Partial Order Reductions for Timed Systems

Johan Bengtsson[1] *Bengt Jonsson*[1] *Johan Lilius*[2] *Wang Yi*[1]

[1] Department of Computer Systems, Uppsala University, Sweden.
Email: {bengt,johanb,yi}@docs.uu.se
[2] Department of Computer Science, TUCS, Åbo Akademi University, Finland.
Email: Johan.Lilius@abo.fi

Abstract. In this paper, we present a partial-order reduction method for timed systems based on a *local-time* semantics for networks of timed automata. The main idea is to remove the implicit clock synchronization between processes in a network by letting local clocks in each process advance independently of clocks in other processes, and by requiring that two processes *resynchronize* their local time scales whenever they communicate. A symbolic version of this new semantics is developed in terms of predicate transformers, which enjoys the desired property that two predicate transformers are independent if they correspond to disjoint transitions in different processes. Thus we can apply standard partial order reduction techniques to the problem of checking reachability for timed systems, which avoid exploration of unnecessary interleavings of independent transitions. The price is that we must introduce extra machinery to perform the resynchronization operations on local clocks. Finally, we present a variant of DBM representation of symbolic states in the local time semantics for efficient implementation of our method.

1 Motivation

During the past few years, a number of verification tools have been developed for timed systems in the framework of timed automata (e.g. KRONOS and UPPAAL) [HH95,DOTY95,BLL+96]. One of the major problems in applying these tools to industrial-size systems is the huge memory-usage (e.g. [BGK+96]) needed to explore the state-space of a network (or product) of timed automata, since the verification tools must keep information not only on the control structure of the automata but also on the clock values specified by clock constraints.

Partial-order reduction (e.g., [God96,GW90,HP94,Pel93,Val90,Val93]) is a well developed technique, whose purpose is to reduce the usage of time and memory in state-space exploration by avoiding to explore unnecessary interleavings of independent transitions. It has been successfully applied to finite-state systems. However, for timed systems there has been less progress. Perhaps the major obstacle to the application of partial order reduction to timed systems is the assumption that all clocks advance at the same speed, meaning that all clocks are implicitly synchronized. If each process contains (at least) one local clock, this means that advancement of the local clock of a process is not independent of time advancements in other processes. Therefore, different interleavings

of a set of independent transitions will produce different combinations of clock values, even if there is no explicit synchronization between the processes or their clocks.

A simple illustration of this problem is given in Fig. 1. In (1) of Fig. 1 is a system with two automata, each of which can perform one internal local transition (α_1 and α_2 respectively) from an initial local state to a synchronization state (m, s) where the automata may synchronize on label a (we use the synchronization model of CCS). It is clear that the two sequences of transitions $(l, r) \xrightarrow{\alpha_1} (m, r) \xrightarrow{\alpha_2} (m, s)$ and $(l, r) \xrightarrow{\alpha_2} (l, s) \xrightarrow{\alpha_1} (m, s)$ are different interleavings of two independent transitions, both leading to the state (m, s), from which a synchronization on a is possible. A partial order reduction technique will explore only one of these two interleavings, after having analyzed that the initial transitions of the two automata are independent.

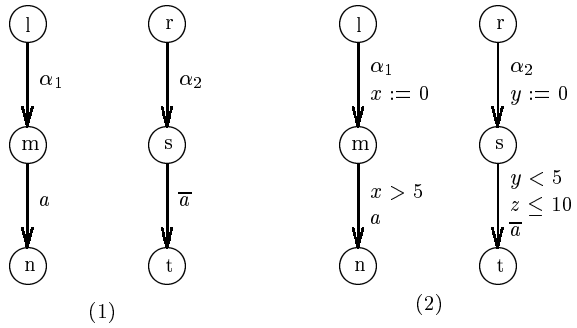

Fig. 1. Illustration of Partial Order Reduction

Let us now introduce timing constraints in terms of clocks into the example, to obtain the system in (2) of Fig. 1 where we add clocks x, y and z. The left automaton can initially move to node m, thereby resetting the clock x, after waiting an arbitrary time. Thereafter it can move to node n after more than 5 time units. The right automaton can initially move to node s, thereby resetting the clock y, after waiting an arbitrary time. Thereafter it can move to node t within 5 time units, but within 10 time units of initialization of the system. We note that the initial transitions of the two automata are logically independent of each other. However, if we naively analyze the possible values of clocks after a certain sequence of actions, we find that the sequence $(l, r) \xrightarrow{\alpha_1} (m, r) \xrightarrow{\alpha_2} (m, s)$ may result in clock values that satisfy $x \geq y$ (as x is reset before y) where the synchronization on a is possible, whereas the sequence $(l, r) \xrightarrow{\alpha_2} (l, s) \xrightarrow{\alpha_1} (m, s)$ may result in clock values that satisfy $x \leq y$ (as x is reset after y) where the synchronization on a is impossible. Now, we see that it is in general not sufficient to explore only one interleaving of independent transitions.

In this paper, we present a new method for partial order reductions for timed systems based on a new local-time semantics for networks of timed automata.

The main idea is to overcome the problem illustrated in the previous example by removing the implicit clock synchronization between processes by letting clocks advance independently of each other. In other words, we *desynchronize* local clocks. The benefit is that different interleavings of independent transitions will no longer remember the order in which the transitions were explored. In this specific example, an interleaving will not "remember" the order in which the clocks were reset, and the two initial transitions are independent. We can then import standard partial order techniques, and expect to get the same reductions as in the untimed case. We again illustrate this on system (2) of Fig. 1. Suppose that in state (l, r) all clocks are initialized to 0. In the standard semantics, the possible clock values when the system is in state (l, r) are those that satisfy $x = y = z$. In the "desynchronized" semantics presented in this paper, any combination of clock values is possible in state (l, r). After both the sequence $(l, r) \xrightarrow{\alpha_1} (m, r) \xrightarrow{\alpha_2} (m, s)$ and $(l, r) \xrightarrow{\alpha_2} (l, s) \xrightarrow{\alpha_1} (m, s)$ the possible clock values are those that satisfy $y \leq z$.

Note that the desynchronization will give rise to many new global states in which automata have "executed" for different amounts of time. We hope that this larger set of states can be represented symbolically more compactly than the original state-space. For example, in system (2), our desynchronized semantics gives rise to the constraint $y \leq z$ at state (m, s), whereas the standard semantics gives rise to the two constraints $x \leq y \leq z$ and $y \leq x \wedge y \leq z$. However, as we have removed the synchronization between local time scales completely, we also lose timing information required for synchronizaton between automata. Consider again system (2) and look at the clock z of the right automaton. Since $z = 0$ initially, the constraint $z \leq 10$ requires that the synchronization on a should be within 10 time units from system initialization. Implicitly, this then becomes a requirement on the left automaton. A naive desynchronization of local clocks including z will allow the left process to wait for more than 10 time units, in its local time scale, before synchronizing. Therefore, before exploring the effect of a transition in which two automata synchronize, we must explicitly "resynchronize" the local time scales of the participating automata. For this purpose, we add to each automaton a local *reference clock*, which measures how far its local time has advanced in performing local transitions. To each synchronization between two automata, we add the condition that their reference clocks agree. In the above example, we add c_1 as a reference clock to the left automaton and c_2 as a reference clock to the right automaton. We require $c_1 = c_2$ at system initialization. After any interleaving of the first two independent transitions, the clock values may satisfy $y \leq z$ and $x - c_1 \leq z - c_2$. To synchronize on a they must also satisfy the constraint $c_1 = c_2$ in addition to $x > 5$, $y < 5$ and $z \leq 10$. This implies that $x \leq 10$ when the synchronization occurs. Without the reference clocks, we would not have been able to derive this condition.

The idea of introducing local time is related to the treatment of local time in the field of parallel simulation (e.g., [Fuj90]). Here, a simulation step involves some local computation of a process together with a corresponding update of its local time. A snapshot of the system state during a simulation will be composed

of many local time scales. In our work, we are concerned with verification rather than simulation, and we must therefore represent sets of such system states symbolically. We shall develop a symbolic version for the local-time semantics in terms of predicate transformers, in analogy with the ordinary symbolic semantics for timed automata, which is used in several tools for reachability analysis. The symbolic semantics allows a finite partitioning of the state space of a network and enjoys the desired property that two predicate transformers are independent if they correspond to disjoint transitions in different component automata. Thus we can apply standard partial order reduction techniques to the problem of checking reachability for timed systems, without disturbance from implicit synchronization of clocks.

The paper is organized as follows: In section 2, we give a brief introduction to the notion of timed automata and its standard semantics i.e. the global time semantics. Section 3 develops a local time semantics for networks of timed automata and a *finite* symbolic version of the new semantics, analogous to the region graph for timed automata. Section 4 presents a partial order search algorithm for reachability analysis based on the symbolic local time semantics; together with necessary operations to represent and manipulate distributed symbolic states. Section 5 concludes the paper with a short summary on related work, our contribution and future work.

2 Preliminaries

2.1 Networks of Timed Automata

Timed automata was first introduced in [AD90] and has since then established itself as a standard model for timed systems. For the reader not familiar with the notion of timed automata we give a short informal description. In this paper, we will work with *networks of timed automata* [YPD94,LPY95] as the model for timed systems.

Let Act be a finite set of *labels* ranged over by a, b etc. Each label is either *local* or *synchronizing*. If a is a synchronizing label, then it has a *complement*, denoted $\overline{a}$, which is also a synchronizing label with $\overline{\overline{a}} = a$.

A timed automaton is a standard finite–state automaton over alphabet Act, extended with a finite collection of real–valued *clocks* to model timing. We use x, y etc. to range over clocks, C and r etc. to range over finite sets of clocks, and $\mathbf{R}$ to stand for the set of non-negative real numbers.

A *clock assignment* u for a set C of clocks is a function from C to $\mathbf{R}$. For $d \in \mathbf{R}$, we use $u + d$ to denote the clock assignment which maps each clock x in C to the value $u(x) + d$ and for $r \subseteq C$, $[r \mapsto 0]u$ to denote the assignment for C which maps each clock in r to the value 0 and agrees with u on $C \backslash r$.

We use $\mathcal{B}(C)$ ranged over by g (and later by D), to stand for the set of conjunctions of atomic constraints of the form: $x \sim n$ or $x - y \sim n$ for $x, y \in C$, $\sim \in \{\leq, <, >, \geq\}$ and n being a natural number. Elements of $\mathcal{B}(C)$ are called *clock constraints* or *clock constraint systems* over C. We use $u \models g$ to denote that the clock assignment $u \in \mathbf{R}^C$ satisfies the clock constraint $g \in \mathcal{B}(C)$.

A *network of timed automata* is the parallel composition $A_1 | \cdots | A_n$ of a collection $A_1, \ldots, A_n$ of timed automata. Each A_i is a timed automaton over the clocks C_i, represented as a tuple $\langle N_i, l_i^0, E_i, I_i \rangle$, where N_i is a finite set of (control) *nodes*, $l_i^0 \in N_i$ is the *initial node*, and $E_i \subseteq N_i \times \mathcal{B}(C_i) \times Act \times 2^{C_i} \times N_i$ is a set of *edges*. Each edge $\langle l_i, g, a, r, l_i' \rangle \in E_i$ means that the automaton can move from the node l_i to the node l_i' if the clock constraint g (also called the enabling condition of the edge) is satisfied, thereby performing the label a and resetting the clocks in r. We write $l_i \xrightarrow{g,a,r} l_i'$ for $\langle l_i, g, a, r, l_i' \rangle \in E_i$. A *local action* is an edge $l_i \xrightarrow{g,a,r} l_i'$ of some automaton A_i with a local label a. A *synchronizing action* is a pair of matching edges, written $l_i \xrightarrow{g_i,a,r_i} l_i' | l_j \xrightarrow{g_j,\overline{a},r_j} l_j'$ where a is a synchronizing label, and for some $i \neq j$, $l_i \xrightarrow{g_i,a,r_i} l_i'$ is an edge of A_i and $l_j \xrightarrow{g_j,\overline{a},r_j} l_j'$ is an edge of A_j. The $I_i : N_i \to \mathcal{B}(C_i)$ assigns to each node an *invariant condition* which must be satisfied by the system clocks whenever the system is operating in that node. For simplicity, we require that the invariant conditions of timed automata should be the conjunction of constraints in the form: $x \leq n$ where x is a clock and n is a natural number. We require the sets C_i to be pairwise disjoint, so that each automaton only references local clocks. As a technical convenience, we assume that the sets N_i of nodes are pairwise disjoint.

Global Time Semantics. A *state* of a network $A = A_1 | \cdots | A_n$ is a pair (l, u) where l, called a *control vector*, is a vector of control nodes of each automaton, and u is a clock assignment for $C = C_1 \cup \cdots \cup C_n$. We shall use $l[i]$ to stand for the ith element of l and $l[l_i'/l_i]$ for the control vector where the ith element l_i of l is replaced by l_i'. We define the invariant $I(l)$ of l as the conjuction $I_1(l[1]) \wedge \cdots \wedge I_n(l[n])$. The initial state of A is (l^0, u^0) where l^0 is the control vector such that $l[i] = l_i^0$ for each i, and u^0 maps all clocks in C to 0.

A network may change its state by performing the following three types of transitions.

- Delay Transition: $(l, u) \longrightarrow (l, u + d)$ if $I(l)(u + d)$
- Local Transition: $(l, u) \longrightarrow (l[l_i'/l_i], u')$ if there exists a local action $l_i \xrightarrow{g,a,r} l_i'$ such that $u \models g$ and $u' = [r \mapsto 0]u$.
- Synchronizing Transition: $(l, u) \longrightarrow (l[l_i'/l_i][l_j'/l_j], u')$ if there exists a synchronizing action $l_i \xrightarrow{g_i,a,r_i} l_i' | l_j \xrightarrow{g_j,\overline{a},r_j} l_j'$ such that $u \models g_i$, $u \models g_j$, and $u' = [r_i \mapsto 0][r_j \mapsto 0]u$.

We shall say that a state (l, u) is *reachable*, denoted $(l^0, u^0) \longrightarrow^* (l, u)$ if there exists a sequence of (delay or discrete) transitions leading from (l^0, u^0) to (l, u).

2.2 Symbolic Global–Time Semantics

Clearly, the semantics of a timed automaton yields an infinite transition system, and is thus not an appropriate basis for verification algorithms. However, efficient

algorithms may be obtained using a *symbolic* semantics based on *symbolic states* of the form (l, D), where $D \in \mathcal{B}(C)$, which represent the set of states (l, u) such that $u \models D$. Let us write $(l, u) \models (l', D)$ to denote that $l = l'$ and $u \models D$.

We perform symbolic state space exploration by repeatedly taking the strongest postcondition with respect to an action, or to time advancement. For a constraint D and set r of clocks, define the constraints $D^{\uparrow}$ and $r(D)$ by

- for all $d \in \mathbf{R}$ we have $u + d \models D^{\uparrow}$ iff $u \models D$, and
- $[r \mapsto 0]u \models r(D)$ iff $u \models D$

It can be shown that $D^{\uparrow}$ and $r(D)$ can be expressed as clock constraints whenever D is a clock constraint. We now define predicate transformers corresponding to strongest postconditions of the three types of transitions:

- For global delay, $sp(\delta)(l, D) \stackrel{def}{=} \left(l, D^{\uparrow} \wedge I(l)\right)$
- For a local action $l_i \stackrel{g,a,r}{\longrightarrow} l_i'$ $sp(l_i \stackrel{g,a,r}{\longrightarrow} l_i')(l, D) \stackrel{def}{=} \left(l[l_i'/l_i], r(g \wedge D)\right)$
- For a synchronizing action $l_i \stackrel{g_i,a,r_i}{\longrightarrow} l_i' | l_j \stackrel{g_j,\overline{a},r_j}{\longrightarrow} l_j'$,
  $sp(l_i \stackrel{g_i,a,r_i}{\longrightarrow} l_i' | l_j \stackrel{g_j,\overline{a},r_j}{\longrightarrow} l_j')(l, D) \stackrel{def}{=} \left(l[l_i'/l_i][l_j'/l_j], (r_i \cup r_j)(g_i \wedge g_j \wedge D)\right)$

It turns out to be convenient to use predicate transformers that correspond to first executing a discrete action, and thereafter executing a delay. For predicate transformers τ_1, τ_2, we use $\tau_1; \tau_2$ to denote the composition $\tau_2 \circ \tau_1$. For a (local or synchronizing) action α, we define $sp_t(\alpha) \stackrel{def}{=} sp(\alpha); sp(\delta)$.

From now on, we shall use (l^0, D^0) to denote the initial symbolic global time state for networks, where $D^0 = (\{u^0\})^{\uparrow} \wedge I(l^0)$. We write $(l, D) \Rightarrow (l', D')$ if $(l', D') = sp_t(\alpha)(l, D)$ for some action α. It can be shown (e.g. [YPD94]) that the symbolic semantics characterizes the concrete semantics given earlier in the following sense:

Theorem 1. *A state (l, u) of a network is reachable if and only if $(l^0, D^0)(\Rightarrow)^*(l, D)$ for some D such that $u \models D$.*

The above theorem can be used to construct a symbolic algorithm for reachability analysis. In order to keep the presentation simple, we will in the rest of the paper only consider a special form of *local* reachability, defined as follows. Given a control node l_k of some automaton A_k, check if there is a reachable state (l, u) such that $l[k] = l_k$. It is straight-forward to extend our results to more general reachability problems. The symbolic algorithm for checking local reachability is shown in Figure 2 for a network of timed automata. Here, the set $enabled(l)$ denotes the set of all actions whose source node(s) are in the control vector l i.e., a local action $l_i \stackrel{g,a,r}{\longrightarrow} l_i'$ is enabled at l if $l[i] = l_i$, and a synchronizing action $l_i \stackrel{g_i,a,r_i}{\longrightarrow} l_i' | l_j \stackrel{g_j,\overline{a},r_j}{\longrightarrow} l_j'$ is enabled at l if $l[i] = l_i$ and $l[j] = l_j$.

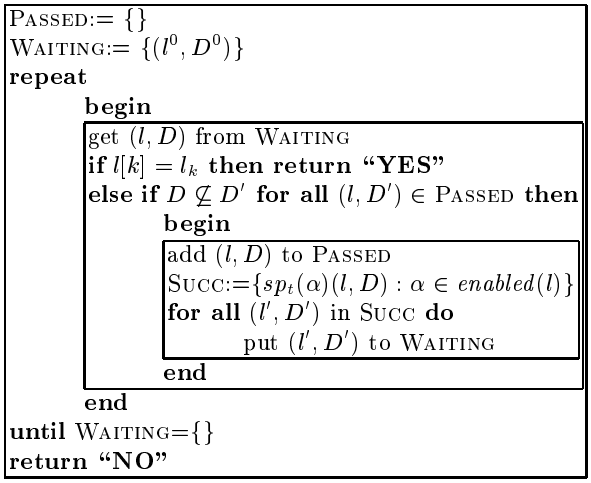

Fig. 2. An Algorithm for Symbolic Reachability Analysis.

3 Partial Order Reduction and Local–Time Semantics

The purpose of partial-order techniques is to avoid exploring several interleavings of independent transitions, i.e., transitions whose order of execution is irrelevant, e.g., because they are performed by different processes and do not affect each other. Assume for instance that for some control vector l, the set $\mathit{enabled}(l)$ consists of the local action α_i of automaton A_i and the local action α_j of automaton A_j. Since executions of local actions do not affect each other, we might want to explore only the action α_i, and defer the exploration of α_j until later. The justification for deferring to explore α_j would be that any symbolic state which is reached by first exploring α_j and thereafter α_i can also be reached by exploring these actions in reverse order, i.e., first α_i and thereafter α_j.

Let τ_1 and τ_2 be two predicate transformers. We say that τ_1 and τ_2 are *independent* if $(\tau_1; \tau_2)((l, D)) = (\tau_2; \tau_1)((l, D))$ for any symbolic state (l, D). In the absence of time, local actions of different processes are independent, in the sense that $sp(\alpha_i)$ and $sp(\alpha_j)$ are independent. However, in the presence of time, we do not have independence. That is, $sp_t(\alpha_i)$ and $sp_t(\alpha_j)$ are in general not independent, as illustrated e.g., by the example in Figure 1.

If timed predicate transformers commute only to a rather limited extent, then partial order reduction is less likely to be successful for timed systems than for untimed systems. In this paper, we present a method for symbolic state-space exploration of timed systems, in which predicate transformers commute to the same extent as they do in untimed systems. The main obstacle for commutativity of timed predicate transformers is that timed advancement is modeled by globally synchronous transitions, which implicitly synchronize all local clocks, and hence all processes. In our approach, we propose to replace the global time-advancement steps by local-time advancement. In other words, we remove the

constraint that all clocks advance at the same speed and let clocks of each automaton advance totally independently of each other. We thus replace one global time scale by a local-time scale for each automaton. When exploring local actions, the corresponding predicate transformer affects only the clocks of that automaton in its local-time scale; the clocks of other automata are unaffected. In this way, we have removed any relation between local-time scales. However, in order to explore pairs of synchronizing actions we must also be able to "resynchronize" the local-time scales of the participating automata, and for this purpose we add a local *reference clock* to each automaton. The reference clock of automaton A_i represents how far the local-time of A_i has advanced, measured in a global time scale. In a totally unsynchronized state, the reference clocks of different automata can be quite different. Before a synchronization between A_i and A_j, we must add the condition that the reference clocks of A_i and A_j are equal.

To formalize the above ideas further, we present a local-time semantics for networks of timed automata, which allows local clocks to advance independently and resynchronizing them only at synchronization points.

Consider a network $A_1|\cdots|A_n$. We add to the set C_i of clocks of each A_i a reference clock, denoted c_i. Let us denote by $u +_i d$ the time assignment which maps each clock x in C_i (including c_i) to the value $u(x) + d$ and each clock x in $C \setminus C_i$ to the value $u(x)$. In the rest of the paper, we shall assume that the set of clocks of a network include the reference clocks and the initial state is (l^0, u^0) where the reference clock values are 0, in both the global and local time semantics.

Local Time Semantics. The following rules define that networks may change their state locally and globally by performing three types of transitions:

- Local Delay Transition: $(l,u)\mapsto(l,u+_i d)$ if $I_i(l_i)(u+_i d)$
- Local Discrete Transition: $(l,u)\mapsto(l[l'_i/l_i],u')$ if there exists a local action $l_i \xrightarrow{g,a,r} l'_i$ such that $u \models g$ and $u' = [r \mapsto 0]u$
- Synchronizing Transition: $(l,u)\mapsto(l[l'_i/l_i][l'_j/l_j],u')$ if there exists a synchronizing action $l_i \xrightarrow{g_i,a,r_i} l'_i | l_j \xrightarrow{g_j,\overline{a},r_j} l'_j$ such that $u \models g_i$, $u \models g_j$, and $u' = [r_i \mapsto 0][r_j \mapsto 0]u$, and $u(c_i) = u(c_j)$

Intuitively, the first rule says that a component may advance its local clocks (or execute) as long as the local invariant holds. The second rule is the standard interleaving rule for discrete transitions. When two components need to synchronize, it must be checked if they have executed for the same amount of time. This is specified by the last condition of the third rule which states that the local reference clocks must agree, i.e. $u(c_i) = u(c_j)$.

We call (l,u) a local time state. Obviously, according to the above rules, a network may reach a large number of local time states where the reference clocks take different values. To an external observer, the interesting states of a network will be those where all the reference clocks take the same value.

Definition 1. *A local time state (l,u) with reference clocks $c_1 \cdots c_n$ is synchronized if $u(c_1) = \cdots = u(c_n)$.*

Now we claim that the local-time semantics simulates the standard global time semantics in which local clocks advance concurrently, in the sense that they can generate precisely the same set of reachable states of a timed system.

Theorem 2. *For all networks,* $(l_0, u_0)(\longrightarrow)^*(l, u)$ *iff for all synchronized local time states* (l, u) $(l_0, u_0)(\mapsto)^*(l, u)$.

3.1 Symbolic Local–Time Semantics

We can now define a local-time analogue of the symbolic semantics given in Section 2.2 to develop a symbolic reachability algorithm with partial order reduction. We need to represent local time states by constraints. Let us first assume that the constraints we need for denote symbolic local time states are different from standard clock constraints, and use $\widehat{D}, \widehat{D}'$ etc to denote such constraints. Later, we will show that such constraints can be expressed as a clock constraint.

We use $\widehat{D}^{\uparrow_i}$ to denote the clock constraint such that for all $d \in \mathbf{R}$ we have $u +_i d \models \widehat{D}^{\uparrow_i}$ iff $u \models \widehat{D}$. For local-time advance, we define a *local-time predicate transformer*, denoted $\widehat{sp_t}(\delta_i)$, which allows only the local clocks C_i including the reference clock c_i to advance as follows:

- $\widehat{sp_t}(\delta_i)(l, \widehat{D}) \stackrel{def}{=} \left(l, \widehat{D}^{\uparrow_i} \wedge I(l)\right)$

For each local and synchronizing action α, we define a local-time predicate transformer, denoted $\widehat{sp_t}(\alpha)$, as follows:

- If α is a local action $l_i \stackrel{g,a,r}{\longrightarrow} l_i'$, then $\widehat{sp_t}(\alpha) \stackrel{def}{=} sp(\alpha); \widehat{sp_t}(\delta_i)$
- If α is a synchronizing action $l_i \stackrel{g_i,a,r}{\longrightarrow} l_i' | l_j \stackrel{g_j,\overline{a},r_j}{\longrightarrow} l_j'$, then $\widehat{sp_t}(\alpha) \stackrel{def}{=} \{c_i = c_j\}; sp(\alpha); \widehat{sp_t}(\delta_i); \widehat{sp_t}(\delta_j)$

Note that in the last definition, we treat a clock constraint like $c_i = c_j$ as a predicate transformer, defined in the natural way by $\{c_i = c_j\}(l, \widehat{D}) \stackrel{def}{=} (l, \widehat{D} \wedge (c_i = c_j))$.

We use $(l^0, \widehat{D}^0)$ to denote the initial symbolic local time state of networks where $\widehat{D}^0 = \widehat{sp_t}(\delta_1); \cdots; \widehat{sp_t}(\delta_n)(\{u^0\})$. We shall write $(l, \widehat{D}) \models\!\!\Rightarrow (l', \widehat{D}')$ if $(l', \widehat{D}') = \widehat{sp_t}(\alpha)(l, \widehat{D})$ for some action α.

Then we have the following characterization theorem.

Theorem 3. *For all networks, a synchronized state* (l, u), $(l^0, u^0) \longrightarrow^* (l, u)$ *if and only if* $(l^0, \widehat{D}^0)(\models\!\!\Rightarrow)^*(l, \widehat{D})$ *for a symbolic local time state* $(l, \widehat{D})$ *such that* $u \models \widehat{D}$.

The above theorem shows that the symbolic local time semantics fully characterizes the global time semantics in terms of reachable states. Thus we can perform reachability analysis in terms of the symbolic local time semantics. However, it requires to find a symbolic local time state that is *synchronized*

in the sense that it constains synchronized states. The searching for such a synchronized symbolic state may be time and space-consuming. Now, we relax the condition for a class of networks, namely those containing no local time-stop.

Definition 2. *A network is local time-stop free if for all* (l,u), $(l^0,u^0)(\mapsto)^*(l,u)$ *implies* $(l,u)(\mapsto)^*(l',u')$ *for some synchronized state* (l',u').

The local time-stop freeness can be easily guaranteed by syntactical restriction on component automata of networks. For example, we may require that at each control node of an automaton there should be an edge with a local label and a guard weaker than the local invariant. This is precisely the way of modelling time-out handling at each node when the invariant is becoming false and therefore it is a natural restriction.

The following theorem allows us to perform reachability analysis in terms of symbolic local time semantics for local time-stop free networks without searching for synchronized symbolic states.

Theorem 4. *Assume a local time-stop free network* A *and a local control node* l_k *of* A_k. *Then* $(l^0,D^0)(\Rightarrow)^*(l,D)$ *for some* (l,D) *such that* $l[k]=l_k$ *if and only if* $(l^0,\widehat{D}^0)(\models\!\!\Rightarrow)^*(l',\widehat{D}')$ *for some* $(l',\widehat{D}')$ *such that* $l'[k]=l_k$.

We now state that the version of the timed predicate transformers based on local time semantics enjoy the commutativity properties that were missing in the global time approach.

Theorem 5. *Let* α_1 *and* α_2 *be two actions of a network* A *of timed automata. If the sets of component automata of* A *involved in* α_1 *and* α_2 *are disjoint, then* $\widehat{sp_t}(\alpha_1)$ *and* $\widehat{sp_t}(\alpha_2)$ *are independent.*

3.2 Finiteness of the Symbolic Local Time Semantics

We shall use the symbolic local time semantics as the basis to develop a partial order search algorithm in the following section. To guarantee termination of the algorithm, we need to establish the finiteness of our local time semantics, i.e. that the number of *equivalent* symbolic states is finite. Observe that the number of symbolic local time states is in general infinite. However, we can show that there is finite partitioning of the state space. We take the same approach as for standard timed automata, that is, we construct a finite graph based on a notion of regions.

We first extend the standard region equivalence to synchronized states. In the following we shall use C_r to denote the set of reference clocks.

Definition 3. *Two synchronized local time states (with the same control vector)* (l,u) *and* (l,u') *are synchronized-equivalent if* $([C_r\mapsto 0]u)\sim([C_r\mapsto 0]u')$ *where* $\sim$ *is the standard region equivalence for timed automata.*

Note that $([C_r \mapsto 0]u) \sim ([C_r \mapsto 0]u')$ means that only the non-reference clock values in (l, u) and (l, u') are region-equivalent. We call the equivalence classes w.r.t. the above equivalence relation *synchronized regions*. Now we extend this relation to cope with local time states that are not synchronized. Intuitively, we want two non-synchronized states, (l, u) and (l', u') to be classified as equivalent if they can reach sets of equivalent synchronized states just by letting the automata that have lower reference clock values advance to catch up with the automaton with the highest reference clock value.

Definition 4. *A local delay transition $(l, u) \mapsto (l, u')$ of a network is a catch-up transition if $max(u(C_r)) \leq max(u'(C_r))$.*

Intuitively a catch-up transition corresponds to running one of the automata that lags behind, and thus making the system more synchronized in time.

Definition 5. *Let (l, u) be a local time state of a network of timed automata. We use $R((l, u))$ to denote the set of synchronized regions reachable from (l, u) only by discrete transitions or catch-up transitions.*

We now define an equivalence relation between local time states.

Definition 6. *Two local time states (l, u) and (l', u') are catch-up equivalent denoted $(l, u) \sim_c (l', u')$ if $R((l, u) = R((l', u'))$. We shall use $|(l, u)|_{\sim_c}$ to denote the equivalence class of local time states w.r.t. $\sim_c$.*

Intuitively two catch-up equivalent local time states can reach the same set of synchronized states i.e. states where all the automata of the network have been synchronized in time.

Note that the number of synchronized regions is finite. This implies that the number of catch-up classes is also finite. On the other hand, there is no way to put an upper bound on the reference clocks c_i, since that would imply that for every process there is a point in time where it stops evolving which is generally not the case. This leads to the conclusion that there must be a periodicity in the region graph, perhaps after some initial steps. Nevertheless, we have a finiteness theorem.

Theorem 6. *For any network of timed automata, the number of catch-up equivalence classes $|(l, u)|_{\sim_c}$ for each vector of control nodes is bounded by a function of the number of regions in the standard region graph construction for timed automata.*

As the number of vectors of control nodes for each network of automata is finite, the above theorem demonstrates the finiteness of our symbolic local time semantics.

4 Partial Order Reduction in Reachability Analysis

The preceding sections have developed the necessary machinery for presenting a method for partial-order reduction in a symbolic reachability algorithm. Such an algorithm can be obtained from the algorithm in Figure 2 by replacing the initial symbolic global time state (l^0, D^0) by the initial symbolic local time state $(l^0, \widehat{D}^0)$ (as defined in Theorem 4), and by replacing the statement

$$\text{SUCC}:=\{sp_t(\alpha)(l, D) : \alpha \in \mathit{enabled}(l)\}$$

by $\text{SUCC}:=\{\widehat{sp_t}(\alpha)(l, D) : \alpha \in \mathit{ample}(l)\}$ where $\mathit{ample}(l) \subseteq \mathit{enabled}(l)$ is a subset of the actions that are enabled at l. Hopefully the set $\mathit{ample}(l)$ can be made significantly smaller than $\mathit{enabled}(l)$, leading to a reduction in the explored symbolic state-space.

In the literature on partial order reduction, there are several criteria for choosing the set $\mathit{ample}(l)$ so that the reachability analysis is still complete. We note that our setup would work with any criterion which is based on the notion of "independent actions" or "independent predicate transformers". A natural criterion which seems to fit our framework was first formulated by Overman [Ove81]; we use its formulation by Godefroid [God96].

The idea in this reduction is that for each control vector l we choose a subset $\mathcal{A}$ of the automata $A_1, \ldots, A_n$, and let $\mathit{ample}(l)$ be all enabled actions in which the automata in $\mathcal{A}$ participate. The choice of $\mathcal{A}$ may depend on the control node l_k that we are searching for. The set $\mathcal{A}$ must satisfy the criteria below. Note that the conditions are formulated only in terms of the control structure of the automata. Note also that in an implementation, these conditions will be replaced by conditions that are easier to check (e.g. [God96]).

C0 $\mathit{ample}(l) = \emptyset$ if and only if $\mathit{enabled}(l) = \emptyset$.
C1 If the automaton $A_i \in \mathcal{A}$ from its current node $l[i]$ can possibly synchronize with another process A_j, then $A_j \in \mathcal{A}$, regardless of whether such a synchronization is enabled or not.
C2 From l, the network cannot reach a control vector l' with $l'[k] = l_k$ without performing an action in which some process in $\mathcal{A}$ participates.

Criteria **C0** and **C2** are obviously necessary to preserve correctness. Criterion **C1** can be intuitively motivated as follows: If automaton A_i can possibly synchronize with another automaton A_j, then we must explore actions by A_j to allow it to "catch up" to a possible synchronization with A_i. Otherwise we may miss to explore the part of the state-space that can be reached after the synchronization between A_i and A_j.

A final necessary criterion for correctness is *fairness*, i.e., that we must not indefinitly neglect actions of some automaton. Otherwise we may get stuck exploring a cyclic behavior of a subset of the automata. This criterion can be formulated in terms of the *global control graph* of the network. Intuitively, this graph has control vectors as nodes, which are connected by symbolic transitions where the clock constraints are ignored. The criterion of fairness then requires that

C3 In each cycle of the global control graph, there must be at least one control vector at which $ample(l) = enabled(l)$.

In the following theorem, we state correctness of our criteria.

Theorem 7. *A partial order reduction of the symbolic reachability in Figure 2, obtained by replacing*

1. *the initial symbolic global time state* (l^0, D^0) *with the initial symbolic local time state* $(l^0, \widehat{D}^0)$ *(as defined in theorem 4)*
2. *the statement* SUCC:=$\{sp_t(\alpha)(l, D) : \alpha \in enabled(l)\}$ *with the statement* SUCC:=$\{\widehat{sp_t}(\alpha)(l, D) : \alpha \in ample(l)\}$ *where the function* $ample(\cdot)$ *satisfies the criteria* **C0 - C3**,
3. *and finally the inclusion checking i.e.* $D \not\subseteq D'$ *between constraints with an inclusion checking that also takes* $\sim_c$ *into account*[1].

is a correct and complete decision procedure for determining whether a local state l_k *in* A_k *is reachable in a local time-stop free network* A.

The proof of the above theorem follows similar lines as other standard proofs of correctness for partial order algorithms. See e.g., [God96].

4.1 Operations on Constraint Systems

Finally, to develop an efficient implementation of the search algorithm presented above, it is important to design efficient data structures and algorithms for the representation and manipulation of symbolic distributed states i.e. constraints over local clocks including the reference clocks.

In the standard approach to verification of timed systems, one such well-known data structure is the Difference Bound Matrix (DBM), due to Bellman [Bel57], which offers a canonical representation for *clock constraints*. Various efficient algorithms to manipulate (and analyze) DBM's have been developed (e.g [LLPY97]).

However when we introduce operations of the form $\widehat{sp_t}(\delta_i)$, the standard clock constraints are no longer adequate for describing possible sets of clock assignments, because it is not possible to let only a subset of the clocks grow. This problem can be circumvented by the following. Instead of considering values of clocks x as the basic entity in a clock constraint, we work in terms of the relative offset of a clock from the local reference clock. For a clock $x_i^l \in C_i$, this offset is represented by the difference $x_i^l - c_i$. By analogy, we must introduce the constant offset $0 - c_i$. An *offset constraint* is then a conjunction of inequalities of form $x_i \sim n$ or $(x_i^l - c_i) - (x_j^k - c_j) \sim n$ for $x_i^l \in C_i, x_j^k \in C_j$, where $\sim \in \{\leq, \geq\}$. Note that an inequality of the form $x_i^l \sim n$ is also an offset, since it is the same as

[1] This last change is only to guarantee the termination but not the soundness of the algorithm. Note that in this paper, we have only shown that there exists a finite partition of the local time state space according to $\sim_c$, but not how the partitioning should be done. This is our future work.

$(x_i^l - c_i) - (0 - c_i) \sim n$. It is important to notice, that given an offset constraint $(x_i^l - c_i) - (x_j^k - c_j) \sim n$ we can always recover the absolute constraint by setting $c_i = c_j$.

The nice feature of these constraints is that they can be represented by DBM's, by changing the interpretation of a clock from being its value to being its local offset. Thus given a set of offset constraints D over a C, we construct a DBM M as follows. We number the clocks in C_i by $x_i^0, \ldots, x^{|C_i|-2}, c_i$. An offset of the form $x_i^l - c_i$ we denote by $\hat{x}_i^l$ and a constant offset $0 - c_i$ by $\hat{c}_i$. The index set of the matrix is then the set of offsets $\hat{x}_i^l$ and $\hat{c}_i$ for $x_i^l, c_i \in C_i$ for all $C_i \in C$, while an entry in M is defined by $M(\hat{x}, \hat{y}) = n$ if $\hat{x} - \hat{y} \leq n \in D$ and $M(\hat{x}, \hat{y}) = \infty$ otherwise. We say that a clock assignment u is a solution of a DBM M, $u \models M$, iff $\forall x, y \in C : \quad u(\hat{x}) - u(\hat{y}) \leq M(\hat{x}, \hat{y})$, where $u(\hat{x}) = u(x) - u(c_i)$ with c_i the reference clock of x.

The operation $D^{\uparrow_i}$ now corresponds to the deletion of all constraints of the form $\hat{c}_i \geq \hat{x} + n$. The intuition behind this is that when we let the clocks in i grow, we are keeping the relative offsets $\hat{x}_i^k$ constant, and only the clock $\hat{c}_i$ will decrease, because this offset is taken from 0. $D^{\uparrow_i}$ can be defined as an operation on the corresponding DBM M: $M^{\uparrow_i}(\hat{x}, \hat{y}) = \infty$ if $\hat{y} = \hat{c}_i$ and $M^{\uparrow_i}(\hat{x}, \hat{y}) = M(\hat{x}, \hat{y})$ otherwise. It then easy to see that $u \models M$ iff $u +_i d \models M^{\uparrow_i}$.

Resetting of a clock x_i^k corresponds to the deletion of all constraints regarding $\hat{x}_i^k$ and then setting $\hat{x}_i^k - \hat{c}_i = 0$. This can be done by an operation $[x_i^k \to 0](M)(\hat{x}, \hat{y}) = 0$ if $\hat{x} = \hat{x}_i^k$ and $\hat{y} = \hat{c}_i$ or $\hat{x} = \hat{c}_i$ and $\hat{y} = \hat{x}_i^k$, ∞ if $\hat{x} = \hat{x}_i^k$ and $\hat{y} \neq \hat{c}_i$ or $\hat{x} \neq \hat{c}_i$ and $\hat{y} = \hat{x}_i^k$, and $M(\hat{x}, \hat{y})$ otherwise. Again it is easy to see, that $[x_i^k \to 0]u \models [x_i^k \to 0](M)$ iff $u \models M$.

5 Conclusion and Related Work

In this paper, we have presented a partial-order reduction method for timed systems, based on a *local-time* semantics for networks of timed automata. We have developed a symbolic version of this new (local time) semantics in terms of predicate transformers, in analogy with the ordinary symbolic semantics for timed automata which is used in current tools for reachability analysis. This symbolic semantics enjoys the desired property that two predicate transformers are independent if they correspond to disjoint transitions in different processes. This allows us to apply standard partial order reduction techniques to the problem of checking reachability for timed systems, without disturbance from implicit synchronization of clocks. The advantage of our approach is that we can avoid exploration of unnecessary interleavings of independent transitions. The price is that we must introduce extra machinery to perform the resynchronization operations on local clocks. On the way, we have established a theorem about finite partitioning of the state space, analogous to the region graph for ordinary timed automata. For efficient implementation of our method, we have also presented a variant of DBM representation of symbolic states in the local time semantics. We should point out that the results of this paper can be easily extended to deal with shared variables by modifying the predicate transformer in the form

$c_i = c_j$) for clock resynchronization to the form $c_i \leq c_j$ properly for the reading and writing operations. Future work naturally include an implementation of the method, and experiments with case studies to investigate the practical significance of the approach.

Related Work Currently we have found in the literature only two other proposals for partial order reduction for real time systems: The approach by Pagani in [Pag96] for timed automata (timed graphs), and the approach of Yoneda et al. in [YSSC93,YS97] for time Petri nets.

In the approach by Pagani a notion of independence between transitions is defined based on the global-time semantics of timed automata. Intuitively two transitions are independent iff we can fire them in any order and the resulting states have the same control vectors and clock assignments. When this idea is lifted to the symbolic semantics, it means that two transitions can be independent only if they can happen in the same global time interval. Thus there is a clear difference to our approach: Pagani's notion of independence requires the comparison of clocks, while ours doesn't.

Yoneda et al. present a partial order technique for model checking a timed LTL logic on time Petri nets [BD91]. The symbolic semantics consists of constraints on the differences on the possible firing times of enabled transitions instead of clock values. Although the authors do not give an explicit definition of independence (like our Thm. 5) their notion of independence is structural like ours, because the persistent sets, ready sets, are calculated using the structure of the net. The difference to our approach lies in the calculation of the next state in the state-space generation algorithm. Yoneda et al. store the relative firing order of enabled transitions in the clock constraints, so that a state implicitly remembers the history of the system. This leads to branching in the state space, a thing which we have avoided. A second source of branching in the state space is synchronization. Since a state only contains information on the relative differences of firing times of transitions it is not possible to synchronize clocks.

Acknowledgement: We would like to thank Paul Gastin, Florence Pagani and Stavros Tripakis for their valuable comments and discussions.

References

[AD90] R. Alur and D. Dill. Automata for Modelling Real-Time Systems. In *Proc. of of International Colloquium on Algorithms, Languages and Programming*, vol. 443 of *LNCS*, pp. 322–335. Springer Verlag, 1990.

[BD91] B. Berthomieu and M. Diaz. Modelling and verification of time dependent systems using time Petri nets. *IEEE Transactions on Software Engineering*, 17(3):259–273, 1991.

[Bel57] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[BGK+96] J. Bengtsson, D. Griffioen, K. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Verification of an Audio Protocol with Bus

Collision Using UPPAAL. In *Proc. of 9th Int. Conf. on Computer Aided Verification*, vol. 1102 of *LNCS*, pp. 244–256. Springer Verlag, 1996.

[BLL+96] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL in 1995. In *Proc. of the 2nd Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, vol. 1055 of *Lecture Notes in Computer Science*, pp. 431–434. Springer Verlag, 1996.

[DOTY95] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Proc. of Workshop on Verification and Control of Hybrid Systems III*, vol. 1066 of *LNCS*, pp. 208–219. Springer Verlag, 1995.

[Fuj90] R. M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, Oct. 1990.

[God96] P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem*, vol. 1032 of *LNCS*. Springer Verlag, 1996.

[GW90] P. Godefroid and P. Wolper. Using partial orders to improve automatic verification methods. In *Proc. of Workshop on Computer Aided Verification*, 1990.

[HH95] T. A. Henzinger and P.-H. Ho. HyTech: The Cornell HYbrid TECHnology Tool. *Proc. of Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, 1995. BRICS report series NS–95–2.

[HP94] G. J. Holzmann and D. A. Peled. An improvement in formal verification. In *Proc. of the 7th International Conference on Formal Description Techniques*, pp. 197–211, 1994.

[LLPY97] F. Larsson, K. G. Larsen, P. Pettersson, and W. Yi. Efficient Verification of Real-Time Systems: Compact Data Structures and State-Space Reduction. In *Proc. of the 18th IEEE Real-Time Systems Symposium*, pp. 14–24, December 1997.

[LPY95] K. G. Larsen, P. Pettersson, and W. Yi. Compositional and Symbolic Model-Checking of Real-Time Systems. In *Proc. of the 16th IEEE Real-Time Systems Symposium*, pp. 76–87, December 1995.

[Ove81] W. Overman. *Verification of Concurrent Systems: Function and Timing*. PhD thesis, UCLA, Aug. 1981.

[Pag96] F. Pagani. Partial orders and verification of real-time systems. In *Proc. of Formal Techniques in Real-Time and Fault-Tolerant Systems*, vol. 1135 of *LNCS*, pp. 327–346. Springer Verlag, 1996.

[Pel93] D. Peled. All from one, one for all, on model-checking using representatives. In *Proc. of 5th Int. Conf. on Computer Aided Verification*, vol. 697 of *LNCS*, pp. 409–423. Springer Verlag, 1993.

[Val90] A. Valmari. Stubborn sets for reduced state space generation. In *Advances in Petri Nets*, vol. 483 of *LNCS*, pp. 491–515. Springer Verlag, 1990.

[Val93] A. Valmari. On-the-fly verification with stubborn sets. In *Proc. of 5th Int. Conf. on Computer Aided Verification*, vol. 697 of *LNCS*, pp. 59–70, 1993.

[YPD94] W. Yi, P. Pettersson, and M. Daniels. Automatic Verification of Real-Time Communicating Systems By Constraint-Solving. In *Proc. of the 7th International Conference on Formal Description Techniques*, 1994.

[YS97] T. Yoneda and H. Schlingloff. Efficient verification of parallel real-time systems. *Journal of Formal Methods in System Design*, 11(2):187–215, 1997.

[YSSC93] T. Yoneda, A. Shibayama, B.-H. Schlingloff, and E. M. Clarke. Efficient verification of parallel real-time systems. In *Proc. of 5th Int. Conf. on Computer Aided Verification*, vol. 697 of *LNCS*, pp. 321–332. Springer Verlag, 1993.