Abstractions for the local-time semantics of timed automata: a foundation for partial-order methods

R. Govind
Dept. of Computer Science and Engineering, IIT Bombay
Mumbai, India
govindr@cse.iitb.ac.in

Frédéric Herbreteau
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR5800
Talence, France
fh@labri.fr

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

Igor Walukiewicz
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR5800
Talence, France
igw@labri.fr

ABSTRACT

A timed network is a parallel composition of timed automata synchronizing on common actions. We develop a methodology that allows to use partial-order methods when solving the reachability problem for timed networks. It is based on a local-time semantics proposed by [Bengtsson et al. 1998]. A new simulation based abstraction of local-time zones is proposed. The main technical contribution is an efficient algorithm for testing subsumption between local-time zones with respect to this abstraction operator. The abstraction is not finite for all networks. It turns out that, under relatively mild conditions, there is no finite abstraction for local-time zones that works for arbitrary timed networks. To circumvent this problem, we introduce a notion of a bounded-spread network. The spread of a network is a parameter that says how far the local times of individual processes need to diverge. For bounded-spread networks, we show that it is possible to use subsumption and partial-order methods at the same time.

CCS CONCEPTS

• **Theory of computation** → **Logic**; **Verification by model checking**.

KEYWORDS

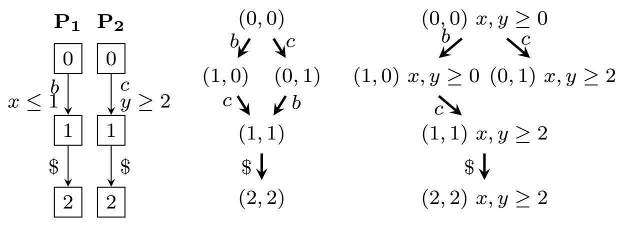
Timed automata, local-time semantics, abstraction, partial-order reduction

ACM Reference Format:
R. Govind, Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. 2022. Abstractions for the local-time semantics of timed automata: a foundation for partial-order methods. In *37th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS) (LICS '22), August 2–5, 2022, Haifa, Israel.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3531130.3533343

1 INTRODUCTION

The reachability problem for timed automata [3] is to decide if a given automaton has an execution from an initial to a final state.

Publication rights licensed to ACM. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of a national government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.
LICS '22, August 2–5, 2022, Haifa, Israel
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9351-5/22/08…$15.00
https://doi.org/10.1145/3531130.3533343



(a) Network **(b) Untimed product** **(c) Timed product**

Figure 1: A network of two processes. Timing constraints break the diamond formed by actions involving different processes.

Usually, a model is given as a network of timed automata working in parallel and synchronizing on common actions. It is tempting to exploit the concurrency information provided by such a representation to speed up reachability testing. For untimed systems, partial-order methods [2, 15, 19, 20, 22, 35, 37, 38] can provide exponential improvements. The presence of time greatly complicates the picture because individual automata may synchronize implicitly via time. In this work we extend the classical zone based approach to the reachability problem so that partial-order reduction methods become applicable.

Let us explain the challenge on a simple example. Figure 1a shows a network of two processes. The first process does a local action b, the second a local action c, and then they synchronize on action \$. If we ignore the timing constraints, the graph of all executions of this system has a *diamond*: since b and c are executed on different processes they commute, so the sequence bc leads to the same state as cb as shown in Figure 1b. The timing constraints break this diamond: the sequence cb is impossible since doing c requires to wait at least 2 time units, and then it is too late for doing b that needs to be executed within 1 time unit from the start, as in Figure 1c. This is a major obstacle for applying partial-order methods in the timed automata setting.

Before addressing this obstacle let us review how the reachability problem of a single timed automaton is solved. The most efficient solutions to the reachability problem construct an explicit graph, that we call here an *abstract zone graph with subsumptions.*

$$\begin{tikzcd}
 & (0,0)\ t_1=0, t_2=0 \arrow[dl, "b"'] \arrow[dr, "c"] & \\
(1,0)\ t_1=0, t_2=0 \arrow[dr, "c"'] & & (0,1)\ t_1=0, t_2=2 \arrow[dl, "b"] \\
 & (1,1)\ t_1=0, t_2=2 \arrow[d, "\$"'] & \\
 & (2,2)\ t_1=2, t_2=2 &
\end{tikzcd}$$

Figure 2: Diamonds are recovered in local-time semantics. In the rightmost path the local times of the two process differ.

Zones [17] are special convex sets of clock valuations with the property that a set of valuations reachable from a zone is once again a zone. The nodes of the graph represent the zones that are reachable by the transitions of the automaton. For some timed automata, there could be infinitely many reachable zones. This is why *abstractions* [6, 17], such as $Extra_{LU}$ or $\mathfrak{a}_{\preccurlyeq LU}$, are used to "abstract" a finite number of representative sets. Finally, only zones whose abstractions are maximal with respect to inclusion are kept during exploration. This technique is called *subsumption* [17], and it is essential for efficiency.

When applying this method to networks of timed automata, the state explosion problem occurs. For untimed systems this problem can be alleviated either by partial-order methods, or by symbolic methods based on BDDs or SAT-solving. For timed systems BDD and SAT based solutions [4, 5, 7, 18, 33, 34, 39] have been tried with mixed results. Here we pursue a partial-order approach to tackle the state explosion problem.

Partial-order methods limit the search space by using the diamonds present in the graph of executions. In our example from Figure 1b, it is enough to explore the sequence $bc\$$ as thanks to the diamond we are sure that the sequence $cb\$$ leads to the same state. For more complicated cases, this approach can give exponential gains in time as well in the size of the graph to be explored. In order to apply partial-order methods for timed systems it is essential to recover the diamonds lost due to implicit synchronization caused by time constraints. Otherwise, choosing $cb\$$ as a representative for $bc\$$ in Figure 1c would lead to incompleteness. Solutions proposed in the literature consider only diamonds where time does not elapse [9, 10, 30], or try to deduce which diamonds are still bound to stay despite time constraints [16, 24]. Here we develop a set of theoretical results permitting a much wider use of partial-order methods in constructing abstract zone graphs with subsumption.

Our starting point is the local-time semantics [8] for networks of timed automata, that addresses exactly the diamond problem by making time local to each process. The processes are required to synchronize their times when performing common actions. As a result, local-time semantics can be actually used to solve the reachability problem even if it allows more behaviours as compared to the standard global-time semantics. Moreover, actions executing on different processes are independent as there are no implicit synchronizations on time.

Let us revisit our example to see how diamonds are recovered thanks to local-time. Figure 2 illustrates two executions under the local-time semantics. The two processes have their local and independent times represented by clocks t_1 and t_2, respectively. The path $bc\$$ is still feasible as before, by keeping the local times of process P_1 and P_2 synchronized. But now, $cb\$$ becomes feasible as well. P_2 may delay by 2 time units and do c while P_1 does not delay at all. Then, P_1 can do b, and then delay 2 time units to synchronize its time with P_2 and enable the common action \$. As in the standard (global-time) setting, there is a notion of a *local-zone*, and one can try to use the local-zone graph for checking reachability. The advantage is that the local-zone graph has diamonds. However, this graph may be infinite.

What is lacking to make the local-time approach algorithmically interesting, is an efficient abstraction operator that would guarantee finiteness of abstract local-zone graphs. An abstraction operator has been proposed in [8], but as we show here, the associated decision problem is PSPACE-hard (Proposition 3.11), so there is little hope that it can be used to give an efficient solution.

To sum up, to be able to use partial-order methods with the help of local-time semantics, we need to find an abstraction operator for local-zones that:

(1) preserves reachability,
(2) leads to a finite abstract local-zone graph,
(3) is efficient algorithmically,
(4) preserves diamonds of actions from distinct processes.

The first two conditions are required for correctness and termination of an exploration algorithm. The third is essential to be competitive with existing solutions: computing an inclusion between two abstracted zones should be easier than solving the reachability problem in the first place. The fourth condition is needed to apply partial-order methods. The formalization of the fourth condition is actually weaker than requiring diamonds to exist in the abstract local-zone graph with subsumptions. The latter property would be much too strong to demand; c.f. Figure 5.

Our first result is an extension of the well-known $\mathfrak{a}_{\preccurlyeq LU}$ abstraction for global-time semantics [6, 26] to the local-time setting (Theorem 6.5). We call it $\mathfrak{a}^{\star}_{\preccurlyeq LU}$. The main technical result is an efficient algorithm for testing inclusion $\mathfrak{a}^{\star}_{\preccurlyeq LU}(Z) \subseteq \mathfrak{a}^{\star}_{\preccurlyeq LU}(Z')$ in time $O((|X|+n)^2)$, where $|X|$ is the number of clocks, and n is the number of processes in the network (Theorem 6.8). This complexity is essentially the same as in the global-time setting, with the factor n coming due to extra clocks added by the local-time semantics.

Unfortunately, the $\mathfrak{a}^{\star}_{\preccurlyeq LU}$ abstraction is not finite, that is, it does not satisfy property (2). Actually, we observe a strong negative result: there is no simulation based abstraction operator satisfying properties (1), (2) and (4) at the same time (Theorem 5.1). This is a serious obstacle because we do not know how to guarantee (4) for abstractions that are not simulation based. To the best of our knowledge, all abstractions used in timed automata verification algorithms are simulation based. The main hindrance to get finiteness is that the local times of processes can drift from each other by arbitrary amounts, but this quantity cannot be abstracted away.

Given this roadblock, we propose a restricted setting of *bounded-spread networks*. These are networks where the drift between processes can be controlled: every sequence of actions can be realized while maintaining a bounded drift between processes. For such networks, a suitable adaptation of the $\mathfrak{a}^{\star}_{\preccurlyeq LU}$ abstraction becomes finite, and has all the required four properties (Theorem 7.10).

The final step is to apply partial-order methods to the finite abstract local-zone graph with subsumptions. This is slightly delicate because the abstract local-zone graph with subsumptions does not have diamonds, precisely due to subsumptions (cf. Figure 5). Yet, we do not want to disallow subsumptions as they are essential to get an effective and an algorithmically efficient solution. We show that every partial-order method that works on graphs without subsumptions, intuitively for untimed systems, can be used for bounded-spread networks (Theorem 4.3). Abstractly, we see a partial-order method as computing a function *src* indicating for every state a subset of its outgoing transitions, such that exploring only this subset of transitions is sufficient to verify reachability. In our example from Figure 1b we may have $src(0,0) = \{b\}$ indicating that it is sufficient to explore only the transition b from the initial node. Since the $\mathfrak{a}^{\star}_{\preccurlyeq LU}$ abstraction is based on a simulation, we can show that if the *src* function is correct for the local-zone graph (without subsumptions), it is also correct to use it for the abstract local-zone graph with subsumptions, even though the latter does not have diamonds.

Putting these results together we obtain a methodology allowing to apply existing partial-order methods to timed-systems. The methodology is not general because it applies only to networks of bounded spread, while in general networks could have unbounded spread. Moreover, computing a spread of a given network is at least as difficult as testing reachability. On a positive side, we give examples of some types of networks that are guaranteed to have a bounded spread. We also propose a method to convert an arbitrary network into a bounded-spread network by introducing synchronizations between processes. We conclude with simple examples where our method brings exponential gains.

Related work. Local-time semantics has been considered by three groups. Bengtsson et al. in their article introducing local-time semantics [8] propose an algorithm for reachability checking. For this they introduce an abstraction called catch-up equivalence. It is rather improbable that an algorithm using this equivalence can be competitive against standard solutions because, as we show here, checking if two valuations are catch-up equivalent is PSPACE-hard. In [32] another equivalence is proposed, but it turns out to be not sound [23]. Sync-subsumption has been introduced in [23], but this subsumption does not preserve diamonds, hence it is not suitable for partial-order reduction.

An alternative to local-time zones was proposed by [31]. In that approach zones maintain a partial-order between clocks. For finiteness, an abstraction similar to sync-subsumption of [23] is used. Again, this does not preserve diamonds and hence the approach is not suitable for partial-order reduction on the control states.

Other works have proposed partial-order methods for timed automata, while keeping the standard semantics. For example, limiting partial-order methods only to parts where independent actions occur in zero-time [11, 30, 33]. Some works propose ways to discover which actions remain independent despite time constraints, either statically [16] or dynamically [24]. Two works [12, 13] apply unfolding techniques to bounded timed automata which admit a finite representation of their state space without abstraction.

Partial-order methods have been introduced in the 90s [22, 35, 37] as a method to speed up verification of transition systems. Later the accent shifted to program verification, and in particular to stateless model-checking [21]. The subject has become very active since the work of Abdulla et al. [1, 2] introducing a notion of optimal partial-order reduction (see [14, 28, 40] and references within). In this paper we take an abstract view of partial-order methods and do not focus on any concrete methods. The most recent works need some adaptation to be applicable in our setting. One reason is that they consider only straight-line processes, i.e., without branching. This is too restrictive in our setting.

Synopsis. In the Preliminaries, we introduce local-time semantics, local-zone graphs and their most important properties. We also present a succinct description of partial-order methods that is sufficient for this work. In Section 3 we introduce a notion of abstraction for local-zone graphs, and study conditions under which an abstraction can be used for reachability. In Section 4 we show how partial-order methods can be used in the presence of abstractions. Unfortunately, under mild assumptions, abstractions of local-zone graphs compatible with partial-order cannot be finite (Section 5). Our solution is to put a restriction on timed networks, but before this we develop in Section 6 an abstraction operator $\mathfrak{a}^{\star}_{\preccurlyeq LU}$, which is a generalization of the well-known $\mathfrak{a}_{\preccurlyeq LU}$ operator. We show that inclusion testing with respect to the new operator $\mathfrak{a}^{\star}_{\preccurlyeq LU}$ can be done efficiently. In Section 7 we introduce bounded-spread networks, give examples of such networks, as well as a general construction transforming a timed network into a bounded-spread network. We show that a modification of $\mathfrak{a}^{\star}_{\preccurlyeq LU}$ is finite on bounded-spread networks. We conclude with some examples where our method gives exponential gains and discuss where they come from.

2 PRELIMINARIES

In this section we introduce networks of timed automata, local-time semantics, and partial-order reduction. We present the standard global-time semantics as a special case of the local-time semantics. This clearly shows the differences between the two. Our approach allows to transfer any partial-order method from the untimed setting to the timed setting. In this paper, a partial-order method is given as an oracle that tells which transitions need to be explored. The only constraint is that the method keeps at least one execution from each trace equivalence class. At the end of the section, we introduce local-zone graphs, and state their properties.

We use $\mathbb{N}$ for the set of natural numbers, $\mathbb{R}$ for the set of reals and $\mathbb{R}_{\geq 0}$ for 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 $\sim \in \{<, \leq, =, \geq, >\}$. The base constraints $x \sim c$ will be called *atomic constraints.*

A *network of timed automata* is a collection of timed automata communicating with each other via shared actions. We have seen an example of a network with two automata in Figure 1a. Each automaton participating in the network is called a *process.* Formally, a timed network is a k-tuple of processes $\mathcal{N} = (A_1, \dots, A_k)$. Each process $A_p = \left(Q_p, \Sigma_p, X_p, q_p^{init}, T_p\right)$ has a finite set of states Q_p, a finite alphabet of actions Σ_p, a finite set of clocks X_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 $p_1 \neq p_2$. The sets of labels need not be disjoint - a label shared by two processes represents an action synchronizing the processes. The remaining components are an initial state q_p^{init} and a set of transitions $T_p \subseteq (Q_p \times \Sigma_p \times \phi(X_p) \times 2^{X_p} \times Q_p)$. A transition $(q, b, g, R, q') \in T_p$ has a label b, a *guard* g, and a set R of clocks to be *reset*. We write *Proc* for the set of all processes. We will use some abbreviations: $Q = \Pi_{p=1}^{k} Q_p$, $\Sigma = \bigcup_{p=1}^{k} \Sigma_p$ and $X = \bigcup_{p=1}^{k} X_p$. For a tuple of states $q \in Q$, we write $q(p)$ to be the state of process p in the tuple q. Every action b has its domain $dom(b) = \{p : b \in \Sigma_p\}$. The execution of action b requires participation of all processes in the domain. We denote by q^{init} the tuple of initial states q_p^{init} for each process p.

Local-time semantics. We introduce the local-time semantics of timed automata [8], and then the standard global-time semantics as a particular case. Fix a timed network $\mathcal{N}$.

In the local-time semantics, each process p has its local time represented by a clock t_p. The processes synchronize their times when doing a common action. The clock t_p, called the *reference clock* of process p, is never tested in a guard nor reset by the process. We will denote by X^t the set $\{t_p \mid p \in Proc\}$ of reference clocks. The other clock variables will store the local-time when the clock was last reset. Thus the value of t_p cannot be smaller than values of clocks of process p. More formally, a *local valuation* assigns a value, a real number, to each clock in $X \cup X^t$:

reset point semantics

$$v : (X \cup X^t) \to \mathbb{R} \qquad \text{provided } v(t_p) \geq v(x) \text{ for } x \in X_p$$

With this intuition, the difference $v(t_p) - v(x)$ gives the time since the last reset of clock x. This is what is considered as the value of x in the standard semantics. In the local-time semantics we allow negative values for $v(t_p), v(x)$ since we will always work with the difference $v(t_p) - v(x)$ and allowing for negative values offers some simplicity later while handling zones of local valuations. We use *LocalVal*$(X, Proc)$ for the set of local valuations, but mostly we will just write *LocalVal* as X and *Proc* will be clear from the context. In the sequel, we use the notation $v(x - y)$ for $v(x) - v(y)$.

Operations of clock reset for local valuations as well as local time elapse are defined accordingly, based on the interpretation given above. For a set of clocks R, let $v[R]$ denote the local valuation obtained by resetting R in v. That is: $v[R](x) = v(t_p)$ if $x \in R \cap X_p$ for some $p \in Proc$, and $v[R](x) = v(x)$ otherwise. For a tuple of non-negative reals $\Delta = \{\delta_p \in \mathbb{R}_{\geq 0}\}_{p \in Proc}$ we define $v \xrightarrow{\Delta} v'$ when $v'(t_p) = v(t_p) + \delta_p$ for all $p \in Proc$, and $v'(x) = v(x)$ for all $x \in X$. This denotes a *local delay* of Δ from the valuation v. The notion of a local valuation satisfying a guard is also adapted to this interpretation. For x a clock of process p, i.e. $x \in X_p$, we define $v \models x \sim c$ if $v(t_p - x) \sim c$ for $\sim \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{N}$.

Remark: One may wonder why not just keep the value of the clock in $v(x)$. This interpretation gives a big problem later when we consider zones of local valuations. It turns out that in this interpretation the set of valuations reachable by a transition from a zone may not be a zone. Quite remarkably the interpretation presented above avoids this problem [8].

A configuration of the network is a pair (q, v) where q is a tuple of control states of all processes, and v is a local valuation. An *initial valuation* v_0 associates the same real to each clock: $v_0(r - s) = 0$ for all $r, s \in X \cup X^t$. Let V_0 denote the set of initial local valuations. The initial configurations are $\{q^{init}\} \times V_0$. For an action b, the network $\mathcal{N}$ can execute a transition $(q, v) \xrightarrow{b} (q', v')$ if there is a tuple of b-transitions $\{(q_p, b, g_p, R_p, q'_p)\}_{p \in dom(b)}$ such that:

- states of involved processes change: $q_p = q(p)$, $q'_p = q'(p)$, if $p \in dom(b)$, and $q(p) = q'(p)$ if $p \notin dom(b)$;
- local times are synchronized: $v(t_{p_1}) = v(t_{p_2})$, for every $p_1, p_2 \in dom(b)$;
- guards are satisfied: $v \models g_p$, for every $p \in dom(b)$;
- resets are performed: $v' = v[\bigcup_{p \in dom(b)} R_p]$;

A *local run* of $\mathcal{N}$ is a sequence of local delay and action transitions from an initial configuration (q_0, v_0):

$$(q_0, v_0) \xrightarrow{\Delta_0} (q_0, v'_0) \xrightarrow{b_1} (q_1, v_1) \xrightarrow{\Delta_1} \cdots \xrightarrow{b_n} (q_n, v_n) \xrightarrow{\Delta_n} (q_n, v'_n)$$

We write $(q_0, v_0) \overset{u}{\dashrightarrow} (q_n, v'_n)$ to say that there is a sequence as above for $u = b_1 \dots b_n$ and adequate delays.

Global-time semantics and reachability. The standard semantics of a network, which we refer to as global-time semantics or just global semantics in short, is given by the semantics of the monolithic timed automaton obtained as the "synchronized product" of the individual processes. There is a common time for all processes: their reference clocks are always equal. In other words global semantics uses only *synchronized valuations* v where $v(t_p) = v(t_q)$ for all $p, q \in Proc$. In consequence, in the global semantics, we only allow *global delays* $v \xrightarrow{\Delta} v'$ which are local delays such that $\delta_p = \delta_q$ for any two processes $p, q \in Proc$. We use δ for global delays to distinguish from local delays Δ. A *global run* of $\mathcal{N}$ is an alternating sequence of global delay and action transitions starting from a synchronized valuation: $(q_0, v_0) \xrightarrow{\delta_0} (q_0, v'_0) \xrightarrow{b_1} (q_1, v_1) \xrightarrow{\delta_1} (q_1, v'_1) \xrightarrow{b_2} \cdots \xrightarrow{b_n} (q_n, v_n) \xrightarrow{\delta_n} (q_n, v'_n)$. Observe that all the valuations on a global run are synchronized.

The *reachability problem* asks if a state q_f is reachable in the global-time semantics. In other words, does there exist a global run from an initial configuration to a configuration (q_f, v) for some synchronized valuation v. This problem is known to be PSPACE-complete [3]. Most algorithms solving the reachability problem use the global semantics [6, 17, 25, 26]. The state q_f that we check for reachability is called the *final state* of the network $\mathcal{N}$ in the sequel.

Partial-order reduction (POR). We give a general outline of partial-order reductions that is sufficient for this work. The main idea is to use information about concurrency to avoid exploring equivalent interleavings of actions. In Figure 1, we have seen that the order of execution between b and c is irrelevant: starting from $(0, 0)$ both sequences end in $(1, 1)$. We say that b and c are *independent* in a transition system $\mathcal{S}$ if this property holds for every state of $\mathcal{S}$. The notion of independence leads to trace equivalence on sequences: two sequences are trace equivalent, denoted $u \sim_{\mathcal{S}} v$, if one can be obtained from the other by permuting adjacent independent actions. This is an equivalence relation on sequences of actions. Moreover, if $u \sim_{\mathcal{S}} v$ and u leads from an initial to a final state in $\mathcal{S}$ then so does v. A POR method aims at exploring at least one path from every trace-equivalence class, but preferably not much more.

For instance in Figure 1 we may only explore the sequence $bc\$$, and ignore the sequence $cb\$$. This avoids visiting state $(0, 1)$. In some cases this optimization may lead to exponential reductions in the number of visited states.

We think of a POR method as a way of computing for a given transition system $\mathcal{S}$ a source function, $src : Q \to \mathcal{P}(\Sigma)$ assigning to every state of $\mathcal{S}$ a set of relevant actions. A path in $\mathcal{S}$ is a *source path* if it is a path in the restriction of $\mathcal{S}$ where from every state q we eliminate transitions on actions that are not in $src(q)$. A source function should be *trace-faithful* meaning that for every state q and every path $q \to uq_f$ to a final state q_f, there must be a trace-equivalent sequence $v \sim_{\mathcal{S}} u$ such that $q \to vq_f$ is a source path. In the example from Figure 1 we may take $src(0, 0) = src(0, 1) = \{b\}$, $src(1, 0) = \{c\}$ and $src(1, 1) = \{\$\}$. The goal is to find a trace-faithful source function without exploring the transition system.

A common way to get a src function is to look at the parallelism in a given system. In a network of automata without timing constraints, two actions with disjoint domains are independent in the sense of the previous paragraph. Stubborn sets [37], ample sets [35], persistent sets [22], faithful decompositions [27], stamper sets [36], source sets [2], are different ways of computing a source function in this setting.

Our goal in this work is to develop a theory allowing to use the same approach for networks of timed automata. As seen in Figure 1c, in timed networks, two domain-disjoint actions may not be independent. Global time destroys diamonds, making it difficult to find out which actions are independent. Local-time semantics allows to recover diamonds, cf. Figure 2. Reachability can be solved using local-time, as we see next.

Reachability and diamonds in local-time. Observe that every global run is a local-time run. Conversely, for every local-time run there is a trace equivalent global run.

Lemma 2.1. *[23] Let v, v' be synchronized local valuations, and let $(q, v) \overset{u}{\dashrightarrow} (q', v')$ be a local run. Then there exists a global run $(q, v) \overset{w}{\dashrightarrow} (q', v')$ such that $u \sim w$.*

Since the initial valuations are all synchronized, the above lemma ensures that a control state q is reachable in the local-time semantics iff it is reachable in the global-time semantics. This is particularly true of the final state q_f. Given this correspondence, we will henceforth work completely with the local-time semantics. Additionally, the local-time semantics offers the diamond property which is essential for POR.

Lemma 2.2 (Diamond property). *Suppose $dom(a) \cap dom(b) = \emptyset$. If $(q, v) \overset{ab}{\dashrightarrow} (q', v')$ then $(q, v) \overset{ba}{\dashrightarrow} (q', v')$.*

Local-zone graphs. To make the local-time semantics feasible for use in algorithms, a notion of local-zones, analagous to the zones in the global-time setting [17], is employed. A *local-zone* is a set of local valuations given by conjunctions of constraints: $x - y \lessdot c$ where $x, y \in X \cup X^t$, $\lessdot \in \{<, \leq\}$ and $c \in \mathbb{Z}$. For a set of local valuations W, define:

- local-elapse$(W) := \left\{v + \Delta \mid v \in W,\ \Delta \in \mathbb{R}^k_{\geq 0}\right\}$,
- $W[R] := \{v[R] \mid v \in W\}$, for a set of clocks $R \subseteq X$.
- $W \cap g := \{v \mid v \models g\}$ for a guard g.

It can be shown that for a local-zone Z, the sets local-elapse(Z) (local-time delay), $Z[R]$ (clock reset) and $Z \cap g$ (intersection with guard) are local-zones [8, 23].

Local-zones can be implemented using Difference Bound Matrices (DBMs), similar to the case of standard zones. Hence, they can be computed and stored as efficiently as standard zones. Before defining the local-zone graph, we lift the local semantics from configurations to sets of configurations.

Definition 2.3 (Symbolic transition relation). Let W be a set of local valuations. We write $(q, W) \overset{b}{\Rightarrow} (q', W')$ if there exists a tuple of b-transitions $\{(q_p, b, g_p, R_p, q'_p)\}_{p \in dom(b)}$ such that

- $q(p) = q_p$ and $q'(p) = q'_p$ for all $p \in dom(b)$, and $q(p) = q'(p)$ for all $p \notin dom(b)$;
- $W' =$ local-elapse(W_2) is not empty, where W_2 is defined as: $W_2 = W_1[\bigcup_{p \in dom(b)} R_p]$ and $W_1 = W \cap (\bigwedge_{p \in dom(b)} g_p \wedge \bigwedge\{t_p = t_q \mid p, q \in dom(b)\})$

We write $(q, W) \overset{b_1 \ldots b_n}{\Rightarrow} (q_n, W_n)$ if there is a sequence of symbolic transitions $(q, W) \overset{b_1}{\Rightarrow} (q_1, W_1) \cdots \overset{b_n}{\Rightarrow} (q_n, W_n)$.

The following lemma states the relation between transitions on zones and on valuations. Its proof follows from the definition of symbolic transitions. We say that a local-zone Z is *time-elapsed* if $Z =$ local-elapse(Z).

Lemma 2.4 (Pre and post properties). *For every network of timed automata and every action b:*

pre-property: *If $(q, v) \overset{b}{\dashrightarrow} (q', v')$ and $v \in Z$ for some time-elapsed local-zone Z then $(q, Z) \overset{b}{\Rightarrow} (q', Z')$ and $v' \in Z'$ for some local-zone Z'.*

post-property: *If $(q, Z) \overset{b}{\Rightarrow} (q', Z')$ and $v' \in Z'$ for local-zones Z, Z', then $(q, v) \overset{b}{\dashrightarrow} (q', v')$ for some $v \in Z$.*

Definition 2.5 (Local-zone graph LZG$(\mathcal{N})$*).* The local-zone graph LZG$(\mathcal{N})$ of a network $\mathcal{N}$ is a transition system whose nodes are of the form (q, Z) where q is a state of the network, and Z is a local-zone. The initial node is (q_0, Z_0) with $Z_0 =$ local-elapse(V_0) where V_0 is the set of initial valuations and $q_0 = q^{init}$. The transitions are given by the symbolic transition relation $(q, Z) \overset{b}{\Rightarrow} (q', Z')$.

The initial zone is time-elapsed. This entails that every zone reachable by $\Rightarrow$ transitions is also time-elapsed, due to Definition 2.3. Using this observation along with the pre- and post-properties of Lemma 2.4, we get the following theorem.

Theorem 2.6. *[8, 23] For a given network $\mathcal{N}$, there is a run of $\mathcal{N}$ reaching a state q iff there is a path in* LZG$(\mathcal{N})$ *from the initial node to a node (q, Z).*

This theorem suggests that the local-zone graph LZG$(\mathcal{N})$ could potentially be used to analyze reachability. The local-zone graph is an untimed transition system and we are interested in applying partial-order methods on it. As desired, domain-disjoint actions are independent in the local-zone graph. This is a consequence of Lemmas 2.2 and 2.4.

Proposition 2.7 (Diamond property of LZG$(\mathcal{N})$). *Assume $dom(a) \cap dom(b) = \emptyset$. If $(q, Z) \overset{ab}{\Rightarrow} (q', Z')$ then $(q, Z) \overset{ba}{\Rightarrow} (q', Z')$.*

Let us remark that the so called *enabledness* property [15] may not hold in a local-zone graph: it is possible to construct a network, a local-zone Z and two domain-disjoint actions a, b such that from (q, Z) there are both $\overset{a}{\Rightarrow}$ and $\overset{b}{\Rightarrow}$ transitions but neither $\overset{ab}{\Rightarrow}$ nor $\overset{ba}{\Rightarrow}$ are feasible from (q, Z) [32]. This happens when every valuation in Z can do either $\overset{a}{\Rightarrow}$ or $\overset{b}{\Rightarrow}$, but not both. Enabledness is however true at the level of configurations. Indeed if both $\xrightarrow{a}$ and $\xrightarrow{b}$ are feasible from (q, v), then both sequences $\overset{ab}{\dashrightarrow}$ and $\overset{ba}{\dashrightarrow}$ exist since domain-disjoint actions a and b involve distinct clocks.

Although the local-zone graph is sound and complete for reachability, and has the diamond property, it may be infinite. Hence a finite abstraction of the local-zone graph is required for analysis. This is the subject for the next section.

3 ABSTRACT LOCAL-ZONE GRAPHS

The goal of this section is to study finite abstractions of local-zone graphs that can be used to answer the reachability question. We introduce a general definition of an abstraction and of an abstract local-zone graph. Then we put restrictions on abstractions that make the abstract local-zone graph sound and complete for reachability.

We fix a timed network $\mathcal{N}$. This allows us to omit indexing every notion with $\mathcal{N}$.

Definition 3.1. A *quasi-abstraction operator* $\mathfrak{a} : \mathcal{P}(\mathit{LocalVal}) \to \mathcal{P}(\mathit{LocalVal})$ is a function from sets of local valuations to sets of local valuations such that $\mathfrak{a}(\mathfrak{a}(W)) = \mathfrak{a}(W)$ for all sets of local valuations W. If the operator additionally satisfies $W \subseteq \mathfrak{a}(W)$ for all sets W, we call it an *abstraction operator*.

The definition of the abstraction operator is the same as in the global-time semantics [6, 26], except that now we work with local valuations. We will use the weaker notion which we have called a quasi-abstraction to get finite abstractions in our setting.

A quasi-abstraction operator allows to compute an abstract local-zone graph. An exploration of a local-zone graph is stopped when a node with a bigger abstraction is already in the graph. The smaller node is said to be *subsumed* by the bigger node. If the quasi-abstraction is finite as defined below, then we can have a finite abstract graph.

Definition 3.2. A quasi-abstraction $\mathfrak{a}$ is said to be finite if for every infinite sequence of zones $Z_1, Z_2, \ldots$ there exist $i < j$ such that $\mathfrak{a}(Z_i) \supseteq \mathfrak{a}(Z_j)$.

Definition 3.3 ($\mathrm{LZG}^{\mathfrak{a}}(\mathcal{N})$*).* Suppose $\mathfrak{a}$ is a quasi-abstraction operator. An *abstract local-zone graph* is a subset of nodes and edges of $\mathrm{LZG}(\mathcal{N})$ together with some new edges called *subsumption edges*. Each node is labeled either *covered* or *uncovered*. The graph must satisfy the following conditions:

- The initial node of $\mathrm{LZG}(\mathcal{N})$ belongs to the graph.
- For every uncovered node (q, Z), all its successors together with associated transitions $(q, Z) \overset{b}{\Rightarrow} (q', Z')$ in $\mathrm{LZG}(\mathcal{N})$ should be in the graph.
- For every covered node (q, Z) there is an uncovered node (q, Z') with $\mathfrak{a}(Z) \subseteq \mathfrak{a}(Z')$; moreover there is an explicit subsumption edge $(q, Z) \rightsquigarrow (q, Z')$.
- Every node of the graph must be reachable from the initial node by a path of $\Rightarrow$ edges.

We denote by $\mathrm{LZG}^{\mathfrak{a}}(\mathcal{N})$ some abstract zone graph for $\mathcal{N}$. One can imagine that we take the first one in some fixed order on graphs.

A point worth noting is that the algorithm stores zones and not abstract sets. Indeed, we do not assume that an abstraction of a zone is a zone, and therefore we do not know a priori how to store and manipulate abstract sets directly.

The question now is when it is correct to examine the abstract local-zone graph instead of the network itself: when can we say that a given state is reachable by a run in a network iff it is reachable in its abstract local-zone graph. Since every node of $\mathrm{LZG}^{\mathfrak{a}}(\mathcal{N})$ is reachable by a sequence of $\Rightarrow$ transitions, we have:

Lemma 3.4. *Every abstract local-zone graph is sound: if a final state is reachable in* $\mathrm{LZG}^{\mathfrak{a}}(\mathcal{N})$ *then it is reachable in* $\mathcal{N}$.

We now study the converse implication.

Definition 3.5. A quasi-abstraction operator $\mathfrak{a}$ is *complete* when reachability of a state q in $\mathcal{N}$ implies its reachability in $\mathrm{LZG}^{\mathfrak{a}}(\mathcal{N})$.

The challenge is to get complete and finite quasi-abstraction operators for which the test $\mathfrak{a}(Z) \subseteq \mathfrak{a}(Z')$ is efficient. Abstractions for the global semantics are based on simulation relations [6, 17, 26]. Our next step is to consider abstractions based on simulations for the local semantics.

Definition 3.6. A *(time-abstract) simulation* relation $\preccurlyeq$ on the local semantics is a reflexive and transitive relation $(q, v) \preccurlyeq (q, v')$ between configurations having the same discrete state that satisfies two conditions:

(1) for every local delay transition $(q, v) \xrightarrow{\Delta} (q, v_1)$, there exists a local delay Δ' such that $(q, v') \xrightarrow{\Delta'} (q, v'_1)$ and $(q, v_1) \preccurlyeq (q, v'_1)$,

(2) for every transition $(q, v) \xrightarrow{b} (q_1, v_1)$ there is a transition $(q, v') \xrightarrow{b} (q_1, v'_1)$ with $(q_1, v_1) \preccurlyeq (q_1, v'_1)$.

We say $v \preccurlyeq v'$ if $(q, v) \preccurlyeq (q, v')$ for all states q. When $\Delta' = \Delta$ in the first condition above, the relation is called a *strong-timed simulation.*

Definition 3.7. A quasi-abstraction operator $\mathfrak{a}$ is *simulation based* if there is a simulation $\preccurlyeq$ such that $\mathfrak{a}(W) \subseteq \{v : \exists v' \in W.\ v \preccurlyeq v'\}$.

In particular, there is the biggest abstraction operator based on a simulation $\preccurlyeq$. It is simply the downward closure operator with respect to $\preccurlyeq$.

Lemma 3.8. *A simulation based abstraction operator is complete.*

This lemma is not true in general for quasi-abstractions. The proof of the lemma crucially uses $Z \subseteq \mathfrak{a}(Z)$, a property which may not hold in a quasi-abstraction. We propose an additional condition for quasi-abstractions that requires the abstraction $\mathfrak{a}(Z)$ to keep some of the "good" valuations from the local-zone Z.

Definition 3.9. A quasi-abstraction $\mathfrak{a}$ *keeps runs* if for every node (q, Z) in $\mathrm{LZG}(\mathcal{N})$ that is reachable from the initial node, and every path $(q, Z) \overset{u}{\Rightarrow} (q_f, Z_f)$ to the final state q_f there is a valuation $v \in \mathfrak{a}(Z)$ and a run $(q, v) \overset{u}{\dashrightarrow} (q_f, v_f)$.

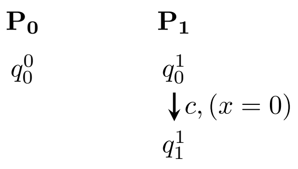


Figure 3: Warm-up example.

This property means that $\mathfrak{a}$ should keep all paths leading to a final state. Observe that every abstraction operator keeps runs since $Z \subseteq \mathfrak{a}(Z)$. The property of keeping runs, along with the operator being simulation based, gives a complete quasi-abstraction.

Lemma 3.10. *A simulation based quasi-abstraction operator that keeps runs is complete for reachability.*

Proof. We show a more general result in Theorem 4.3. This lemma follows by taking *src* in Theorem 4.3 to be the set of all enabled actions for every (q, Z). □

The aim now is to come up with a concrete quasi-abstraction $\mathfrak{a}$ that satisfies the properties of the above lemma and for which the test $\mathfrak{a}(Z) \subseteq \mathfrak{a}(Z')$ is efficient. We make a short digression into one of the first quasi-abstractions proposed for the local-time semantics.

Catch-up equivalence. A quasi-abstraction operator based on a relation between configurations called *catch-up equivalence* has been defined in [8]. However, as we show below, deciding whether two configurations are catch-up equivalent is Pspace-hard.

We start with a definition of the equivalence. A delay $(q, v) \xrightarrow{\Delta} (q, v')$ is a *catch-up delay* if $\max(\{v'(t)\}_{t \in T}) = \max(\{v(t)\}_{t \in T})$. So catch-up delays only allow the processes that are behind in time to join the most advanced processes. Two local-time configurations (q, v) and (q', v') are *catch-up equivalent* if the two can reach the same synchronized regions (i.e. Alur & Dill's regions [3]) through catch-up delays and discrete transitions.

Proposition 3.11. *The problem of deciding if two given configurations (q, v), (q', v') of a given timed network are catch-up equivalent is* Pspace*-hard.*

Proof. To warm-up we consider the simple network in Figure 3. Process P_0 has no transitions, and process P_1 has one transition guarded with $x = 0$. Recall that the guard $(x = 0)$ is enabled in a valuation v if $v(t_1 - x) = 0$. We claim that the following configurations are not catch-up equivalent:

$$(q_0^0, q_0^1, [t_0 = t_1 = 1, x = 0]) \nsim (q_0^0, q_0^1, [t_0 = 1, t_1 = x = 0])$$

Indeed, in the first configuration no transition or catch-up delay is possible. However, in the second configuration, process P_1 can do transition c immediately, and then delay to catch-up P_0, eventually reaching $(q_0^0, q_1^1, [t_0 = t_1 = 1, x = 0])$. On the other hand if there is no c transition then the two configurations are equivalent, because the only thing possible is that process P_1 lets the time pass to catch-up with process P_0.

We use the same idea to reduce the language emptiness of the intersection of n finite automata.

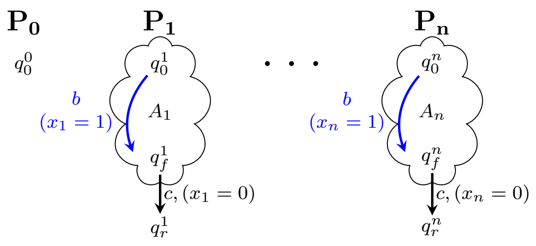


Figure 4: Reduction from intersection of finite automata.

We are given finite automata $A_1, \ldots, A_n$ with initial states q_0^i and final states q_f^i. From each automaton A_i we build a timed process P_i as depicted in Figure 4. Process P_i has the same states as A_i and one extra state q_r^i. Each transition of A_i exists in P_i but with guard $(x_i = 0)$. Moreover on a new letter b we add transitions from the initial state of automaton P_i to every state of P_i with guard $(x_i = 1)$. Finally, from the final state q_f^i we add a transition c with the guard $(x_i = 0)$ to the new state q_r^i. The transitions b and c synchronize processes $P_1, \ldots, P_n$. In particular, observe that transition c is only enabled if all processes are in their final states, that is if the all automata $A_1, \ldots, A_n$ can reach their final states. As in our warm-up example, we add an extra process P_0 with initial state q_0^0 and no transition. We claim that $L(A_1) \cap \cdots \cap L(A_n) \neq \emptyset$ iff the following two configurations are not equivalent:

$$conf_{1,1} = (q_0^0, q_0^1, \ldots, q_0^n, [t_0 = t_1 = \cdots = t_n = 1, x_1 = \cdots = x_n = 0])$$
$$conf_{1,0} = (q_0^0, q_0^1, \ldots, q_0^n, [t_0 = 1, t_1 = x_1 = \cdots = t_n = x_n = 0])$$

If intersection is empty then transition c cannot be taken. The synchronized configurations reachable from $conf_{1,1}$ are all combinations of states of $P_1, \ldots, P_n$ thanks to the added b transitions (recall that the guard $(x_i = 1)$ is enabled in v if $v(t_i - x_i) = 1$). Similarly, from $conf_{1,0}$, synchronized configurations are reached either by first doing a catch-up delay of 1 unit in all processes $P_1, \ldots, P_n$ leading to $conf_{1,1}$ and then performing the b actions, or by taking the internal actions in 0-time and then delaying 1 unit to catch-up with P_0. Either way, the set of synchronized configurations reached is the same as that of $conf_{1,1}$.

If the intersection is non-empty then from $conf_{1,0}$ processes can get to $q_f^1, \ldots, q_f^n$ in 0-time, and then do transition c followed by a delay, reaching $(q_0^0, q_r^1, \ldots, q_r^n, [t_0 = t_1 \cdots = t_n = 1, x_1 = \cdots = x_n = 0])$. This synchronized state cannot be reach from $conf_{1,1}$. □

To summarize this section, we have seen the properties we need of a quasi-abstraction to get a correct abstract local-zone graph (Lemma 3.10). In Section 6 we will present an efficient simulation based abstraction for local-zone graphs. Before that, we talk about partial-order reduction.

4 POR ON ABSTRACT LOCAL-ZONE GRAPHS

We discuss how to use partial-order methods on abstract zone graphs. At this point, we have a local-zone graph of a network LZG($\mathcal{N}$) that has diamonds but may be infinite. We suppose that we have some quasi-abstraction $\mathfrak{a}$ giving a finite abstract local-zone graph LZG$^{\mathfrak{a}}$($\mathcal{N}$). We would like to use partial-order methods on

$$\begin{tikzcd}
 & (q, Z) \arrow[dl, Rightarrow, "a"'] \arrow[dr, Rightarrow, "b"] & & \\
(q_a, Z_a) \arrow[dr, Rightarrow, "b"'] & & (q_b, Z_b) \arrow[r, rightsquigarrow] & (q_b, Z'_b) \arrow[d, Rightarrow, "a"] \\
 & (q_{ab}, Z_{ab}) & & (q_{ab}, Z'_{ab})
\end{tikzcd}$$

Figure 5: A diamond that is destroyed by a subsumption.

$\mathrm{LZG}^{\mathfrak{a}}(\mathcal{N})$, but this graph may not have diamonds as we illustrate in Figure 5. Due to subsumption there are no transitions from (q_b, Z_b). So, $\mathrm{LZG}(\mathcal{N})$ has diamonds but may be infinite, and $\mathrm{LZG}^{\mathfrak{a}}(\mathcal{N})$ is finite but has no diamonds. We show that when $\mathfrak{a}$ satisfies the conditions given by Lemma 3.10, every partial-order method for $\mathrm{LZG}(\mathcal{N})$ can be used on $\mathrm{LZG}^{\mathfrak{a}}(\mathcal{N})$ even if it has no diamonds.

In this section we will assume that we have a source function *src* for $\mathrm{LZG}(\mathcal{N})$ given by a partial-order method as described in Section 2. In $\mathrm{LZG}(\mathcal{N})$, we have diamonds and so we can use any partial-order method to calculate a source function. Recall that the nodes of $\mathrm{LZG}(\mathcal{N})$ are pairs (q, Z). The graph $\mathrm{LZG}(\mathcal{N})$ may be infinite since there are infinitely many local-zones. As we want the source function to be given by some finite description, we assume that it does not depend on the local-zone, and instead depends only on the state q and the set of actions enabled from (q, Z), denoted as $\mathrm{enabled}(q, Z)$.

Definition 4.1. A *source function* for a timed network $\mathcal{N}$ is a function $src : Q \times \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$. A source function of $\mathcal{N}$ is trace-faithful if for every node (q, Z) and a path u from (q, Z) to a final state there is a source path $w \sim u$ from (q, Z).

The concept of trace-faithful source function is directly inspired by partial-order methods. Indeed, partial-order techniques always compute trace-faithful source functions as they guarantee that every path has at least one equivalent source path.

Remark. Partial-order methods in general require both the diamond and enabledness properties [15]. In our case $\mathrm{LZG}(\mathcal{N})$ has diamonds, but not necessarily the enabledness property. The latter property is not needed if, for example, final states are reached by a global synchronization action, or final states are determined by a state of one of the processes. The definition above of the source function hides this problem. When applying some existing partial-order methods, some precaution, or transformation of a system, should be done to ensure that the source function is indeed trace-faithful.

We can now combine abstraction and partial-order reduction.

Definition 4.2. For a timed network $\mathcal{N}$ and a source function $src : Q \times \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$, the graph $\mathrm{LZG}^{src}(\mathcal{N})$ is obtained from $\mathrm{LZG}(\mathcal{N})$ by keeping only the edges allowed by the *src* function: $(q, Z) \overset{b}{\Rightarrow} (q', Z')$ such that $b \in src(q, \mathrm{enabled}(q, Z))$.

Then, $\mathrm{LZG}^{\mathfrak{a}, src}(\mathcal{N})$ is a graph obtained from $\mathrm{LZG}^{src}(\mathcal{N})$ that satisfies the conditions in Definition 3.3.

We now have a graph $\mathrm{LZG}^{\mathfrak{a}, src}(\mathcal{N})$ on which both subsumption and POR have been applied. Used separately, both of them yield transition systems that are sound and complete for reachability. The next theorem says that even the combination is correct.

Theorem 4.3. *If src is a trace-faithful source function and $\mathfrak{a}$ is a simulation based quasi-abstraction that keeps runs, then a final state is reachable in* $\mathrm{LZG}(\mathcal{N})$ *iff it is reachable in* $\mathrm{LZG}^{\mathfrak{a}, src}(\mathcal{N})$.

Proof. If a final state is reachable in $\mathrm{LZG}^{\mathfrak{a}, src}(\mathcal{N})$ then it is reachable by a sequence of $\Rightarrow$ transitions by definition (c.f. Definitions 4.2, 3.3). This gives a path in $\mathrm{LZG}(\mathcal{N})$.

Consider left-to-right direction. Since *src* is trace-faithful, it is sufficient to show that each source path in $\mathrm{LZG}(\mathcal{N})$ that leads to a final state has a representative source path in $\mathrm{LZG}^{\mathfrak{a}, src}(\mathcal{N})$, potentially with subsumption edges. Suppose w_0 is a source path in $\mathrm{LZG}(\mathcal{N})$ from (q_0, Z_0) to (q_n, Z_n), with q_n an accepting state. Let $n = |w_0|$ be the length of w_0. By induction on i we show that there are paths u_i, w_i such that:

- $(q_0, Z_0) \overset{u_i}{\Rightarrow} (q_i, Z_i)$ is a path in $\mathrm{LZG}^{\mathfrak{a}, src}(\mathcal{N})$,
- $(q_i, Z_i) \overset{w_i}{\Rightarrow} (q_n, Z_n^i)$ is a source path in $\mathrm{LZG}(\mathcal{N})$,
- $|w_i| = n - i$

The initial step is trivial. The induction step is easy if (q_i, Z_i) is uncovered in $\mathrm{LZG}^{\mathfrak{a}, src}(\mathcal{N})$. In this case, let b be the first letter of w_i: $w_i = bw_{i+1}$. We have $(q_i, Z_i) \overset{b}{\Rightarrow} (q_{i+1}, Z_{i+1})$ and $b \in src(q_i, \mathrm{enabled}(Z_i))$. Hence taking $u_{i+1} = u_i b$ and w_{i+1} we obtain the induction step.

It remains to check what happens when (q_i, Z_i) is covered in $\mathrm{LZG}^{\mathfrak{a}, src}(\mathcal{N})$. Say (q_i, Z_i) is subsumed by (q_i, Z'_i), meaning $\mathfrak{a}(Z_i) \subseteq \mathfrak{a}(Z'_i)$. Since $(q_i, Z_i) \overset{w_i}{\Rightarrow} (q_n, Z_n^i)$ and $\mathfrak{a}$ keeps runs, there exists a valuation $v_i \in \mathfrak{a}(Z_i)$ and an execution $(q_i, v_i) \longrightarrow w_i(q_n, v_n)$. From $\mathfrak{a}(Z_i) \subseteq \mathfrak{a}(Z'_i)$, we have $v_i \in \mathfrak{a}(Z'_i)$. Secondly, as $\mathfrak{a}$ is simulation based, there exists $v'_i \in Z'_i$ such that $v_i \preccurlyeq v'_i$, where $\preccurlyeq$ is the simulation on which $\mathfrak{a}$ is based on. Hence we also have an execution $(q_i, v'_i) \longrightarrow w_i(q_n, v'_n)$. By pre-property of zones $(q_i, Z'_i) \Rightarrow w_i(q_n, Z'_n)$ in $\mathrm{LZG}(\mathcal{N})$. Since w_i is a path reaching a final state, and *src* function is trace faithful, there is a source path $(q_i, Z'_i) \Rightarrow w'_i(q_n, Z'_n)$ in $\mathrm{LZG}(\mathcal{N})$ with $w_i \sim w'_i$. In particular, $|w_i| = |w'_i|$. Let b be the first letter of w'_i, i.e., $w'_i = bw_{i+1}$. We claim that $u_{i+1} = u_i b$ and w_{i+1} satisfy the induction conditions. The path $u_i b$ contains a subsumption edge. □

5 NO FINITE ABSTRACTIONS FOR LOCAL-ZONE GRAPHS

Theorem 4.3 gives a sufficient condition for a quasi-abstraction to be compatible with POR. There is one ingredient missing to get an algorithm. We need a finite quasi-abstraction. Unfortunately, we show that this is impossible under the assumptions made on the quasi-abstraction in Theorem 4.3. In the argument below, we do not really need that the quasi-abstraction keeps all runs. It would be enough to keep for every path a run with the same Parikh image.

Theorem 5.1. *There is a network $\mathcal{N}^-$ such that* $\mathrm{LZG}^{\mathfrak{a}}(\mathcal{N}^-)$ *is infinite for every simulation based quasi-abstraction operator that keeps runs.*

Proof. We present a network $\mathcal{N}^-$ such that $\mathrm{LZG}^{\mathfrak{a}}(\mathcal{N}^-)$ is infinite for every simulation based quasi-abstraction $\mathfrak{a}$ that keeps runs. The same example appears in Lugiez et al. [31] in a similar context. The network $\mathcal{N}^-$, presented in Figure 6, consists of two processes A_1 and A_2. It is easy to see that any accepting run of the network executes an equal number of b's and c's followed by the global synchronizing action \$.

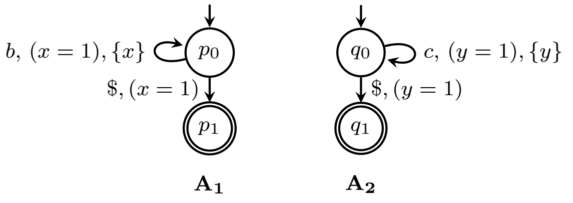


Figure 6: A network of two processes without a finite abstract zone graph that contains all runs.

Consider LZG($\mathcal{N}^-$), the local-zone graph of $\mathcal{N}^-$. For every $m, n \geq 0$, the network has a run on $b^m c^n$. Let $(p_0, q_0, Z_{m,n})$ be the node in LZG($\mathcal{N}^-$) reached from the initial node after the sequence $b^m c^n$: $(p_0, q_0, Z_0) \Rightarrow b^m c^n (p_0, q_0, Z_{m,n})$.

Pick $i > j \geq 0$. We claim that:

- $\mathfrak{a}(Z_{i,j}) \not\subseteq \mathfrak{a}(Z_{k,l})$ for any $k, l \geq 0$ with $(i - j) \neq (k - l)$.

Suppose to the contrary that $\mathfrak{a}(Z_{i,j}) \subseteq \mathfrak{a}(Z_{k,l})$ for some i, j, k, l with $(i-j) \neq (k-l)$. Consider an execution $(p_0, q_0, v_0) \overset{b^i c^j}{\dashrightarrow} (p_0, q_0, v_{i,j}) \overset{c^{i-j}\$}{\dashrightarrow} (p_1, q_1, v)$ of $\mathcal{N}^-$. We have $v_{i,j} \in Z_{i,j}$. Hence by pre-property from Lemma 2.4, there is a path $(p_0, q_0, Z_{i,j}) \Rightarrow c^{i-j}(p_1, q_1, Z)$ in LZG($\mathcal{N}^-$). Now, as the operator $\mathfrak{a}$ keeps runs, there is $v'_{i,j} \in \mathfrak{a}(Z_{i,j})$ and a run $(p_0, q_0, v'_{i,j}) \overset{c^{i-j}}{\dashrightarrow} (p_1, q_1, v')$. By $\mathfrak{a}(Z_{i,j}) \subseteq \mathfrak{a}(Z_{k,l})$ we have $v'_{i,j} \in \mathfrak{a}(Z_{k,l})$. Since $\mathfrak{a}$ is simulation based there is $v_{k,l} \in Z_{k,l}$ with configuration $(p_0, q_0, v_{k,l})$ simulating $(p_0, q_0, v'_{i,j})$. Hence, we have $(p_0, q_0, v_{k,l}) \overset{c^{i-j}\$}{\dashrightarrow} (p_1, q_1, u)$ in $\mathcal{N}^-$. From the fact that $v_{k,l} \in Z_{k,l}$ and the post-property (Lemma 2.4), there is an execution $(p_0, q_0, v_0) \overset{b^k c^l}{\dashrightarrow} (p_0, q_0, v_{k,l})$. Combining the last two executions we obtain: $(p_0, q_0, v_0) \overset{b^k c^l}{\dashrightarrow} (p_0, q_0, v_{k,l}) \overset{c^{i-j}\$}{\dashrightarrow} (p_1, q_1, u)$. This is impossible for $(i - j) \neq (k - l)$.

By the diamond property of LZG($\mathcal{N}^-$), any sequence containing k occurrences of b, and l occurrences of c ends in $(p_0, q_0, Z_{k,l})$. From $\mathfrak{a}(Z_{i,j}) \not\subseteq \mathfrak{a}(Z_{k,l})$, the node $(p_0, q_0, Z_{i,j})$ (reached by any sequence containing i occurrences of b and j occurrences of c) cannot be subsumed by any other node. This shows there is infinitely many nodes in LZG$^{\mathfrak{a}}(\mathcal{N})$: at least one for every difference $(i - j)$. □

In [23], a simulation based quasi-abstraction is defined which is shown to be finite and complete. This operator however does not keep runs, which is in accordance with the above result. Due to this reason, this operator is not amenable for partial-order reduction.

In the following sections we propose a way out from the apparent deadlock created by Theorems 4.3 and 5.1. One direction could be to find an abstraction operator not satisfying the hypothesis of Theorem 5.1, that is, either not simulation based or not keeping runs. We do not know how to do this while still preserving some form of Theorem 4.3. Our solution is to put some restrictions on the timed networks we consider. We will first generalize the $\mathfrak{a}_{\preccurlyeq LU}$ abstraction [6] for global-time semantics to the local-time semantics. Then we will show sufficient conditions under which it is finite.

6 *LU*-ABSTRACTION FOR THE LOCAL SEMANTICS

We will present a concrete strong-timed simulation for local-time semantics. It generalizes the LU-simulation for the global-time semantics from [6]. The simulation is parameterized by two functions L and U that keep for each clock the maximum constant among lower bound constraints $x \geq c, x > c$ and upper bound constraints $x \leq c, x < c$ respectively. The simulation induces an abstraction operator $\mathfrak{a}^{\star}_{\preccurlyeq LU}$ which is sound, complete and keeps runs for networks with bounds L and U. The impossibility result from the previous section still applies though. Indeed the operator is not finite. In Section 7, we will present a restriction on timed networks and modify the abstraction operator to a quasi-abstraction operator that will be finite for the restricted class of networks.

Definition 6.1. An LU*-bound* is a pair of functions $L : X \to \mathbb{N} \cup \{-\infty\}$ and $U : X \to \mathbb{N} \cup \{-\infty\}$, each of which maps process clocks to a natural number or $-\infty$. An atomic constraint $x \sim c$ is an LU*-constraint* if $c \leq L(x)$ when $\sim \in \{\geq, >\}$ (lower bound constraint) and if $c \leq U(x)$ when $\sim \in \{<, \leq\}$ (upper bound constraint). A network $\mathcal{N}$ is an LU*-network* if every guard in $\mathcal{N}$ is a conjunction of LU-constraints.

We next lift the LU-preorder [6], written as $\preccurlyeq_{LU}$ and defined for the global-time semantics to the local-time setting. Here, when we relate v and v', we require that the difference between reference clocks is the same for both v and v'.

Definition 6.2 ($\preccurlyeq^{\star}_{LU}$-preorder). Given LU-bounds L and U. For two local valuations v, v', we say $v \preccurlyeq^{\star}_{LU} v'$ if:

- $v(t_p - t_q) = v'(t_p - t_q)$ for all $p, q \in Proc$
- for all $p \in Proc$ and all $x \in X_p$
  - $v(t_p - x) \leq U_x \implies v'(t_p - x) \leq v(t_p - x)$
  - $v(t_p - x) \leq L_x \implies v'(t_p - x) \geq v(t_p - x)$
  - $v(t_p - x) > L_x \implies v'(t_p - x) > L_x$

Intuitively, the relation $v \preccurlyeq^{\star}_{LU} v'$ ensures the following: (1) whenever $v + \Delta$ synchronizes t_p and t_q, $v' + \Delta$ also synchronizes them, (2) whenever $v + \Delta$ satisfies an LU-constraint, $v' + \Delta$ also satisfies the same constraint. This is the basis for $\preccurlyeq^{\star}_{LU}$ to induce a simulation over the local semantics. When v, v' are synchronized valuations, the $\preccurlyeq^{\star}_{LU}$ preorder is identical to the $\preccurlyeq_{LU}$ preorder of the global-time semantics.

We use the same notation $\preccurlyeq^{\star}_{LU}$ to denote a relation between configurations: we define $(q, v) \preccurlyeq^{\star}_{LU} (q, v')$ whenever $v \preccurlyeq^{\star}_{LU} v'$. The next theorem states that $\preccurlyeq^{\star}_{LU}$ relation is a strong-timed simulation on $\mathcal{N}$. We illustrate the theorem on an example. Consider a transition $q \to bq_1$ with guard $x_1 > c \wedge x_2 \leq d$ and a reset $\{x_1\}$. Action b is shared between processes 1 and 2. Suppose $(q, v) \to b(q_1, v_1)$. Then $v(t_1) = v(t_2)$, and v satisfies the guard. Let $v \preccurlyeq^{\star}_{LU} v'$. We will see that $(q, v') \to b(q_1, v'_1)$ and $v_1 \preccurlyeq^{\star}_{LU} v'_1$. Firstly, we have $v'(t_1) = v'(t_2)$ by the first item in the $\preccurlyeq^{\star}_{LU}$ definition. Next, we have $v(t_1 - x_1) > c$ and $v(t_2 - x_2) \leq d$. If $v(t_1 - x_1) \leq L(x_1)$, then $v'(t_1 - x_1) \geq v(t_1 - x_1)$ by the second sub-item in the second condition; else $v'(t_1 - x_1) > L(x_1)$ by third sub-item. Since $L(x_1) \geq c$, we get $v'(t_1 - x_1) > c$ in both cases. Similarly, we can argue that $v'(t_2 - x_2) \leq d$ using the first sub-item with $U(x_2)$. Moreover, after resetting x_1, all conditions of $\preccurlyeq^{\star}_{LU}$ are still satisfied in the resulting valuations v_1 and v'_1.

Theorem 6.3. *Let $\mathcal{N}$ be an LU-network. The relation $\preccurlyeq^\star_{LU}$ is a strong-timed simulation on the local semantics of $\mathcal{N}$.*

Definition 6.4. The abstraction operator $\mathfrak{a}^\star_{\preccurlyeq_{LU}}$ is defined, for every set of valuations W, as $\mathfrak{a}^\star_{\preccurlyeq_{LU}}(W) := \{v \mid v \preccurlyeq^\star_{LU} v' \text{ for some } v' \in W\}$. This is the downward closure of W with respect to the $\preccurlyeq^\star_{LU}$ relation.

Theorem 6.5. *For every LU-network $\mathcal{N}$, the abstraction operator $\mathfrak{a}^\star_{\preccurlyeq_{LU}}$ is sound and complete. It also keeps runs.*

Unfortunately, despite this theorem we still miss two pieces to analyze timed networks with local semantics:

- We need an efficient test for $\mathfrak{a}^\star_{\preccurlyeq_{LU}}(Z) \subseteq \mathfrak{a}^\star_{\preccurlyeq_{LU}}(Z')$ because it is used in the definition of $\mathrm{LZG}^{\mathfrak{a}^\star_{\preccurlyeq_{LU}}}(\mathcal{N})$.
- We need $\mathrm{LZG}^{\mathfrak{a}^\star_{\preccurlyeq_{LU}}}(\mathcal{N})$ to be finite.

We discuss an efficient inclusion test in Section 6.1. The impossibility result from Theorem 5.1 tells us that $\mathrm{LZG}^{\mathfrak{a}^\star_{\preccurlyeq_{LU}}}(\mathcal{N})$ cannot be always finite. To address this, we introduce the concept of a bounded-spread network in Section 7 and show that a variant of $\mathrm{LZG}^{\mathfrak{a}^\star_{\preccurlyeq_{LU}}}(\mathcal{N})$ is finite there.

6.1 An algorithm for $\mathfrak{a}^\star_{\preccurlyeq_{LU}}(Z) \subseteq \mathfrak{a}^\star_{\preccurlyeq_{LU}}(Z')$

The counterpart of $\mathfrak{a}^\star_{\preccurlyeq_{LU}}$ in the global-semantics is the abstraction operator $\mathfrak{a}_{\preccurlyeq_{LU}}$ [6]. It is well known that the $\mathfrak{a}_{\preccurlyeq_{LU}}$ abstraction of a zone need not result in a zone, in fact, it may not even be convex [6, 26]. The current abstraction operator $\mathfrak{a}^\star_{\preccurlyeq_{LU}}$ is a generalization of $\mathfrak{a}_{\preccurlyeq_{LU}}$ which is identical to $\mathfrak{a}_{\preccurlyeq_{LU}}$ over zones that contain only synchronized valuations. Therefore, $\mathfrak{a}^\star_{\preccurlyeq_{LU}}$ is not convex. As in the global setting, the challenge is to decide the inclusion $\mathfrak{a}^\star_{\preccurlyeq_{LU}}(Z) \subseteq \mathfrak{a}^\star_{\preccurlyeq_{LU}}(Z')$ by looking at zones Z and Z'. We start with some simplification steps. Since $\mathfrak{a}^\star_{\preccurlyeq_{LU}}$ is the downward closure operator with respect to $\preccurlyeq^\star_{LU}$, we make the first simplication below.

Lemma 6.6. *For every pair of zones Z, Z': $\mathfrak{a}^\star_{\preccurlyeq_{LU}}(Z) \subseteq \mathfrak{a}^\star_{\preccurlyeq_{LU}}(Z')$ iff $Z \subseteq \mathfrak{a}^\star_{\preccurlyeq_{LU}}(Z')$.*

The test $Z \subseteq \mathfrak{a}^\star_{\preccurlyeq_{LU}}(Z')$ can be seen as checking whether for every $v \in Z$ there exists a $v' \in Z'$ such that $v \preccurlyeq^\star_{LU} v'$. Define $\langle v \rangle^\star := \{v' \mid v \preccurlyeq^\star_{LU} v'\}$. The next lemma shows that we can reduce inclusion to intersection.

Lemma 6.7. *Let Z, Z' be non-empty zones. Then, $Z \not\subseteq \mathfrak{a}^\star_{\preccurlyeq_{LU}}(Z')$ iff there exists $v \in Z$ satisfying $\langle v \rangle^\star \cap Z' = \emptyset$.*

As mentioned before, when Z, Z' contain only synchronized valuations, we have $\mathfrak{a}^\star_{\preccurlyeq_{LU}}(Z) = \mathfrak{a}_{\preccurlyeq_{LU}}(Z)$, $\mathfrak{a}^\star_{\preccurlyeq_{LU}}(Z') = \mathfrak{a}_{\preccurlyeq_{LU}}(Z')$ and the test boils down to checking $Z \subseteq \mathfrak{a}_{\preccurlyeq_{LU}}(Z')$, which is studied in [26] for the global semantics. In the local semantics we need to consider valuations that are desynchronized. However, by definition of $\preccurlyeq^\star_{LU}$, for $v \preccurlyeq^\star_{LU} v'$, we require $v(t_p - t_q) = v'(t_p - t_q)$. This property allows us to lift the technique used in [26] to our setting.

For our analysis, we will make use of a graph representation of local-zones, called *distance graphs* [26, 29]. A distance graph has vertices $X \cup X^t$. For every $x, y \in X \cup X^t$ there is an edge $x \longrightarrow y$ with a *weight* that is either $(<, \infty)$ or of the form $(\lessdot, c)$ with $c \in \mathbb{R}$ and $\lessdot$ standing for $\leq$ or $<$. The edge $x \longrightarrow (\lessdot, c) y$ represents the constraint $y - x \lessdot c$. For example, the zone $Z_1 := t_1 - x \geq 5 \wedge t_2 - y \leq 2$ can be represented as a graph with edges: $t_1 \longrightarrow (\leq, -5) x$, $t_2 \longrightarrow (\leq, 0) y$ and $y \longrightarrow (\leq, 2) t_2$. To reason about cumulative constraints of a path in this graph representation, an arithmetic over weights is defined.

Order: for $c_1, c_2 \in \mathbb{R}$, we say $(\lessdot_1, c_1) < (\lessdot_2, c_2)$ if $c_1 < c_2$, or $c_1 = c_2$, $\lessdot_1$ is $<$ and $\lessdot_2$ is $\leq$; secondly, we have $(\lessdot, c) < (<, \infty)$ for every $c \in \mathbb{R}$.

Addition: for $c_1, c_2 \in \mathbb{R}$, we have $(\lessdot_1, c_1) + (\lessdot_2, c_2)$ to be equal to $(\lessdot, d)$ where $d = c_1 + c_2$ and $\lessdot$ is $<$ if one of $\lessdot_1$ or $\lessdot_2$ is $<$, and $\lessdot$ is $\leq$ otherwise; secondly, $(\lessdot, c) + (<, \infty)$ is defined to be $(<, \infty)$ for every weight $(\lessdot, c)$.

The addition allows us to define the weight of a path in a distance graph, as the sum of weights of the edges. A distance graph is *canonical* if for all pairs of vertices $x \neq y$, the smallest weight of a path from x to y is given by the weight of the edge $x \longrightarrow y$. For a zone Z we denote by Z_{xy} the weight of the $x \longrightarrow y$ edge in the canonical distance graph representing Z. We now have all the notation to state our inclusion test.

Theorem 6.8. *Let Z, Z' be non-empty local-zones. We have $Z \not\subseteq \mathfrak{a}^\star_{\preccurlyeq_{LU}}(Z')$ iff there exist two variables $x, y \in X \cup X_t$ s.t.*

- *$Z'_{yx} < Z_{yx}$, and*
- *$(\leq, U_x) + Z_{t_p x} \geq (\leq, 0)$ if $x \in X_p$ for a process p, and*
- *$(<, -L_y) + Z'_{yx} < Z_{t_q x}$, if $y \in X_q$ for some process q.*

The test runs over pairs of variables x, y and uses the weights $Z_{yx}, Z'_{yx}, Z_{t_p x}$ and $Z_{t_q x}$ to check the conditions given by the theorem. This can be acheived in time $O(|X \cup X_t|^2)$. When we look at local-zones consisting of only synchronized valuations, we can add constraints $t_p = t_q$ and derive the test $Z \subseteq \mathfrak{a}_{\preccurlyeq_{LU}}(Z')$ in the global-setting as a special case of the above theorem.

7 BOUNDED-SPREAD NETWORKS

The impossibility result for local-time semantics (Theorem 5.1) says that no simulation based abstraction can ensure finiteness of an abstract zone graph. Even if we go to quasi-abstractions, it is impossible to get a finite abstraction that keeps runs. As we do not know how to obtain abstractions that would go around this problem, we need to look for subclasses of timed networks where abstraction guarantees finiteness. In the example from the proof of Theorem 5.1, the local times of the two processes can differ by an arbitrary amount, and moreover this difference influences future behavior. We give a sufficient condition to avoid this situation.

We introduce the notion of bounded-spread networks and show how we can adapt the $\mathfrak{a}^\star_{\preccurlyeq_{LU}}$ abstraction (Definition 6.4) to get a finite quasi-abstraction that keeps runs for bounded-spread networks. This gives an algorithm for bounded-spread networks that can use both subsumption and partial-order reduction at the same time. We also discuss some cases when a network is guaranteed to be of bounded spread, as well as present a method of converting any network into an equivalent bounded-spread network by adding some synchronizations.

Definition 7.1. The *spread* between processes A_p, A_q in a local valuation v is the absolute value of the difference between their reference clocks: $|v(t_p - t_q)|$. Let $D \geq 0$ be a natural number. We say that a *valuation v has spread D* if the spread between every pair of processes in v is at most D.

Definition 7.2. A run in the local time semantics

$$(q_0, v_0) \xrightarrow{\Delta_0} (q_0, v_0') \xrightarrow{b_1} (q_1, v_1) \xrightarrow{\Delta_1} \cdots \xrightarrow{b_n} (q_n, v_n) \xrightarrow{\Delta_n} (q_n, v_n')$$

is said to be *D-spread* if all $v_0, v_0' \dots, v_n, v_n'$ have spread D.

Definition 7.3. A network $\mathcal{N}$ is said to be *D-spread* if every local run of $\mathcal{N}$ can be converted to a D-spread run by adjusting the delays: that is, for every run $(q_0, v_0) \xrightarrow{\Delta_0} (q_0, v_0') \xrightarrow{b_1} (q_1, v_1) \xrightarrow{\Delta_1} \cdots \xrightarrow{b_n} (q_n, v_n) \xrightarrow{\Delta_n} (q_n, v_n')$ there exists a D-spread run $(q_0, \hat{v}_0) \xrightarrow{\Delta_0'} (q_0, \hat{v}_0') \xrightarrow{b_1} (q_1, \hat{v}_1) \xrightarrow{\Delta_1'} \cdots \xrightarrow{b_n} (q_n, \hat{v}_n) \xrightarrow{\Delta_n'} (q_n, \hat{v'}_n)$ where $\hat{v}_0 = v_0$.

Example. Consider the network in Figure 1a. We have two processes with clocks x and y respectively. There are also two reference clocks t_1 and t_2. Let $v_{i,j}$ stand for a valuation $t_1 = i, t_2 = j, x = y = 0$. In particular, $v_{0,0}$ is an initial valuation. In the local semantics we have a run $(0, 0, v_{0,0}) \xrightarrow{(0,9)} (0, 0, v_{0,9}) \xrightarrow{c} (0, 1, v_{0,9}) \xrightarrow{b} (1, 1, v_{0,9})$. Valuation $v_{0,9}$ has spread 9. Yet the run has a spread 1 because we can adjust the delays: $(0, 0, v_{0,0}) \xrightarrow{(1,2)} (0, 0, v_{1,2}) \xrightarrow{c} (0, 1, v_{1,2}) \xrightarrow{b} (1, 1, v_{1,2})$. If we did not allow for adjusting delays in the definition of D-spread, this network would have an unbounded spread. With the adjustment, it is 1-spread.

7.1 When is a network bounded-spread

We give some examples where it is easy to check that a network is of bounded spread. For such classes we can apply the verification method presented in this work. In general, checking if the spread of a network is bounded by a given D is at least as hard as checking reachability. So an approach consisting of taking an arbitrary network, calculating its spread, and then applying our method, would not work. Observe that every network can be made 0-spread if one makes all the processes synchronize on all actions. However, this removes all parallelism in the network and any possibility of applying partial-order reduction. We show a less radical method of converting any network to a D-spread network. The method introduces some new synchronizations, but still leaves some parallelism where partial-order methods can be applied. We start with some sufficient conditions for a network to be bounded spread and later describe the general construction.

Acyclic systems. We claim that acyclic systems are bounded-spread where the bound depends on the size of the network description. The local-zone graph of an acyclic network is finite and no abstraction is needed. But, making use of "cross" subsumptions reduces the state-space when there are multiple ways to reach a state. Showing that an acyclic system is bounded-spread allows to use both subsumption and partial-order reduction.

Lemma 7.4. *Suppose M is a maximal constant in guards. The spread of a run of length n is bounded by* $nM + 1$.

Proof. Consider a timed automaton and let M be its maximal constant. We claim that the minimal time for executing n actions in the automaton is at most $nM + 1$ in the global semantics. Indeed, in the global-time semantics there is no point of waiting more than M time units in a state, since after waiting M time units the valuation is already in the biggest region and valuations within a region simulate each other (see [3] for the definition and properties of regions). This intuition is less evident in the local-time semantics but we can transfer this observation from global-time to local-time.

Consider a local run on a sequence $b_1 \dots b_n$. By Lemma 2.1 there is a global run on a sequence $c_1 \dots c_n$ such that $b_1 \dots b_n$ is trace equivalent to $c_1 \dots c_n$. This means that there is a bijection $f : [n] \to [n]$ with $b_i = c_{f(i)}$ and respecting order of actions on each process: if b_i and b_j are two actions of process p, and $i < j$ then $f(i) < f(j)$.

The global run has the form:

$$(q_0, v_0) \xrightarrow{\delta_1} (q_0, v_0') \xrightarrow{c_1} (q_1, v_1) \xrightarrow{\delta_2} \dots \xrightarrow{c_n} (q_n, v_n) .$$

We can assume that the cumulated time of this run is at most $nM + 1$; this is because if some δ_i is strictly bigger than M then we can shorten it to $M + \epsilon$ for a $0 < \epsilon < \frac{1}{n}$ as anyway the resulting valuation is the maximal region. Let θ_i be the cumulated time before the i-th action: $\theta_i = \delta_1 + \cdots + \delta_i$. We construct a local-time execution

$$(q_0, v_0) \xrightarrow{\Delta_1} (q_0, v_0') \xrightarrow{b_1} (q_1', v_1) \xrightarrow{\Delta_2} \dots \xrightarrow{b_n} (q_n, v_n) .$$

such that for every i and process $p \in dom(b_i)$ we have $v_i'(t_p) = \theta_{f(i)}$. This means that we execute action b_i exactly at the time when the corresponding action $c_{f(i)}$ was executed in the global run. This constraint determines Δ_i, hence determines the run completely. It can be checked that it is indeed a run: all Δ's are positive, and all guards are satisfied. By definition we have $v_i(t_p) - v_0(t_p) \leq nM + 1$ for all reference clocks and for all i. Since all reference clocks are equal in v_0, we get that the spread is at most $nM + 1$. □

Here is an example where we get the maximal spread:

$$\xrightarrow{b, x>M, x:=0} \cdots \xrightarrow{b, x>M, x:=0} \xrightarrow{a, y=0}$$

Actions a and b are local actions of two processes. At the beginning the two clocks are at 0. The clock of b process gets to some $nM + \epsilon'$ for $0 < \epsilon' < 1$, while the clock of a process is still 0. Lemma 7.4 gives an upper bound for the spread of any run in an acyclic system.

Corollary 7.5. *An acyclic timed network* $\mathcal{N} = \langle A_1, \dots, A_k \rangle$ *is* $(\sum_{i=1}^{i=k} |T_i|) \times M + 1$*-spread bounded where* $|T_i|$ *is the number of transitions in* A_i*, and* M *is the maximum constant in* $\mathcal{N}$.

Frequently communicating systems. More interesting examples of bounded-spread networks are frequently communicating client-server systems. In such systems we have one server process S, and a number of client processes $C_1, \dots, C_n$. The only communication actions are between the server and clients: the domain of an action can be either a singleton or $\{S, C_i\}$ for some i. Such a network is frequently communicating if there is a bound D such that every client communicates with the server in every time interval of length D. It is not difficult to see that in this case the network is $2D$-spread.

Another example is a network with barriers that can be modeled as global synchronizations. Assume there are no other communication actions: each action is either local to a process or it is a global synchronization. If we know that there is a synchronizing action on every loop of every process then the system is bounded-spread thanks to Lemma 7.4.

This idea of using frequent communication to get bounded-spread brings us to the next construction. Any arbitrary network can be converted to a bounded-spread system, at the price of reducing concurrency. Lemma 7.4 suggests adding global synchronizations, say on every loop. This would indeed bound the spread as the length of runs between two global configurations would be bounded. This construction is however not correct: we miss some behaviors of the original system. Another solution is to synchronize all processes every D units of time, which we formalize below.

Definition 7.6. Let $\mathcal{N}$ be an arbitrary timed network and $D \geq 1$ a natural number. Define $\mathcal{N}^D$ to be the timed network obtained from $\mathcal{N}$ as follows. Add a fresh clock z_p to every process p, and a new synchronization action s whose domain is the set of all processes. To every state of every process we add a self-loop on s with a guard $z_p = D$ and reset of z_p. To every other transition add $z_p < D$ to the existing guard.

The construction ensures that in every D units of time every process needs to do the s transition. Hence the resulting system is D-spread bounded. Moreover reachability is preserved as every state that is reachable is reachable by a global run (Lemma 2.1) and global runs are 0-spread. Hence all global runs of $\mathcal{N}$ appear in $\mathcal{N}^D$, with embedded s actions.

Proposition 7.7. *For every network $\mathcal{N}$ and natural number $D \geq 1$, the system $\mathcal{N}^D$ is D-spread. A final state q_f is reachable in $\mathcal{N}$ iff it is reachable in $\mathcal{N}^D$.*

The methodology that we develop in the subsequent section can be applied to $\mathcal{N}^D$. While independence between actions of the original network $\mathcal{N}$ is preserved in $\mathcal{N}^D$, there may be more traces due to new synchronization actions. In Section 8, we will see an example where these extra traces get compensated by partial-order reduction, in fact by an exponential factor.

7.2 Abstraction for bounded-spread networks

We come back to the crucial point of obtaining finite abstractions of bounded-spread networks. For a given bound D we use the $\mathfrak{a}^{\star}_{\preccurlyeq LU}$ abstraction restricted to D-spread valuations. We show that this guarantees finiteness of the abstract graph.

Definition 7.8. For a set of valuations W define $\text{spread}_D(W)$ to be $\{v \in W \mid v \text{ has spread } D\}$. The quasi-abstraction operator $\mathfrak{a}^{D}_{\preccurlyeq LU}$ is defined as $\mathfrak{a}^{D}_{\preccurlyeq LU}(W) := \mathfrak{a}^{\star}_{\preccurlyeq LU}(\text{spread}_D(W))$ for every set of valuations W.

Since $\mathfrak{a}^{D}_{\preccurlyeq LU}$ first restricts to D-spread valuations, let us restrict to zones containing only D-spread valuations. For such zones we have $\mathfrak{a}^{D}_{\preccurlyeq LU}(Z) \subseteq \mathfrak{a}^{D}_{\preccurlyeq LU}(Z')$ iff $Z \subseteq \mathfrak{a}^{\star}_{\preccurlyeq LU}(Z')$. Define an order $Z \preccurlyeq Z'$ if $Z \subseteq \mathfrak{a}^{\star}_{\preccurlyeq LU}(Z')$.

Lemma 7.9. *In every infinite sequence $Z_1, Z_2, \ldots$ of D-spread zones, there exist indices i, j with $i < j$ such that $Z_j \preccurlyeq Z_i$.*

Proof. Suppose there is an infinite sequence $Z_1, Z_2, \ldots$ such that for all j, we have $Z_j \not\preccurlyeq Z_i$ for every $i < j$. If $Z \not\preccurlyeq Z'$ there is a pair of clocks x, y witnessing the non-inclusion. Since the number of such pairs is finite, by standard Ramsey arguments, there are two clocks x, y and an infinite sequence $Z^1, Z^2, \ldots$ such that $Z^{i+1} \not\preccurlyeq Z^i$ is witnessed by x, y for all i. This by Theorem 6.8 means that for all $i \geq 1$ we have:

(1) $Z^i_{yx} < Z^{i+1}_{yx}$
(2) if $x \in X_p$, then $(\leq, U_x) + Z^{i+1}_{t_p x} \geq (\leq, 0)$,
(3) if $y \in X_q$ then $(<, -L_y) + Z^i_{yx} < Z^{i+1}_{t_q x}$

Let us assume that x and y are process clocks, with $x \in X_p$ and $y \in X_q$. The argument below can be adapted to the cases when one of them is a reference clock.

From 1, we see that Z^i_{yx} keeps increasing as i increases and since the zones we get have only integer weights, for every $K \leq 0$, we can find an i such that $Z^i_{yx} > (\leq, K)$. Therefore, from (3), $Z^{i+1}_{t_q x}$ is also increasing. But, $Z^{i+1}_{t_q x} \leq Z^{i+1}_{t_q t_p} + Z^{i+1}_{t_p x}$. We have $Z^{i+1}_{t_q t_p} \leq (\leq, D)$ as Z^{i+1} is D-spread and $Z^{i+1}_{t_p x} \leq (\leq, 0)$ since x is a clock of process p so $x - t_p \leq 0$ for all valuations in Z^{i+1}. Hence $Z^{i+1}_{t_q x} \leq (\leq, D)$. This is a contradiction to the previous inference. □

We now show that we can use the $\mathfrak{a}^{D}_{\preccurlyeq LU}$ abstraction to check state reachability in D-spread networks.

Theorem 7.10. *Quasi-abstraction $\mathfrak{a}^{D}_{\preccurlyeq LU}$ is sound, complete, finite, and keeps runs for D-spread networks.*

Proof. Soundness follows from Lemma 3.4. We now show that $\mathfrak{a}^{D}_{\preccurlyeq LU}$ keeps runs. Consider a D-spread network $\mathcal{N}$ and a node (q, Z) reachable from the initial node in LZG($\mathcal{N}$): there exists a path $(q_0, Z_0) \overset{u'}{\Rightarrow} (q, Z)$. Let $(q, Z) \overset{u}{\Rightarrow} (q_f, Z_f)$ be a path to a final state. By post-property there is a local run $(q_0, v_0) \overset{u'}{\dashrightarrow} (q, v) \overset{u}{\dashrightarrow} (q_f, v_f)$ with $v_0 \in Z_0, v \in Z$ and $v_f \in Z_f$. Since $\mathcal{N}$ is D-spread we can assume $v \in \text{spread}_D(Z)$, and hence by Definition 7.8 we have $v \in \mathfrak{a}^{D}_{\preccurlyeq LU}(Z)$. This proves that $\mathfrak{a}^{D}_{\preccurlyeq LU}$ keeps runs, as per Definition 3.9. Lemma 3.10 then entails that $\mathfrak{a}^{D}_{\preccurlyeq LU}$ is complete. From Lemma 7.9 we know that $\mathfrak{a}^{D}_{\preccurlyeq LU}$ is finite. □

Finally, we show that we can check $\mathfrak{a}^{D}_{\preccurlyeq LU}(Z) \subseteq \mathfrak{a}^{D}_{\preccurlyeq LU}(Z')$ efficiently for time-elapsed zones. Since the local-zone graph has only time-elapsed zones, it is sufficient to consider such zones for the inclusion test.

Lemma 7.11. *Let Z, Z' be time-elapsed zones. The test $\mathfrak{a}^{D}_{\preccurlyeq LU}(Z) \subseteq \mathfrak{a}^{D}_{\preccurlyeq LU}(Z')$ can be done in time $O(|X \cup X_t|^2)$.*

Proof. The test involves two steps: (1) computing the D-spread zones $Z_1 := \text{spread}_D(Z)$ and $Z'_1 := \text{spread}_D(Z')$ and then (2) checking $\mathfrak{a}^{\star}_{\preccurlyeq LU}(Z_1) \subseteq \mathfrak{a}^{\star}_{\preccurlyeq LU}(Z'_1)$. The second step can be done in time $O(|X \cup X_t|^2)$ thanks to Theorem 6.8. We show that $\text{spread}_D(Z)$ can also be computed in the same complexity.

Let G_Z be the canonical distance graph of Z. Let $(\lessdot_{xy}, c_{xy})$ be the weight of $x \to y$ in G_Z. Since Z is time-elapsed, there are no constraints that give an upper bound on the reference clocks, that is, there are no constraints of the form $t_p - x \lessdot c$. This implies that every edge of the form $x \to t_p$ with $x \in X \cup X_t$ and $t_p \in X_t$ has weight $(<, \infty)$. Same is the case with $G_{Z'}$ as Z' is time-elapsed.

Computing $\text{spread}_D(Z)$ involves adding edges $t_p \to (\leq, D) t_q$ between every pair of reference clocks t_p, t_q in G_Z and canonicalizing the resulting graph. Call this resulting graph $\bar{G}$. Since G_Z had no incoming edges to reference clocks, the only incoming edges to t_p in $\bar{G}$ are from other reference clocks. In particular, there are no edges in $\bar{G}$ of the form $x \to t_p$ where $x \in X$. Therefore, the only

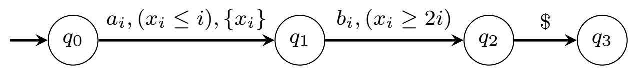


Figure 7: A process in our first example.

shortest paths that can change are of the form $t_p \to y$, where y is a process clock. The shortest path from t_p to y is given by the minimum of $(\lessdot_{t_p y}, c_{t_p y})$ and $(\leq, D) + (\lessdot_{t_q y}, c_{t_q y})$ over all q. This can be computed in time $O(|X_t|)$ for each $t_p \to y$. Overall, ranging over all such edges, we get a complexity $O((|X_t| \cdot |X|) \cdot |X_t|)$. This gives an $O(|X \cup X_t|^2)$ procedure for computing spread$_D(Z)$. The same complexity holds for computing spread$_D(Z')$. □

8 EXAMPLES WITH EXPONENTIAL GAIN

Theorems 7.10 and 4.3, along with Definitions 4.2 and 4.1 give an algorithm for testing reachability in bounded-spread networks: explore the local-zone graph restricted to the successors given by the *src* function, and for each fresh node (q, Z) that is discovered, do not explore further if it is subsumed, that is $\mathfrak{a}^D_{\preccurlyeq LU}(Z) \subseteq \mathfrak{a}^D_{\preccurlyeq LU}(Z')$ for an already visited node (q, Z'). We now present two examples on which this method gives exponential gain.

The first example shows advantages of local-time semantics together with partial-order methods. Consider a network of N timed automata $\mathcal{A}_i$ as depicted in Figure 7, where a_i and b_i are local actions, whereas \$ is a synchronized action.

Notice that not all sequences of actions are feasible in global time. For instance, b_2 requires a delay of 4 time units, hence it cannot happen before a_3 which only allows a delay of 3 time units. Still, all 3^N combinations of states q_0, q_1 and q_2 are reachable, although some of them are deadlocks. Partial-order reduction cannot be applied in algorithms using global-time semantics since the guards remove most diamonds.

In the local-time semantics, all sequences of actions are feasible. The spread is bounded by $2N$. We can apply a very simple partial-order technique: if there is a local action in enabled(q, Z) then keep only the action of the smallest process, otherwise only \$ action is enabled and keep this action. This source function is complete for reachability of the final state $(q_3, \ldots, q_3)$. There is only one source path and it follows the sequence of actions $a_1 b_1 a_2 b_2 \cdots a_N b_N \$$. Without partial-order reduction at least 3^N states are visited.

The second example illustrates the general construction given in Definition 7.6 converting any network to a bounded-spread network (Proposition 7.7). Recall the example of a network with unbounded spread from Figure 6. We consider an extension of it to n processes, $\mathcal{N}_n^-$. We apply to it the construction for bounding the spread to 1, obtaining a network $\mathcal{N}_n^+ = \langle A_1, \ldots, A_n \rangle$, where A_i are as in Figure 8. The actions s and \$ are global actions and b_i is local to A_i.

Consider a *src* function that gives for each (p, Z_σ), the action s and the action b_i with the least index that is enabled at Z_σ. Denote by $\mathrm{LZG}^{1,src}_{LU}(\mathcal{N}_n^+)$ the abstract local-zone graph over the $\mathfrak{a}^D_{\preccurlyeq LU}$ operator with $D = 1$. The uncovered nodes in $\mathrm{LZG}^{1,src}_{LU}(\mathcal{N}_n^+)$ are those accessible by paths $sb_1 \ldots b_i$ for $0 \leq i < n$ and by $sb_1 \ldots b_i s b_1 \ldots b_j$ where $0 \leq i < n$ and $0 \leq j < i$. The covered nodes are the ones accessible by paths $sb_1 \ldots b_n$ and by $sb_1 \ldots b_i s b_1 \ldots b_j s$ where $0 \leq i < n, 0 \leq j \leq i$. This gives an $O(n^2)$ bound on the number

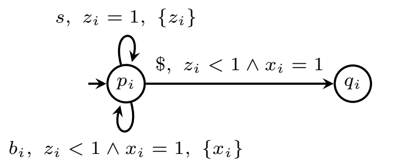

$s,\ z_i = 1,\ \{z_i\}$
$\$,\ z_i < 1 \wedge x_i = 1$
p_i
q_i
$b_i,\ z_i < 1 \wedge x_i = 1,\ \{x_i\}$

Figure 8: A process in our second example.

of nodes in $\mathrm{LZG}^{1,src}_{LU}(\mathcal{N}^+)$. Without partial-order reduction, there will be a node (p, Z_{su}) for each sequence u of b actions without repetitions. This is because Z_{su_1} and Z_{su_2} with u_1 and u_2 not being interleavings of each other cannot be covered with respect to each other by the $\mathfrak{a}^D_{\preccurlyeq LU}$ quasi-abstraction. A detailed analysis shows that without a partial-order method, an exploration needs to visit exponentially many zones, in both local and global-time semantics.

9 CONCLUSION

We have introduced a framework for applying partial-order methods to the analysis of timed automata. It uses local-time semantics in order to regain independence, hence commutativity, of domain-disjoint actions. However, the resulting local-time zone graph is usually infinite, and prior finite abstractions were either impractical or incompatible with partial-order methods. We have introduced a new abstraction $\mathfrak{a}^\star_{\preccurlyeq LU}$ that is simulation based, and hence compatible with partial-order methods. The abstraction $\mathfrak{a}^\star_{\preccurlyeq LU}$ is generally not finite. Even worse, as we have shown here, there does not exist an abstraction that is finite, and simulation based. To circumvent this obstacle, we have introduced bounded-spread timed networks, for which the $\mathfrak{a}^\star_{\preccurlyeq LU}$ abstraction can be made finite. This requires the introduction of quasi-abstractions. We have given examples of subclasses of timed networks that are naturally bounded-spread, and we have shown that every timed network can be made bounded-spread, at the cost of reducing concurrency. We have illustrated the benefits of our framework on two examples.

Our next steps will be designing concrete partial-order methods that provide exponential gains for a wide class of timed networks. We hope that our framework can be extended to other verification problems, like liveness or solving timed games, as well as to richer timed models, like pushdown timed automata, or weighted timed automata.

REFERENCES

[1] Parosh Aziz Abdulla, Stavros Aronis, Bengt Jonsson, and Konstantinos Sagonas. 2014. Optimal dynamic partial order reduction. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, Suresh Jagannathan and Peter Sewell (Eds.). ACM, 373–384. https://doi.org/10.1145/2535838.2535845

[2] Parosh Aziz Abdulla, Stavros Aronis, Bengt Jonsson, and Konstantinos Sagonas. 2017. Source Sets: A Foundation for Optimal Dynamic Partial Order Reduction. *J. ACM* 64, 4 (2017), 25:1–25:49. https://doi.org/10.1145/3073408

[3] Rajeev Alur and David L. Dill. 1994. A Theory of Timed Automata. *Theor. Comput. Sci.* 126, 2 (1994), 183–235. https://doi.org/10.1016/0304-3975(94)90010-8

[4] Gilles Audemard, Alessandro Cimatti, Artur Kornilowicz, and Roberto Sebastiani. 2002. Bounded Model Checking for Timed Systems. In *Formal Techniques for Networked and Distributed Systems - FORTE 2002, 22nd IFIP WG 6.1 International Conference Houston, Texas, USA, November 11-14, 2002, Proceedings (Lecture Notes in Computer Science, Vol. 2529)*, Doron A. Peled and Moshe Y. Vardi (Eds.). Springer, 243–259. https://doi.org/10.1007/3-540-36135-9_16

[5] Bahareh Badban and Martin Lange. 2011. Exact Incremental Analysis of Timed Automata with an SMT-Solver. In *Formal Modeling and Analysis of Timed Systems*

- 9th International Conference, FORMATS 2011, Aalborg, Denmark, September 21-23, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6919), Uli Fahrenberg and Stavros Tripakis (Eds.). Springer, 177–192. https://doi.org/10.1007/978-3-642-24310-3_13

[6] Gerd Behrmann, Patricia Bouyer, Kim Guldstrand Larsen, and Radek Pelánek. 2006. Lower and upper bounds in zone-based abstractions of timed automata. *Int. J. Softw. Tools Technol. Transf.* 8, 3 (2006), 204–215.

[7] Gerd Behrmann, Kim Guldstrand Larsen, Justin Pearson, Carsten Weise, and Wang Yi. 1999. Efficient Timed Reachability Analysis Using Clock Difference Diagrams. In *Computer Aided Verification, 11th International Conference, CAV '99, Trento, Italy, July 6-10, 1999, Proceedings (Lecture Notes in Computer Science, Vol. 1633)*, Nicolas Halbwachs and Doron A. Peled (Eds.). Springer, 341–353. https://doi.org/10.1007/3-540-48683-6_30

[8] Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Wang Yi. 1998. Partial Order Reductions for Timed Systems. In *CONCUR (Lecture Notes in Computer Science, Vol. 1466)*. 485–500.

[9] Frederik M. Bønneland, Peter Gjøl Jensen, Kim Guldstrand Larsen, Marco Muñiz, and Jirí Srba. 2018. Start Pruning When Time Gets Urgent: Partial Order Reduction for Timed Systems. In *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 10981)*, Hana Chockler and Georg Weissenbacher (Eds.). Springer, 527–546. https://doi.org/10.1007/978-3-319-96145-3_28

[10] Frederik M. Bønneland, Peter Gjøl Jensen, Kim G. Larsen, Marco Muñiz, and Jirí Srba. 2021. Stubborn Set Reduction for Timed Reachability and Safety Games. In *Formal Modeling and Analysis of Timed Systems - 19th International Conference, FORMATS 2021, Paris, France, August 24-26, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12860)*, Catalin Dima and Mahsa Shirmohammadi (Eds.). Springer, 32–49. https://doi.org/10.1007/978-3-030-85037-1_3

[11] Frederik Meyer Bønneland, Peter Gjøl Jensen, Kim Guldstrand Larsen, Marco Muñiz, and Jirí Srba. 2021. Stubborn Set Reduction for Two-Player Reachability Games. *Log. Methods Comput. Sci.* 17, 1 (2021). https://lmcs.episciences.org/7278

[12] Patricia Bouyer, Serge Haddad, and Pierre-Alain Reynier. 2006. Timed Unfoldings for Networks of Timed Automata. In *Automated Technology for Verification and Analysis, 4th International Symposium, ATVA 2006, Beijing, China, October 23-26, 2006 (Lecture Notes in Computer Science, Vol. 4218)*, Susanne Graf and Wenhui Zhang (Eds.). Springer, 292–306. https://doi.org/10.1007/11901914_23

[13] Franck Cassez, Thomas Chatain, and Claude Jard. 2006. Symbolic Unfoldings for Networks of Timed Automata. In *Automated Technology for Verification and Analysis, 4th International Symposium, ATVA 2006, Beijing, China, October 23-26, 2006 (Lecture Notes in Computer Science, Vol. 4218)*, Susanne Graf and Wenhui Zhang (Eds.). Springer, 307–321. https://doi.org/10.1007/11901914_24

[14] Krishnendu Chatterjee, Andreas Pavlogiannis, and Viktor Toman. 2019. Value-centric dynamic partial order reduction. *Proc. ACM Program. Lang.* 3, OOPSLA (2019), 124:1–124:29. https://doi.org/10.1145/3360550

[15] Edmund M. Clarke, Orna Grumberg, Marius Minea, and Doron A. Peled. 1999. State Space Reduction Using Partial Order Techniques. *Int. J. Softw. Tools Technol. Transf.* 2, 3 (1999), 279–287. https://doi.org/10.1007/s100090050035

[16] Dennis Dams, Rob Gerth, Bart Knaack, and Ruurd Kuiper. 1998. Partial-order Reduction Techniques for Real-time Model Checking. *Formal Aspects Comput.* 10, 5-6 (1998), 469–482. https://doi.org/10.1007/s001650050028

[17] Conrado Daws and Stavros Tripakis. 1998. Model Checking of Real-Time Reachability Properties Using Abstractions. In *TACAS (Lecture Notes in Computer Science, Vol. 1384)*. Springer, 313–329.

[18] Rüdiger Ehlers, Daniel Fass, Michael Gerke, and Hans-Jörg Peter. 2010. Fully Symbolic Timed Model Checking Using Constraint Matrix Diagrams. In *Proceedings of the 31st IEEE Real-Time Systems Symposium, RTSS 2010, San Diego, California, USA, November 30 - December 3, 2010*. IEEE Computer Society, 360–371. https://doi.org/10.1109/RTSS.2010.36

[19] Cormac Flanagan and Patrice Godefroid. 2005. Dynamic partial-order reduction for model checking software. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2005, Long Beach, California, USA, January 12-14, 2005*, Jens Palsberg and Martín Abadi (Eds.). ACM, 110–121. https://doi.org/10.1145/1040305.1040315

[20] Patrice Godefroid. 1996. *Partial-Order Methods for the Verification of Concurrent Systems - An Approach to the State-Explosion Problem*. Lecture Notes in Computer Science, Vol. 1032. Springer. https://doi.org/10.1007/3-540-60761-7

[21] Patrice Godefroid. 1997. Model Checking for Programming Languages using Verisoft. In *Conference Record of POPL'97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Papers Presented at the Symposium, Paris, France, 15-17 January 1997*, Peter Lee, Fritz Henglein, and Neil D. Jones (Eds.). ACM Press, 174–186. https://doi.org/10.1145/263699.263717

[22] Patrice Godefroid and Pierre Wolper. 1994. A Partial Approach to Model Checking. *Inf. Comput.* 110, 2 (1994), 305–326. https://doi.org/10.1006/inco.1994.1035

[23] R. Govind, Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. 2019. Revisiting Local Time Semantics for Networks of Timed Automata. In *CONCUR (LIPIcs, Vol. 140)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 16:1–16:15.

[24] Henri Hansen, Shang-Wei Lin, Yang Liu, Truong Khanh Nguyen, and Jun Sun. 2014. Diamonds Are a Girl's Best Friend: Partial Order Reduction for Timed Automata with Abstractions. In *CAV (Lecture Notes in Computer Science, Vol. 8559)*. Springer, 391–406.

[25] Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. 2013. Lazy Abstractions for Timed Automata. In *CAV (Lecture Notes in Computer Science, Vol. 8044)*. Springer, 990–1005.

[26] Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. 2016. Better abstractions for timed automata. *Inf. Comput.* 251 (2016), 67–90. https://doi.org/10.1016/j.ic.2016.07.004

[27] Shmuel Katz and Doron A. Peled. 1992. Verification of Distributed Programs Using Representative Interleaving Sequences. *Distributed Comput.* 6, 2 (1992), 107–120. https://doi.org/10.1007/BF02252682

[28] Michalis Kokologiannakis, Iason Marmanis, Vladimir Gladstein, and Viktor Vafeiadis. 2022. Truly Stateless, Optimal Dynamic Partial Order Reduction. *Proc. ACM Program. Lang.* 6, POPL, Article 49 (Jan 2022), 28 pages. https://doi.org/10.1145/3498711

[29] Kim Guldstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. 2003. Compact Data Structures and State-Space Reduction for Model-Checking Real-Time Systems. *Real Time Syst.* 25, 2-3 (2003), 255–275. https://doi.org/10.1023/A:1025132427497

[30] Kim G. Larsen, Marius Mikucionis, Marco Muñiz, and Jirí Srba. 2020. Urgent Partial Order Reduction for Extended Timed Automata. In *Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12302)*, Dang Van Hung and Oleg Sokolsky (Eds.). Springer, 179–195. https://doi.org/10.1007/978-3-030-59152-6_10

[31] Denis Lugiez, Peter Niebert, and Sarah Zennou. 2005. A partial order semantics approach to the clock explosion problem of timed automata. *Theor. Comput. Sci.* 345, 1 (2005), 27–59. https://doi.org/10.1016/j.tcs.2005.07.023

[32] Marius Minea. 1999. Partial Order Reduction for Model Checking of Timed Automata. In *CONCUR (Lecture Notes in Computer Science, Vol. 1664)*. Springer, 431–446.

[33] Jesper B. Møller, Jakob Lichtenberg, Henrik Reif Andersen, and Henrik Hulgaard. 1999. Fully Symbolic Model Checking of Timed Systems using Difference Decision Diagrams. *Electron. Notes Theor. Comput. Sci.* 23, 2 (1999), 88–107. https://doi.org/10.1016/S1571-0661(04)80671-6

[34] Peter Niebert, Moez Mahfoudh, Eugene Asarin, Marius Bozga, Oded Maler, and Navendu Jain. 2002. Verification of Timed Automata via Satisfiability Checking. In *Formal Techniques in Real-Time and Fault-Tolerant Systems, 7th International Symposium, FTRTFT 2002, Co-sponsored by IFIP WG 2.2, Oldenburg, Germany, September 9-12, 2002, Proceedings (Lecture Notes in Computer Science, Vol. 2469)*, Werner Damm and Ernst-Rüdiger Olderog (Eds.). Springer, 225–244. https://doi.org/10.1007/3-540-45739-9_15

[35] Doron A. Peled. 1993. All from One, One for All: on Model Checking Using Representatives. In *Computer Aided Verification, 5th International Conference, CAV '93, Elounda, Greece, June 28 - July 1, 1993, Proceedings (Lecture Notes in Computer Science, Vol. 697)*, Costas Courcoubetis (Ed.). Springer, 409–423. https://doi.org/10.1007/3-540-56922-7_34

[36] Doron A. Peled, Antti Valmari, and Ilkka Kokkarinen. 2001. Relaxed Visibility Enhances Partial Order Reduction. *Formal Methods Syst. Des.* 19, 3 (2001), 275–289. https://doi.org/10.1023/A:1011202615884

[37] Antti Valmari. 1989. Stubborn sets for reduced state space generation. In *Advances in Petri Nets 1990 [10th International Conference on Applications and Theory of Petri Nets, Bonn, Germany, June 1989, Proceedings] (Lecture Notes in Computer Science, Vol. 483)*, Grzegorz Rozenberg (Ed.). Springer, 491–515. https://doi.org/10.1007/3-540-53863-1_36

[38] Antti Valmari. 1992. A Stubborn Attack on State Explosion. *Formal Methods Syst. Des.* 1, 4 (1992), 297–322. https://doi.org/10.1007/BF00709154

[39] Farn Wang. 2001. Symbolic Verification of Complex Real-Time Systems with Clock-Restriction Diagram. In *Formal Techniques for Networked and Distributed Systems, FORTE 2001, IFIP TC6/WG6.1 - 21st International Conference on Formal Techniques for Networked and Distributed Systems, August 28-31, 2001, Cheju Island, Korea (IFIP Conference Proceedings, Vol. 197)*, Myungchul Kim, Byoungmoon Chin, Sungwon Kang, and Danhyung Lee (Eds.). Kluwer, 235–250.

[40] Naling Zhang, Markus Kusano, and Chao Wang. 2015. Dynamic partial order reduction for relaxed memory models. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015*, David Grove and Stephen M. Blackburn (Eds.). ACM, 250–259. https://doi.org/10.1145/2737924.2737956