (q', \mathsf{v}')$.

For the second part, suppose $(q, \mathsf{v}) \xrightarrow{\Delta_a} \xrightarrow{a} \xrightarrow{\Delta'_a} (q_a, \mathsf{v}_a)$ and $(q, \mathsf{v}) \xrightarrow{\Delta_b} \xrightarrow{b} \xrightarrow{\Delta'_b} (q_b, \mathsf{v}_b)$. Let Δ_1 be the delays involving processes in $dom(a)$ in Δ_a, and Δ_2 the delays of processes in $dom(b)$ in Δ_b. As $dom(a) \cap dom(b) = \emptyset$, $(q, \mathsf{v}) \xrightarrow{\Delta_1} \xrightarrow{a} \xrightarrow{\Delta_2} \xrightarrow{b} (q_{ab}, \mathsf{v}_{ab})$ is a local run. ◀

Recall that two sequences of actions are equivalent, written $u \sim w$ if one can be obtained from the other by repeatedly permuting adjacent actions with disjoint domains. Directly from the previous lemma we obtain.

▶ **Lemma 8.** *If* $(q_0, \mathsf{v}_0) \xrightarrow{u} (q_n, \mathsf{v}_n)$ *and* $u \sim w$ *then* $(q_0, \mathsf{v}_0) \xrightarrow{w} (q_n, \mathsf{v}_n)$.

With local-time semantics two equivalent sequences not only reach the same state q_n, but also the same local valuation v_n (in contrast with Lemma 4 for global-time semantics).

2.2 Comparing local and global runs

We have presented two semantics for networks of timed automata: global-time and local-time. Local runs have more freedom as time can elapse independently in every process. Yet, with respect to state reachability the two concepts turn out to be equivalent.

▶ **Definition 9.** *A local valuation* v *is* synchronized *if for every pair of processes* p_1, p_2*, the values of their reference clocks are equal:* $\mathsf{v}(t_{p_1}) = \mathsf{v}(t_{p_2})$.

For a synchronized local valuation v, let $\mathsf{global}(\mathsf{v})$ be the global valuation v such that $v(\widetilde{x}) = \mathsf{v}(\widetilde{x})$ and $v(t) = \mathsf{v}(t_1) = \cdots = \mathsf{v}(t_k)$. Conversely, to every global valuation v, we associate the synchronized local valuation $\mathsf{local}(v) = \mathsf{v}$ where $\mathsf{v}(\widetilde{x}) = v(\widetilde{x})$ and $\mathsf{v}(t_p) = v(t)$ for every reference clock t_p.

Before we prove the main observation of this section (Lemma 13), we develop some intermediate observations.

Every action step in a local run:

$$(q_0, \mathsf{v}_0) \xrightarrow{\Delta_0}_{st} (q_0, \mathsf{v}'_0) \xrightarrow{b_1}_{st} (q_1, \mathsf{v}_1) \dots \xrightarrow{\Delta_{n-1}}_{st} (q_{n-1}, \mathsf{v}'_{n-1}) \xrightarrow{b_n}_{st} (q_n, \mathsf{v}_n) \xrightarrow{\Delta_n}_{st} (q_n, \mathsf{v}'_n)$$

has its execution time; namely the step $(q_{i-1}, \mathsf{v}'_{i-1}) \xrightarrow{b_i}_{st} (q_i, \mathsf{v}_i)$ has the execution time $\mathsf{v}'_{i-1}(t_p) = \mathsf{v}_i(t_p)$ for $p \in dom(b_i)$. Observe that by definition of a step, the choice of p does not matter.

We will say that a local run is *soon* if for every i, the execution time of b_i is not bigger than the execution time of b_{i+1}.

▶ **Lemma 10.** *If* $(q_0, \mathsf{v}_0) \xrightarrow{u} (q_n, \mathsf{v}_n)$ *is a local run then there is* $w \sim u$ *such that* $(q_0, \mathsf{v}_0) \xrightarrow{w} (q_n, \mathsf{v}_n)$ *is a soon local run.*

Proof. Consider a sequence $u = b_1 \dots b_n$ and a run $(q_0, \mathsf{v}_0) \xrightarrow{\Delta_0}_{st} (q_0, \mathsf{v}'_0) \xrightarrow{b_1}_{st} (q_1, \mathsf{v}_1) \cdots \xrightarrow{\Delta_{n-1}}_{st} (q_{n-1}, \mathsf{v}'_{n-1}) \xrightarrow{b_n}_{st} (q_n, \mathsf{v}_n) \xrightarrow{\Delta_n}_{st} (q_n, \mathsf{v}'_n)$. Suppose that the order of execution times of b_i and b_{i+1} is reversed. Then $dom(b_i) \cap dom(b_{i+1}) = \emptyset$ by the definition of a run. So we can take $u' = b_1 \dots b_{i+1} b_i \dots b_n$ where the order of b_i and b_{i+1} is reversed. Since $u \sim u'$, by Lemma 8 we have a run $(q_0, \mathsf{v}_0) \xrightarrow{u'} (q_n, \mathsf{v}_n)$. We can, if necessary, repeat this operation from u' till we get the desired w. ◀

▶ **Lemma 11.** *If* $(q, \mathsf{v}) \xrightarrow{u} (q', \mathsf{v}')$ *is a local run where* v *and* v' *are synchronized valuations, then there is* $w \sim u$ *and a global run* $(q, \mathsf{global}(\mathsf{v})) \xRightarrow{w} (q', \mathsf{global}(\mathsf{v}'))$.

Proof. Let $(q, \mathsf{v}) = (q_0, \mathsf{v}_0)$ and $(q', \mathsf{v}') = (q_n, \mathsf{v}'_n)$. We take a local run, and assume that it is soon thanks to Lemma 10:

$$(q_0, \mathsf{v}_0) \xrightarrow{\Delta_0}_{st} (q_0, \mathsf{v}'_0) \xrightarrow{b_1}_{st} (q_1, \mathsf{v}_1) \xrightarrow{\Delta_1}_{st} \cdots$$
$$\cdots (q_{n-1}, \mathsf{v}_{n-1}) \xrightarrow{\Delta_{n-1}}_{st} (q_{n-1}, \mathsf{v}'_{n-1}) \xrightarrow{b_n}_{st} (q_n, \mathsf{v}_n) \xrightarrow{\Delta_n}_{st} (q_n, \mathsf{v}'_n) .$$

Let θ_i be the execution time of action b_i. For convenience, we set $\theta_0 = \mathsf{v}_0(t_p)$ and $\theta_{n+1} = \mathsf{v}'_n(t_p)$, for some process p. Since v_0 and v'_n is synchronized, the choice of p is irrelevant. We claim that there is a global run

$$(q_0, v_0) \overset{\delta_1}{\Longrightarrow}_{st} \overset{b_1}{\Longrightarrow}_{st} (q_1, v_1) \overset{\delta_2}{\Longrightarrow}_{st} \overset{b_2}{\Longrightarrow}_{st} \cdots \overset{\delta_i}{\Longrightarrow}_{st} \overset{b_i}{\Longrightarrow}_{st} (q_i, v_i)$$

with

- $v_0 = \mathsf{global}(\mathsf{v}_0)$
- $\delta_i = \theta_i - \theta_{i-1}$ for $i = 1, \ldots, n$
- $v_i(t) = \mathsf{v}_i(t_p)$ for some $p \in dom(b_i)$ and
- $v_i(\widetilde{x}) = \mathsf{v}_i(\widetilde{x})$ for all other clocks x

This statement is proved by induction on i. For $i = n$, this statement gives a global valuation v_n such that $v_n(t) = \mathsf{v}_n(t_p)$ where $p \in dom(b_n)$ and for all other clocks $v_n(\widetilde{x}) = \mathsf{v}_n(\widetilde{x})$. Note that valuations v_n and v'_n differ only in the values of the reference clocks. Moreover, in v'_n, all reference clocks are at θ_{n+1}. A global delay of $\delta_{n+1} = \theta_{n+1} - \theta_n$ from (q_n, v_n) gives (q_n, v'_n) such that $v'_n = \mathsf{global}(\mathsf{v}'_n)$. ◀

▶ **Lemma 12.** *If $(q, v) \overset{u}{\Longrightarrow} (q', v')$ is a global run, then there is a local run $(q, \mathsf{local}(v)) \xrightarrow{u} (q', \mathsf{local}(v'))$.*

Proof. A global run can be directly converted to a local run by changing a global delay to a sequence of local delays. ◀

▶ **Lemma 13.** *If $(q, \mathsf{v}) \xrightarrow{u} (q', \mathsf{v}')$ is a local run where v and v' are synchronized local valuations, there exists a global run $(q, \mathsf{global}(\mathsf{v})) \overset{w}{\Longrightarrow} (q', \mathsf{global}(\mathsf{v}'))$ for some $w \sim u$. Conversely, if $(q, v) \overset{u}{\Longrightarrow} (q', v')$ is a global run, then there is a local run $(q, \mathsf{local}(v)) \xrightarrow{u} (q', \mathsf{local}(v'))$.*

Proof. Follows from lemma 11 and 12. ◀

The reachability problem with respect to local semantics is defined as before: q is reachable if there is a local run $(q^{init}, \mathsf{v}^{init}) \xrightarrow{u} (q, \mathsf{v})$ for some v where $\mathsf{v}^{init} = \mathsf{local}(v^{init})$. By adding some local delays at the end of the run we can always assume that v is synchronized. Lemma 13 thus implies that the reachability problem in local semantics is equivalent to the standard one in global semantics.

3 Zone graphs

We introduce zones, a standard approach for solving reachability in timed automata. Zones are sets of valuations that can be represented efficiently using simple constraints.

Let us fix a network $\mathcal{N}$ of timed automata with k processes. Recall that each process p has a set of clocks X_p and corresponding offset variables $\widetilde{X}_p$. The set of clocks in $\mathcal{N}$ is $X = \bigcup_{i=1}^{k} X_p$. Similarly, the set of offset variables in $\mathcal{N}$ is $\widetilde{X} = \bigcup_{i=1}^{k} \widetilde{X}_p$.

3.1 Standard zone-based algorithm for reachability

Recall that in the global-time semantics, $\mathcal{N}$ has a reference clock t. A *global zone* is a set of global valuations described by a conjunction of constraints of the form $y_1 - y_2 \triangleleft c$ where $y_1, y_2 \in \widetilde{X} \cup \{t\}$, $\triangleleft \in \{<, \leq\}$ and $c \in \mathbb{Z}$. Since global valuations need to satisfy $v(\widetilde{x}) \leq v(t)$ for every offset variable $\widetilde{x} \in \widetilde{X}$, a global zone satisfies $\widetilde{x} \leq t$ for every $\widetilde{x} \in \widetilde{X}$.

Let g be a guard and R a set of clocks. We define the following operations on zones: $Z_g = \{v \mid v \models g\}$ is the set of global valuations satisfying g, $[R]Z := \{[R]v \mid v \in Z\}$ and $\overrightarrow{Z} := \{v \mid \exists v' \in Z,\ \exists \delta \in \mathbb{R}_{\geq 0} \text{ s. t. } v = v' + \delta\}$. From [4], Z_g, $[R]Z$ and $\overrightarrow{Z}$ are all zones. We say that a zone is *time-elapsed* if $Z = \overrightarrow{Z}$.

The semantics of a network of timed automata can be described in terms of global zones. For an action b, consider a set of b-transitions of respective processes $\{(q_p, g_p, R_p, q'_p)\}_{p \in dom(b)}$. Let $R = \bigcup_{p \in dom(b)} R_p$, and $g = \bigwedge_{p \in dom(b)} g_p$. Then we have a transition $(q, Z) \xRightarrow{b} (q', Z')$ where $Z' = \overrightarrow{[R](Z \cap Z_g)}$ provided that $q(p) = q_p$, $q'(p) = q'_p$ if $p \in dom(b)$, and $q'(p) = q(p)$ otherwise, and Z' is not empty. We write $\Rightarrow$ for the union over all $\xRightarrow{b}$.

The *global zone graph* $ZG(\mathcal{N})$ of a timed automaton network $\mathcal{N}$ is a transition system whose nodes are of the form (q, Z) where $q \in Q$ and Z is a time-elapsed global zone. The transition relation is given by $\Rightarrow$. The initial node is (q^{init}, Z^{init}) where q^{init} is the tuple of initial states and $Z^{init} = \overrightarrow{\{v^{init}\}}$ where v^{init} is the initial global valuation. The zone graph $ZG(\mathcal{N})$ is known to be sound and complete with respect to reachability. This means that a state q is reachable by a run of $\mathcal{N}$ iff a node (q, Z) for some non-empty Z is reachable from (q^{init}, Z^{init}) in the zone graph. As zone graphs may be infinite, an abstraction operator is used to obtain a finite quotient.

An *abstraction operator* $\mathfrak{a} : P(\mathbb{R}^X_{\geq 0}) \to P(\mathbb{R}^X_{\geq 0})$ [2] is a function from sets of valuations to sets of valuations such that $W \subseteq \mathfrak{a}(W)$ and $\mathfrak{a}(\mathfrak{a}(W)) = \mathfrak{a}(W)$. Simulation relations between valuations are a convenient way to construct abstraction operators that are correct for reachability. A *time-abstract simulation* is a relation between valuations that depends on a given network $\mathcal{N}$. We say that v_1 can be simulated by v_2, denoted $v_1 \preccurlyeq v_2$ if for every state q of $\mathcal{N}$, and every delay-action step $(q, v_1) \xRightarrow{\delta_1} \xRightarrow{b} (q', v'_1)$ there is a delay $\delta_2 \in \mathbb{R}_{\geq 0}$ such that $(q, v_2) \xRightarrow{\delta_2} \xRightarrow{b} (q', v'_2)$ and $v'_1 \preccurlyeq v'_2$. The simulation relation can be lifted to global zones: we say that Z is simulated by Z', written as $Z \preccurlyeq Z'$ if for all $v \in Z$ there exists a $v' \in Z'$ such that $v \preccurlyeq v'$. An abstraction $\mathfrak{a}$ based on $\preccurlyeq$ is defined as $\mathfrak{a}(W) = \{v \mid \exists v' \in W \text{ with } v \preccurlyeq v'\}$. The abstraction $\mathfrak{a}$ is finite if its range is finite. Given two nodes (q, Z) and (q', Z') of $ZG(\mathcal{N})$, (q, Z) is *subsumed* by (q', Z'), denoted $(q, Z) \sqsubseteq^{\mathfrak{a}} (q', Z')$, if $q = q'$ and $Z \subseteq \mathfrak{a}(Z')$.

▶ Remark 14. Our definition of zones slightly differs from the standard definition in the literature (e.g. [5]) since we use offset variables to represent clock valuations. Yet, finite time-abstract simulations from the literature [2, 11] can be adapted to our settings as a simulation over standard valuations $\overline{v}$ can be expressed as a simulation over global valuations v since for every clock x, $\overline{v}(x) = v(t) - v(\widetilde{x})$, and zones over valuations v can be translated to zones over standard valuations $\overline{v}$.

A finite abstraction $\mathfrak{a}$ allows to construct a finite *global zone graph with subsumption* for a network of timed automata $\mathcal{N}$. The construction starts from the initial node of $ZG(\mathcal{N})$. Using, say, a breadth-first-search (BFS), for every constructed node we examine all its successors in $ZG(\mathcal{N})$, and keep only those that are maximal w.r.t. to $\sqsubseteq^{\mathfrak{a}}$ relation. Computing such a zone graph with subsumption gives an algorithm for the reachability problem. However, the global zone graph, and hence the algorithm above, are sensitive to the combinatorial explosion arising from parallel composition. Global time makes any two

actions potentially dependent - the same is still true on the level of zones. Zone graphs based on the local-time semantics, as presented next, solve this problem.

3.2 Local zone graphs

The goal of this section is to introduce a concept similar to global zones and global zone graphs for local-time semantics. Recall that in the local-time semantics, each process p has a reference clock t_p.

A *local zone* is a zone over local valuations: a set of local valuations defined by constraints $y_1 - y_2 \lhd c$ where $y_1, y_2 \in \widetilde{X} \cup \{t_1, \dots t_k\}$. Recall that a local valuation v needs to satisfy: $\mathsf{v}(\widetilde{x}) \leq \mathsf{v}(t_p)$ for every process p and every $\widetilde{x} \in \widetilde{X}_p$. This means that a local zone satisfies $\widetilde{x} \leq t_p$ for every process p and every $\widetilde{x} \in \widetilde{X}_p$. We will use Z, eventually with subscripts, to range over local zones, and distinguish from global zones Z. Local zones are closed under all basic operations involved in a local step of a network of timed automata, namely: local time elapse, intersection with a guard, and reset of clocks [4]. That is, for every local zone Z:

- the set local-elapse$(\mathsf{Z}) = \{\mathsf{v} +_1 \delta_1 +_2 \cdots +_k \delta_k \mid \mathsf{v} \in \mathsf{Z},\ \delta_1, \dots, \delta_k \in \mathbb{R}_{\geq 0}\}$ is a local zone.
- for every guard g the set $\mathsf{Z}_g = \{\mathsf{v} \mid \mathsf{v} \vDash g\}$ is a local zone.
- for every set of clocks $R \subseteq X$, the set $[R]\mathsf{Z} = \{[R]\mathsf{v} \mid \mathsf{v} \in \mathsf{Z}\}$ is a local zone.

Local zones can be implemented using DBMs. Hence, they can be computed and stored as efficiently as standard zones. The proofs of the above three statements are given in the subsequent lemmas.

A zone Z is said to be in *canonical form* if every constraint defining Z is tight: in other words, removing some constraint $y_1 - y_2 \lhd c$ from Z results in a different set of valuations.

▶ **Lemma 15.** *The local zone for* local-elapse(Z) *is obtained by removing constraints of the form $t_i - \widetilde{x} \lhd c$, and $t_i - t_j \lhd c$ for all i, j and $\widetilde{x} \in \widetilde{X}$ from the canonical representation of* Z.

Proof. Let Z_1 be the set of constraints of the form $\widetilde{x} - t_i \lhd c$ and $\widetilde{x} - \widetilde{y} \lhd c$ of Z. To show Z_1 describes local-elapse(Z).

local-elapse$(\mathsf{Z}) \subseteq \mathsf{Z}_1$: Each valuation $\mathsf{v}' \in$ local-elapse(Z) is of the form $\mathsf{v}+_1\delta_1+_2\delta_2 \cdots +_k\delta_k$ for some $\mathsf{v} \in \mathsf{Z}$. Hence v' is obtained by increasing the values of reference clocks from v and keeping the other offsets the same. This gives $\mathsf{v}'(\widetilde{x}) - \mathsf{v}'(\widetilde{y}) = \mathsf{v}(\widetilde{x}) - \mathsf{v}(\widetilde{y})$ and $\mathsf{v}'(\widetilde{x}) - \mathsf{v}'(t_i) \leq \mathsf{v}(\widetilde{x}) - \mathsf{v}(t_i)$ for all component clocks $\widetilde{x}, \widetilde{y}$ and reference clocks t_i. Since $\mathsf{v} \in \mathsf{Z}$ and hence satisfies the constraints in Z_1, valuation v' satisfies them as well.

$\mathsf{Z}_1 \subseteq$ local-elapse(Z): Pick $\mathsf{v}' \in \mathsf{Z}_1$. Consider a set of constraints Z_2: $\widetilde{x} = \mathsf{v}'(\widetilde{x})$ and $t_i \leq \mathsf{v}'(t_i)$ for all component clocks $\widetilde{x}$ and process clocks t_i. A solution to $\mathsf{Z} \cap \mathsf{Z}_2$ gives a valuation v such that $\mathsf{v} \in \mathsf{Z}$ and v' is obtained by local elapse from v, and hence will imply $\mathsf{v}' \in$ local-elapse(Z). We need to show that $\mathsf{Z} \cap \mathsf{Z}_2$ is non-empty. Note that Z_2 is a canonical zone. Since both Z and Z_2 are canonical, $\mathsf{Z} \cap \mathsf{Z}_2$ is empty iff there is a constraint $y_1 - y_2 \lhd c$ from Z and a constraint $y_2 - y_1 \lhd d$ from Z_2 which contradict each other (using standard literature on zones and DBMs). Inspection of the constraints in Z_2 and the fact that $\mathsf{v}' \in \mathsf{Z}_1$ imply that this cannot happen. ◀

▶ **Lemma 16.** *For every guard, the set* v *of local valuations satisfying g is a local zone.*

Proof. We construct a set of constraints over local variables. For every constraint $x \sim c$ in g with $\widetilde{x} \in \widetilde{X}_p$, we add a constraint $t_p - \widetilde{x} \sim c$. The obtained set of constraints gives the desired zone. ◀

▶ **Lemma 17.** *For a set of clocks R, and a local zone* Z*, the set* $[R]\mathsf{Z}$ *is a zone.*

Proof. For each $\widetilde{x} \in R \cap \widetilde{X}_p$, we remove all the edges involving $\widetilde{x}$ in the distance graph of Z, and then add the constraints $\widetilde{x} - t_p \leq 0$ and $t_p \leq \widetilde{x}$. The resulting zone is $[R]\mathsf{Z}$. ◀

The operations of local time elapse, guard intersection, and reset, enable us to describe a local step $(q, \mathsf{Z}) \xrightarrow{b} (q', \mathsf{Z}')$ on the level of local zones. This is done in the same way as for global zones. Observe that a local step is indexed only by an action, as the time is taken care of by the local time elapse operation. Formally, for an action b consider a set of b-transitions $\{(q_p, g_p, R_p, q'_p)\}_{p \in dom(b)}$ of respective processes. Then we have a transition $(q, \mathsf{Z}) \xRightarrow{b} (q', \mathsf{Z}')$ for $\mathsf{Z}' = \text{local-elapse}([R](\mathsf{Z} \cap \mathsf{Z}_g \cap \mathsf{Z}_{sync}))$ where $\mathsf{Z}_g = \bigcap_{p \in dom(b)} \mathsf{Z}_{g_p}$, $\mathsf{Z}_{sync} = \{\mathsf{v} \mid \mathsf{v}(t_{p_1}) = \mathsf{v}(t_{p_2}) \text{ for } p_1, p_2 \in dom(b)\}$ and $R = [\bigcup_{p \in dom(b)} R_p]$. Intuitively, Z' is the set of valuations obtained through reset and then local-time elapse, from valuations in Z that satisfy the guard and such that the processes involved in action b are synchronised. We extend $\xrightarrow{b}$ to $(q, \mathsf{Z}) \xrightarrow{u} (q', \mathsf{Z}')$ for a sequence of actions u in the obvious way.

Using local zones, we construct a local zone graph and show that it is sound and complete for reachability testing. The only missing step is to verify the pre/post properties of runs on local zones. We say a local zone is *time-elapsed* if $\mathsf{Z} = \text{local-elapse}(\mathsf{Z})$.

▶ **Lemma 18** (Pre and post properties of runs on local zones)**.** *Let u be a sequence of actions.*

- *If $(q, \mathsf{v}) \xrightarrow{u} (q', \mathsf{v}')$ and $\mathsf{v} \in \mathsf{Z}$ for some time-elapsed local zone Z then $(q, \mathsf{Z}) \xrightarrow{u} (q', \mathsf{Z}')$ and $\mathsf{v}' \in \mathsf{Z}'$ for some local zone Z'.*
- *If $(q, \mathsf{Z}) \xrightarrow{u} (q', \mathsf{Z}')$ and $\mathsf{v}' \in \mathsf{Z}'$ then $(q, \mathsf{v}) \xrightarrow{u} (q', \mathsf{v}')$, for some $\mathsf{v} \in \mathsf{Z}$.*

Proof. This is proved by induction on the length of u.

For the pre property. Suppose u is a single action a. Then $(q, \mathsf{v}) \xrightarrow{a} (q', \mathsf{v}')$ implies there is a sequence of local steps: $(q, \mathsf{v}) \xrightarrow{\Delta} (q, \mathsf{v}_1) \xrightarrow{a} (q', \mathsf{v}_2) \xrightarrow{\Delta'} (q', \mathsf{v}')$. Since $\mathsf{v} \in \mathsf{Z}$ and Z is local time elapsed, valuation $\mathsf{v}_1 \in \mathsf{Z}$. By definition of $(q, \mathsf{Z}) \xrightarrow{a} (q', \mathsf{Z}')$ we get $\mathsf{v}' \in \mathsf{Z}'$. The induction step follows by a similar argument, and noting the fact that Z' is local-time-elapsed in $(q, \mathsf{Z}) \xrightarrow{a} (q', \mathsf{Z}')$.

For the post property. When u is a single action, the definition entails that there is a $\mathsf{v} \in \mathsf{Z}$ such that $(q, \mathsf{v}) \xrightarrow{a} \xrightarrow{\Delta} (q', \mathsf{v}')$. When $u = a_1 \dots a_n$, consider the sequence of zones $(q, \mathsf{Z}) \xrightarrow{a_1} (q_1, \mathsf{Z}_1) \cdots (q_{n-1}, \mathsf{Z}_{n-1}) \xrightarrow{a_n} (q', \mathsf{Z}')$, use a similar argument to obtain a $\mathsf{v}_{n-1} \in \mathsf{Z}_{n-1}$ and then the induction hypothesis for the shorter sequence $a_1 \dots a_{n-1}$. ◀

▶ **Definition 19** (Local zone graph)**.** *For a network of timed automata $\mathcal{N}$ the* local zone graph *of $\mathcal{N}$, denoted* $\text{LZG}(\mathcal{N})$*, is a transition system whose nodes are of the form (q, Z) where Z is a time elapsed local zone, and whose transitions are steps $(q, \mathsf{Z}) \xrightarrow{b} (q', \mathsf{Z}')$. The initial node $(q^{init}, \mathsf{Z}^{init})$ consists of the initial state q_{init} of the network and the local zone $\mathsf{Z}^{init} = \text{local-elapse}(\{\mathsf{v}^{init}\})$.*

Directly from Lemma 18 we obtain the main property of local zone graphs stated in [4]. This allows us to use local zone graphs for reachability testing.

▶ **Theorem 20.** *For a given network of timed automata $\mathcal{N}$, there is a run of the network reaching a state q iff for some non-empty local zone Z, node (q, Z) is reachable in* $\text{LZG}(\mathcal{N})$ *from its initial node.*

Notice that $\text{LZG}(\mathcal{N})$ may still be infinite and it cannot be used directly for reachability checking. The solution in Remark 14 does not apply to local zones due to the multiple reference clocks. This problem will be addressed in Section 5. We first focus on important properties of the local-time zone graph w.r.t. concurrency.

4 Why are local zone graphs better than global zone graphs?

The important feature about local zone graphs, as noticed in [4], is that two transitions on actions with disjoint domains commute (see Figure 1).

▶ **Lemma 21** (Commutativity on local zones [4]). *Suppose* $dom(a) \cap dom(b) = \emptyset$. *If* $(q, \mathsf{Z}) \xrightarrow{ab} (q', \mathsf{Z}')$ *then* $(q, \mathsf{Z}) \xrightarrow{ba} (q', \mathsf{Z}')$.

Proof. Suppose $(q, \mathsf{Z}) \xrightarrow{ab} (q', \mathsf{Z}'_{ab})$. We will show that there is $(q, \mathsf{Z}) \xrightarrow{ba} (q', \mathsf{Z}'_{ba})$, and that $\mathsf{Z}'_{ab} \subseteq \mathsf{Z}'_{ba}$. By symmetry this will show the lemma.

Take $\mathsf{v}' \in \mathsf{Z}'_{ab}$. Using the backward (post) property of steps on zones (Lemma 18) we get a run:

$$(q, \mathsf{v}) \xrightarrow{a} (q_a, \mathsf{v}_a) \xrightarrow{\Delta_a} (q_a, \mathsf{v}'_a) \xrightarrow{b} (q', \mathsf{v}_b) \xrightarrow{\Delta_b} (q', \mathsf{v}'),$$

where $\mathsf{v} \in \mathsf{Z}$. Using commutation on the level of runs, Lemma 7, we get a run

$$(q, \mathsf{v}) \xrightarrow{\Delta'_b} (q, \mathsf{v}'_b) \xrightarrow{b} (q_b, \mathsf{v}''_b) \xrightarrow{\Delta'_a} (q_b, \mathsf{v}''_a) \xrightarrow{a} (q', \mathsf{v}'''_a) \xrightarrow{\Delta''_a} (q', \mathsf{v}')$$

Now using the forward (pre) property of steps on zones from Lemma 18 we obtain that Z_{ba} exists and $\mathsf{v}' \in \mathsf{Z}_{ba}$. ◀

From the above lemma, we get the following property.

▶ **Corollary 22.** *If* $(q, \mathsf{Z}) \xrightarrow{u} (q', \mathsf{Z}')$ *and* $u \sim w$, *then* $(q, \mathsf{Z}) \xrightarrow{w} (q', \mathsf{Z}')$

Thus starting from a local zone, all equivalent interleavings of a sequence of actions u end up in the same local zone. This is in stark contrast to the global zone graph, where each interleaving results in a possibly different global zone. Let

$$MZ(q, Z, u) = \{v' \mid \exists v \in Z,\ \exists w,\ w \sim u \text{ and } (q, v) \stackrel{w}{\Longrightarrow} (q', v')\}$$

denote the union of all these global zones. Salah et al. [16] have shown that, surprisingly, $MZ(q, Z, u)$ is always a global zone. We call it *aggregated zone*, and the notation MZ is in the memory of Oded Maler. In the same work, this observation was extended to an algorithm for *acyclic timed automata* that from time to time merged zones reached by equivalent paths to a single global zone. We prove below that this aggregated zone can, in fact, be obtained directly in the local zone graph: the aggregated (global) zone is exactly the set of synchronized valuations obtained after executing u in the local zone semantics. Here we need some notation: let Z be a global zone and Z a local zone; define $\mathrm{sync}(\mathsf{Z}) = \{\mathsf{v} \in \mathsf{Z} \mid \mathsf{v} \text{ is synchronized}\}$; $\mathsf{local}(Z) = \{\mathsf{local}(v) \mid v \in Z\}$ and $\mathsf{global}(\mathrm{sync}(\mathsf{Z})) = \{\mathsf{global}(\mathsf{v}) \mid \mathsf{v} \in \mathrm{sync}(\mathsf{Z})\}$.

▶ **Lemma 23.** *For every global zone* Z *and local zone* Z: $\mathrm{sync}(\mathsf{Z})$ *and* $\mathsf{local}(Z)$ *are local zones and* $\mathsf{global}(\mathrm{sync}(\mathsf{Z}))$ *is a global zone.*

Proof. $\mathrm{sync}(\mathsf{Z})$ is the local zone $\mathsf{Z} \wedge \bigwedge_{i,j}(t_i = t_j)$; $\mathsf{local}(Z)$ is the local zone obtained by replacing t with some t_i in each constraint, and adding the constraints $\bigwedge_{i,j}(t_i = t_j)$; $\mathsf{global}(\mathrm{sync}(\mathsf{Z}))$ is obtained by replacing each t_i with t in each constraint of Z. ◀

▶ **Theorem 24.** *Consider a state* q, *a sequence of actions* u *and a time elapsed global zone* Z. *Consider the local zone* $\mathsf{Z} = \text{local-elapse}(\mathsf{local}(Z))$. *If* $(q, \mathsf{Z}) \xrightarrow{u} (q', \mathsf{Z}')$, *we have* $MZ(q, Z, u) = \mathsf{global}(\mathrm{sync}(\mathsf{Z}'))$, *otherwise* $MZ(q, Z, u) = \emptyset$.

Proof. Pick $v' \in MZ(q, Z, u)$. There exists $w \sim u$, $v \in Z$ and a global run $(q, v) \xRightarrow{w} (q', v')$. From Lemma 13, there exists a local run $(q, \mathsf{local}(v)) \xrightarrow{w} (q', \mathsf{local}(v'))$. By assumption, $\mathsf{local}(v) \in \mathsf{Z}$. Hence from the pre property of local zones (Lemma 18), there exists $(q, \mathsf{Z}) \xrightarrow{w} (q', \mathsf{Z}_w)$ such that $\mathsf{local}(v') \in \mathsf{Z}_w$. As $\mathsf{local}(v')$ is synchronized, we get $\mathsf{local}(v') \in \mathrm{sync}(\mathsf{Z}_w)$. But, by Corollary 22, $\mathsf{Z}_w = \mathsf{Z}'$. This proves $\mathsf{local}(v') \in \mathrm{sync}(\mathsf{Z}')$ and hence $v' \in \mathsf{global}(\mathrm{sync}(\mathsf{Z}'))$.

For the other direction take $v' \in \mathsf{global}(\mathrm{sync}(\mathsf{Z}'))$. As $(q, \mathsf{Z}) \xrightarrow{u} (q', \mathsf{Z}')$, by post property of local zones (Lemma 18) there is a local run $(q, \mathsf{v}_u) \xrightarrow{u} (q', \mathsf{local}(v'))$ for some $\mathsf{v}_u \in \mathsf{Z}$. Since $\mathsf{v}_u \in \mathsf{Z}$, it is obtained by a local time elapse from some $\mathsf{v} \in \mathsf{local}(Z)$. Hence v is synchronized and $\mathsf{global}(\mathsf{v}) \in Z$. From Lemma 13 we get that for some $w \sim u$ there is a global run $(q, \mathsf{global}(\mathsf{v})) \xRightarrow{w} (q', v')$. Hence $v' \in MZ(q, Z, u)$. ◀

Theorem 24 gives an efficient way to compute aggregated zones: it is sufficient to compute local zone graphs. Computing local zone graphs is not more difficult than computing global zone graphs. But, surprisingly, the combinatorial explosion due to interleaving does not occur in local zone graphs, thanks to the theorem above. Hence, this gives an incentive to work with local zone graphs instead of global zone graphs.

This contrasts with the aggregation algorithm in [16] which requires to store all the paths to a global zone and detect situations where zones can be merged, that is, when all the equivalent permutations have been visited. Another important limitation of the algorithm in [16] is that it can only be applied to acyclic zone graphs. If local zone graphs can be computed for general timed automata (which contain cycles), we can get to use the aggregation feature for all networks (and not only acyclic ones). To do this, there is still a major problem left: local zone graphs could be infinite when the automata contain cycles.

5 Making local zone graphs finite

The standard approach to make a global zone graph finite is to use a subsumption operation between global zones [2, 11]. Such a subsumption operation is usually based on a finite index simulation between (global) valuations parameterized by certain maximum constants occurring in guards. We first discuss technical problems that arise when we lift these simulations to local valuations. In the paper introducing local time semantics and local zone graphs [4] a notion of a *catch-up equivalence* between local valuations is defined. This equivalence is a finite index simulation. So one could, in theory, construct a finite local zone graph using catch-up subsumption as a finite abstraction. Unfortunately, the question of effective algorithms for catch-up subsumption was left open in [4], and, to the best of our knowledge, there is no efficient procedure for catch-up subsumption.

Another finite abstraction of the local zone graph was proposed by Minea [14, 15]. We however believe that Minea's approach carries a flaw, and a different idea is needed to get finiteness. Minea's approach is founded on an equivalence between local valuations in the lines of the region equivalence [1]. Let $c_{\max}$ be the maximum constant appearing among the guards of a timed automata network. Two local valuations v_1 and v_2 are said to be equivalent, written as $\mathsf{v}_1 \simeq_{\mathrm{reg}} \mathsf{v}_2$ if for all variables $\alpha, \beta \in \widetilde{X} \cup \{t_1, \ldots, t_k\}$ (note that all reference clocks are included):

- either $\lfloor \mathsf{v}_1(\alpha) - \mathsf{v}_1(\beta) \rfloor = \lfloor \mathsf{v}_2(\alpha) - \mathsf{v}_2(\beta) \rfloor$,
- or $\lfloor \mathsf{v}_1(\alpha) - \mathsf{v}_1(\beta) \rfloor > c_{\max}$ and $\lfloor \mathsf{v}_2(\alpha) - \mathsf{v}_2(\beta) \rfloor > c_{\max}$,
- or $\lfloor \mathsf{v}_1(\alpha) - \mathsf{v}_1(\beta) \rfloor < -c_{\max}$ and $\lfloor \mathsf{v}_2(\alpha) - \mathsf{v}_2(\beta) \rfloor < -c_{\max}$.

It is claimed (in Proposition 6 of [14]) that this equivalence is preserved over local time elapse: for every process p, and for $\delta \geq 0$ there exists $\delta' \geq 0$ such that $\mathsf{v}_1 +_p \delta \simeq_{\mathrm{reg}} \mathsf{v}_2 +_p \delta'$.

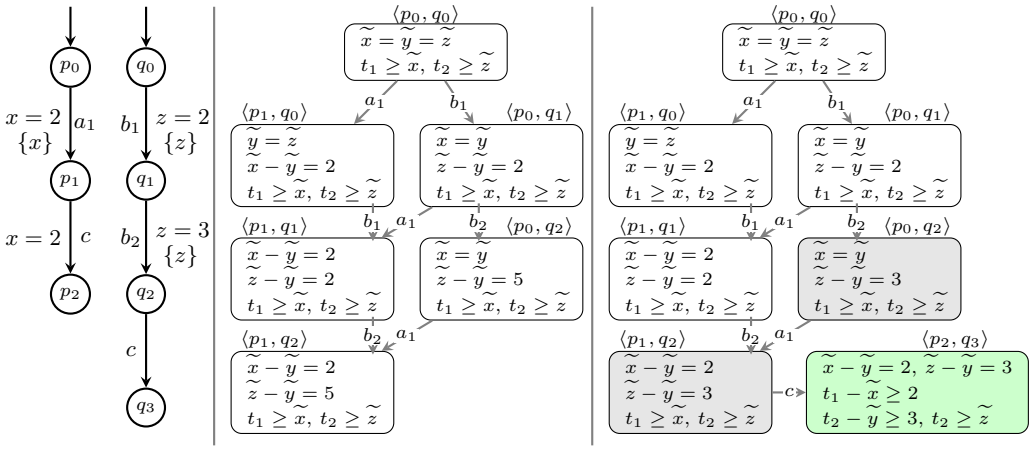

Figure 2 *Left:* network $\langle A_1, A_2 \rangle$; *Middle:* local zone graph; *Right:* maximized local zone graph of [14]. State (p_2, q_3) is not reachable in local zone graph, but becomes reachable after maximization.

However, this is not true, as we now exhibit a counter-example. Consider 2 processes with clocks $X_1 = \{x\}$, $X_2 = \{y\}$. This gives $\widetilde{X} = \{\widetilde{x}, \widetilde{y}\}$ and $T = \{t_1, t_2\}$. Let $c_{\max} = 3$. Define local valuations $\mathsf{v}_1 : \widetilde{x} = 0, t_1 = 0, \widetilde{y} = 0, t_2 = 4$ and $\mathsf{v}_2 : \widetilde{x} = 0, t_1 = 0, \widetilde{y} = 0, t_2 = 5$.

Note that the differences in v_1 are either $0, 4$ or -4 and the corresponding differences in v_2 are $0, 5$ or -5. Hence by definition, $\mathsf{v}_1 \simeq_{\text{reg}} \mathsf{v}_2$. Consider valuation $\mathsf{v}_1 +_1 2$ obtained from v_1 by local delay of 2 units in component 1, that is $\mathsf{v}_1 +_1 2 : \widetilde{x} = 0, t_1 = 2, \widetilde{y} = 0, t_2 = 4$ Observe that in $\mathsf{v}_1 +_1 2$, the difference $t_1 - \widetilde{x} = 2$ and $t_2 - t_1 = 2$ which are both smaller than $c_{\max}$. We claim there is no local delay δ' such that $\mathsf{v}_1 +_1 2 \simeq_{\text{reg}} \mathsf{v}_2 +_1 \delta'$. Valuation $\mathsf{v}_2 +_1 \delta'$ is given by $\widetilde{x} = 0, t_1 = \delta', \widetilde{y} = 0, t_2 = 5$. If $\mathsf{v}_1 +_1 2 \simeq_{\text{reg}} \mathsf{v}_2 +_1 \delta'$, we need $\delta' = 2$ (due to difference $t_1 - \widetilde{x}$) and $5 - \delta' = 2$ (due to difference $t_2 - t_1$). This is not possible.

The main problem is that the above equivalence "forgets" actual values when the difference between reference clocks is above c_{max}. Even if this difference is bigger than the maximum constant, local delays can bring them within the constant c_{max}. Such a situation does not arise in the global semantics, as there is a single reference clock.

In [14] a widening operator on local zones based on $\simeq_{\text{reg}}$ is used for finiteness: given a canonical representation of a zone Z, the *maximized zone* with respect to $c_{\max}$ is obtained by changing every constraint $y_1 - y_2 \triangleleft c$ to $y_1 - y_2 < \infty$ if $c > c_{\max}$, and to $y_1 - y_2 < -c_{\max}$ if $c < -c_{\max}$. In the local zone graph construction, each local zone is maximized and inclusion between maximized zones is used for termination. Figure 2 gives an example of a network $\langle A_1, A_2 \rangle$ where the maximized local zone graph is unsound. This is shown by making use of the valuations v_1 and v_2 above. We also add an extra clock z in component A_2 for convenience. Note that $c_{\max} = 3$. Although clock y does not appear in A_2, one can assume that there are other transitions from q_0 that deal with y (we avoid illustrating these transitions explicitly). In the discussion below, $\mathsf{v}_1, \mathsf{v}_2$ are valuations restricted to $\widetilde{x}, \widetilde{y}, t_1$ and t_2. In order to reach the state p_2, the synchronization action c needs to be taken: transition sequence $a_1 c$ requires c to be taken at global time 4, and transition sequence $b_1 b_2 c$ requires c at global time 5. Hence c is not enabled in the network. This is witnessed by c not being enabled in the local zone graph (middle picture). Valuation v_2 is present in the zone reached after $b_1 b_2$. The maximized local zone graph is shown on the right. Zones where maximization makes a difference are shaded gray. In particular, the zone $b_1 b_2$ on maximization adds valuation v_1, from which $a_1 c$ is enabled, giving a zone in the maximized local zone graph with state p_2.

5.1 Synchronized valuations for subsumption

A finite abstraction of the differences between reference clocks constitutes the main challenge in obtaining a finite local zone graph. We propose a different solution which bypasses the need to worry about such differences: *restrict to synchronized valuations for subsumption.*

Subsumptions over global zones are well studied [1, 2, 11]. Taking an off-the-shelf finite abstraction $\mathfrak{a}$ and a subsumption $\sqsubseteq^{\mathfrak{a}}$ between global zones, we want to do the following: given two local zones Z_1 and Z_2 we perform a subsumption test $\mathsf{global}(\mathrm{sync}(\mathsf{Z}_1)) \sqsubseteq^{\mathfrak{a}} \mathsf{global}(\mathrm{sync}(\mathsf{Z}_2))$. By finiteness of the abstraction, we will get only finitely many local zones Z with incomparable $\mathsf{global}(\mathrm{sync}(\mathsf{Z}))$. In order to do this, we need the crucial fact that $\mathsf{global}(\mathrm{sync}(\mathsf{Z}))$ is a zone (shown in Lemma 23). Given two configurations $s := (q, \mathsf{Z})$ and $s' := (q', \mathsf{Z}')$ of the local zone graph, we write $s \sqsubseteq^{\mathfrak{a}}_{sync} s'$ if $q = q'$ and $\mathsf{global}(\mathrm{sync}(\mathsf{Z})) \sqsubseteq^{\mathfrak{a}} \mathsf{global}(\mathrm{sync}(\mathsf{Z}'))$.

▶ **Definition 25** (Local sync graph). *Let $\mathfrak{a}$ be a finite abstraction over global zones. A* local sync graph *G of a network of timed automata $\mathcal{N}$ based on $\mathfrak{a}$, is a tree and a subgraph of* $\mathrm{LZG}(\mathcal{N})$ *satisfying the following conditions:*

- ***C0*** *every node of G is labeled either* covered *or* uncovered*;*
- ***C1*** *the initial node of* $\mathrm{LZG}(\mathcal{N})$ *belongs to G and is labeled uncovered;*
- ***C2*** *every node is reachable from the initial node;*
- ***C3*** *for every uncovered node s, all its successor transitions $s \xrightarrow{a} s'$ occurring in* $\mathrm{LZG}(\mathcal{N})$ *should be present in G;*
- ***C4*** *for every covered node $s \in G$ there is an uncovered node $s' \in G$ such that $s \sqsubseteq^{\mathfrak{a}}_{sync} s'$. A covered node has no successors.*

The above definition essentially translates to this algorithm: explore the local zone graph say in a BFS fashion, and subsume (cover) using $\sqsubseteq^{\mathfrak{a}}_{sync}$ (similar to algorithm for global zone graph as described in page 9). Local sync graphs are not unique, since the final graph depends on the order of exploration. Every local sync graph based on a finite abstraction $\mathfrak{a}$ is finite. Theorem 26 below states that local sync graphs are sound and complete for reachability, and this algorithm is correct.

▶ **Theorem 26** (Soundness and completeness of local sync graphs). *A state q is reachable in a network of timed automata $\mathcal{N}$ iff a node (q, Z), with Z non-empty, is reachable from the initial node in a local sync graph for $\mathcal{N}$.*

Proof. If (q, Z) is reachable in a local sync graph then it is trivially reachable in the local zone graph and the (backward) implication follows from soundness of local zone graphs (Theorem 20). For the other direction which proves completeness of local sync graphs, let us take a global run: $(q_{init}, v_{init}) \xRightarrow{\delta_1, b_1} (q_1, v_1) \cdots \xRightarrow{\delta_n, b_n} (q_n, v_n)$. Take a local sync graph G, based on abstraction $\mathfrak{a}$ coming from a simulation $\preccurlyeq$. By induction on i, for every (q_i, v_i) we will find an uncovered node (q_i, Z_i) of G, and a synchronized local valuation $\mathsf{v}_i \in \mathsf{Z}_i$ such that $v_i \preccurlyeq \mathsf{global}(\mathsf{v}_i)$. This proves completeness, since every reachable (q_i, v_i) will have a representative node in the local sync graph.

The induction base is immediate, so let us look at the induction step. Consider the global step $(q_i, v_i) \xRightarrow{\delta_i, b_i} (q_{i+1}, v_{i+1})$. Since $v_i \preccurlyeq \mathsf{global}(\mathsf{v}_i)$, there is a delay δ'_i such that $(q_i, \mathsf{global}(\mathsf{v}_i)) \xRightarrow{\delta'_i, b_i} (q_{i+1}, v'_{i+1})$ and $v_{i+1} \preccurlyeq v'_{i+1}$. As the global delay δ'_i can be thought of a sequence of local delays, we have local run $(q_i, \mathsf{v}_i) \xrightarrow{b_i} (q_{i+1}, \mathsf{v}'_{i+1})$, where $\mathsf{v}'_{i+1} = \mathsf{local}(v'_{i+1})$. Note that v'_{i+1} is synchronized and $v_{i+1} \preccurlyeq \mathsf{global}(\mathsf{v}'_{i+1})$. From the pre-property of local zones (Lemma 18) there exists a transition $(q_i, \mathsf{Z}_i) \xrightarrow{b_i} (q_{i+1}, \mathsf{Z}'_{i+1})$ with $\mathsf{v}'_{i+1} \in \mathsf{Z}'_{i+1}$, in fact, $\mathsf{v}'_{i+1} \in \mathrm{sync}(\mathsf{Z}'_{i+1})$. If $(q_{i+1}, \mathsf{Z}'_{i+1})$ is uncovered, take v'_{i+1} for v_{i+1} and Z'_{i+1} for Z_{i+1} (needed

by the induction step). Otherwise, from condition C4, there is an uncovered node $(q_{i+1}, \mathsf{Z}''_{i+1})$ such that $\mathsf{global}(\mathrm{sync}(\mathsf{Z}'_{i+1})) \preccurlyeq \mathsf{global}(\mathrm{sync}(\mathsf{Z}''_{i+1}))$. This gives $\mathsf{v}''_{i+1} \in \mathrm{sync}(\mathsf{Z}''_{i+1})$ such that $\mathsf{global}(\mathsf{v}'_{i+1}) \preccurlyeq \mathsf{global}(\mathsf{v}''_{i+1})$. Now take v''_{i+1} for v_{i+1} and Z''_{i+1} for Z_{i+1}. ◀

6 Experiments

We have implemented the construction of local sync graphs in our prototype TChecker [10] and compared it with two implementations of the usual global zone graph method: TChecker and UPPAAL [12, 3], the state-of-the-art verification tool for timed automata. The three implementations use a breadth-first search with subsumption, and the $\sqsubseteq^{\mathfrak{a}}_{sync}$ subsumption in the case of local sync graph. Table 1 presents results of our experiments on standard models from the literature (except "Parallel" that is a model we have introduced).

Local sync graphs yield no gain on 3 standard examples (which are not given in Table 1): "CSMA/CD", "FDDI" and "Fischer". In these models, the three algorithms visit and store the same number of nodes. The reason is that for "CSMA/CD" and "FDDI", replacing each local zone Z in the local zone graph by its set of synchronized valuations $sync(\mathsf{Z})$ yields exactly the zone graph. In the third model, "Fischer", every control state appears at most once in the global zone graph. So there is no hope to achieve any gain with our technique. This is due to the fact that doing ab or ba results in two different control states in the automaton. So "Fischer" is out of the scope of our technique.

In contrast, we observe significant improvements on other standard models (Table 1). Observe that due to subsumption, the order in which nodes are visited impacts the total number of visited nodes. UPPAAL and our prototype TChecker (Global ZG column) may not visit the same number of nodes despite the fact that they implement the same algorithm. In our prototype we use the same order of exploration for Global ZG and Local ZG. "CorSSO" and "Critical region" are standard examples from the literature. "Dining Philosophers" is a modification of the classical problem where a philosopher releases her left fork if she cannot take her right fork within a fixed amount of time [13]. "Parallel" is a model we have introduced, where concurrent processes compete to access a resource in mutual exclusion. We observe an order of magnitude gains for most of these four models. The reason is that in most states when two processes can perform actions a and b, doing ab or ba leads to the same control-state of the automaton. Hence, a difference between ab and ba (if any) is encoded in distinct zones Z_{ab} and Z_{ba} obtained along these two paths in the global zone graph. In contrast, the two paths result in the same zone Z (containing both Z_{ab} and Z_{ba}) in the local sync graph. In consequence, our approach that combines the local zone graph and abstraction using synchronized zones is very efficient in this situation.

7 Conclusions

We have revisited local-time semantics of timed automata and local zone graphs. We have discovered a very useful fact that local zones calculate aggregated zones: global zones that are unions of all the zones obtained by equivalent executions [16]. We have used this fact as a theoretical foundation for an algorithm constructing local zone graphs and using subsumption on aggregated zones at the same time.

We have shown that, unfortunately, subsumption operations on local zones proposed in the literature do not work. We have proposed a new subsumption for local zone graphs based on standard abstractions for timed automata, applied on synchronized zones. The restriction to synchronized zones is crucial as standard abstractions cannot handle multiple reference

Models	UPPAAL			Global ZG			Local ZG		
(# processes)	visited	stored	sec.	visited	stored	sec.	visited	stored	sec.
CorSSO 3	64378	61948	1.48	64378	61948	1.41	1962	1962	0.05
CorSSO 4	timeout			timeout			23784	23784	0.69
CorSSO 5	timeout			timeout			281982	281982	16.71
Critical reg. 4	78049	53697	1.45	75804	53697	2.27	44490	28400	2.40
Critical reg. 5	timeout			timeout			709908	389614	75.55
Dining Phi. 7	38179	38179	34.61	38179	38179	7.28	2627	2627	0.32
Dining Phi. 8	timeout			timeout			8090	8090	1.65
Dining Phi. 9	timeout			timeout			24914	24914	7.10
Dining Phi. 10	timeout			timeout			76725	76725	30.20
Parallel 6	11743	11743	4.82	11743	11743	1.09	256	256	0.02
Parallel 7	timeout			timeout			576	576	0.04
Parallel 8	timeout			timeout			1280	1280	0.11

Table 1 Experimental results obtained by running UPPAAL and our prototype TChecker (Global ZG and Local ZG) on a MacBook Pro 2013 with 4 2.4GHz Intel Core i5 and 16 GB of memory. The timeout is 90 seconds. For each model we report the number of concurrent processes.

clocks. A direction for future work is to find abstractions for local zones.

Our algorithm is the first efficient implementation of local time zone graphs and aggregated zones. Experimental results show an order of magnitude gain with respect to state-of-the-art algorithms on several standard examples.

As future work, we plan to develop partial-order techniques taking advantage of the high level of commutativity in local zone graphs. Existing methods are not directly applicable in the timed setting. In particular, contrary to expectations, actions with disjoint domains may not be independent (in a partial order sense) in a local zone graph [14]. Thus, it will be very interesting to understand the structure of local zone graphs better. A recent partial-order method proposed for timed-arc Petri nets [6] gives a hope that such obstacles can be overcome. For timed networks with cycles, the interplay of partial-order and subsumption adds another level of difficulty.

References

1 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

2 Gerd Behrmann, Patricia Bouyer, Kim G. Larsen, and Radek Pelánek. Lower and upper bounds in zone-based abstractions of timed automata. *International Journal on Software Tools for Technology Transfer*, 8(3):204–215, 2006.

3 Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Hakansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. UPPAAL 4.0. In *QEST*, pages 125–126. IEEE Computer Society, 2006.

4 Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Wang Yi. Partial order reductions for timed systems. In *CONCUR*, volume 1466 of *Lecture Notes in Computer Science*, pages 485–500, 1998.

5 Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *ACPN 2003*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2003.

6 Frederik M. Bønneland, Peter Gjøl Jensen, Kim Guldstrand Larsen, Marco Muñiz, and Jirí Srba. Start pruning when time gets urgent: Partial order reduction for timed systems. In *CAV*, volume 10981 of *Lecture Notes in Computer Science*, pages 527–546. Springer, 2018.

7 Conrado Daws and Stavros Tripakis. Model checking of real-time reachability properties using abstractions. In *TACAS*, volume 1384 of *Lecture Notes in Computer Science*, pages 313–329. Springer, 1998.

8 D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*, pages 197–212, 1990.

9 Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Fast algorithms for handling diagonal constraints in timed automata. *CoRR*, abs/1904.08590, 2019.

10 F. Herbreteau and G. Point. TChecker. `https://github.com/fredher/tchecker`, v0.2 - April 2019.

11 Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Better abstractions for timed automata. In *LICS*, pages 375–384. IEEE Computer Society, 2012.

12 Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *STTT*, 1(1-2):134–152, 1997.

13 Denis Lugiez, Peter Niebert, and Sarah Zennou. A partial order semantics approach to the clock explosion problem of timed automata. *Theor. Comput. Sci.*, 345(1):27–59, 2005.

14 Marius Minea. Partial order reduction for model checking of timed automata. In *CONCUR*, volume 1664 of *Lecture Notes in Computer Science*, pages 431–446. Springer, 1999.

15 Marius Minea. *Partial Order Reduction for Verification of Timed Systems*. PhD thesis, School of Computer Science, Carnegie Mellon University Pittsburgh, PA 15213, 1999.

16 Ramzi Ben Salah, Marius Bozga, and Oded Maler. On interleaving in timed automata. In *CONCUR*, volume 4137 of *Lecture Notes in Computer Science*, pages 465–476. Springer, 2006.

17 Marcelo Sousa, César Rodríguez, Vijay D'Silva, and Daniel Kroening. Abstract interpretation with unfoldings. In *CAV*, pages 197–216, 2017.