Adding Invariants to Event Zone Automata

Peter Niebert Hongyang Qu

Laboratoire d'Informatique Fondamentale de Marseille, Université de Provence
{niebert,hongyang}@cmi.univ-mrs.fr

Abstract. Recently, a new approach to the symbolic model checking of timed automata based on a partial order semantics was introduced, which relies on *event zones* that use vectors of event occurrences instead of *clock zones* that use vectors of clock values grouped in polyhedral clock constraints. Symbolic state exploration with event zones rather than clock zones can result in significant reductions in the number of symbolic states explored. In this work, we show how to extend the event zone approach to networks of automata with local state invariants, an important feature for modeling complex timed systems. To avoid formalizing local states, we attach to each transition an urgency constraint, that allows to code local state invariants. We have integrated the extension into a prototype tool with event zones and reported very promising experimental results.

1 Introduction

Timed automata [1] are a powerful tool for the modeling and the analysis of timed systems. They extend classical automata by *clocks*, continuous variables "measuring" the flow of time. A state of a timed automaton is a combination of its discrete control location and the *clock values* taken from the real domain. While the resulting state space is infinite, *clock constraints* have been introduced to reduce the state spaces to a finite set of equivalence classes, thus yielding a finite (although often huge) symbolic state graph on which reachability and some other verification problems can be resolved.

While the theory, algorithms [9, 10] and tools [12, 3] for timed automata represent a considerable achievement (and indeed impressing industrial applications have been treated), the combinatorial explosion particular to this kind of modeling and analysis – sometimes referred to as "clock explosion" (at the same time similar to and different from classical "state explosion") – remains a challenge for research and practice. Despite the theoretical limits (for a PSPACE complete problem), great effort has been invested into the optimization of the symbolic approach (see e.g. [8, 4, 7, 2]).

Event zone automata [11] are a partial order based approach to reduce one source of clock explosion, interleaving semantics. Partial order methods basically try to avoid redundant research by exploiting knowledge about the structure of the reachability graph, in particular *independence* of pairs of transitions of loosely related parts of a complex system. Such pairs a and b commute, i.e.

a state s allowing a sequence ab of transitions to state s' also allows ba and this sequence also leads to a state s'' that has the same control location as s'. However, this kind of commutation is easily lost in classical symbolic analysis algorithms for timed automata, which represent sets of possible clock values by symbolic states: Consider two "independent" actions a resetting clock $x := 0$, and b resetting clock $y := 0$. Executing a first and then b means that afterwards (time may have elapsed) $x \geq y$ whereas executing b first and then a implies that afterwards $x \leq y$. The result of this is that in the algorithms used in tools like UppAal [3] and Kronos [12], ab and ba lead to incomparable symbolic states.

The *event zone* approach successfully avoids such zone splitting, while preserving most algorithmic possibilities offered by clock zone automata. The underlying notion of independence is based on reading and writing of shared variables: If for some clock x, transition a resets x and transition b has a condition on x or if both a and b reset x, then they must be dependent.

However, being based on Mazurkiewicz trace theory and thus more action than state oriented, [11] did not address the question of state invariants (urgency constraints), an important modeling feature used in tools like UppAal. Unlike transition guards that give conditions on the interval in which a transition can be executed, state invariants are conditions attached to states (locations) that limit the allowed stay in the state, somewhat like a residence permit : before a violation of the invariant, a transition must be taken. Typically, allowed invariants are conjunctions of upper bounds on clock values.

In this work, we show how to add local state invariants to the event zone approach. A technical difficulty to overcome for this is the formalization of invariants: The event zone approach assumes only a single timed automaton with structural properties (diamonds) but does not as such expose "local states". It was not obvious, how state based invariants could be added to that framework. Our solution attaches urgency constraints to transitions, and then requires a consistency between these constraints and *global* state invariants. The urgency constraints allow a way of coding of local state invariants: One attach to a transition the local state invariant of its original automaton.

Then we show how to extend the framework of [11] to the timed automata with these invariants. An auxiliary tool, a "separator action" \$, in [11] is transformed into a "snapshot action": A snapshot is an artificial action that separates past and future and at which the global state invariant must be satisfied. For all other actions, the invariants to be satisfied are local: We only require satisfaction of its invariant whenever a clock is reset! It turns out that this approach allows to preserve completely the notion of independence from [11].

In this extended abstract, we concentrate on the formalization of the local invariants and the snapshot action and show the relevant properties. Moreover, we have extended the algorithm in [11] and added it to a new prototype tool POEM (Partial Order Environment of Marseille) based on the code of ELSE [13] to give an interesting (albeit not exhaustive) experimental comparison with a recent build of UppAal (v3.6 beta 1).

The paper is structured as follows: In Section 2, we informally explain the use and problems related to local state invariants. In Section 3, we formalize timed automata with invariants attached to transitions and we introduce the notion of a run and the language of a timed automaton. In Section 4, we develop the notion of independence in the context of the automata introduced in Section 3 and show the main fundamental results of this paper, the consistency of the relaxed semantics with the standard semantics. In Section 5, we give hints on how event zone based exploration tools for timed automata with invariants actually work. We give some experimental results and conclude in Section 6.

2 State invariants and reachability in timed automata

Many verification problems, notably *safety properties*, of timed automata can be coded as (non-)reachability of a location (discrete state), and we concentrate on such specifications in this section. This does not exclude the extension of our ideas to more sophisticated properties like emptiness of Büchi automata.

A state invariant is a downward closed condition on the clocks, a conjunction of constraints like "$x < 5$" or "$y \leq 7$" or even "$z \leq 0$", stating, how long at most the "stay" in a state is "allowed", i.e. the stay in the state must not extend to a moment where the invariant is violated. In a run, a transition must thus occur before the expiry of the state invariant.

For reachability in timed automata, state invariants do not fundamentally add expressive power! We can eliminate invariants without touching the set of reachable states by two transformation steps:

- Strengthen the conditions of each outgoing transition by the invariant without changing the possible behaviour of the automaton; likewise strengthen the conditions so to guarantee that the invariants of the target states are respected at entry.
- Remove the invariants from the states after having strengthened the transitions. While this change may allow partial runs where the automaton stays beyond invariant in a state, it cannot add to the reachable states.

However, the situation changes when we consider the use of invariants in modelling *networks of timed automata*. Consider the following timed system in Figure 1: A multi lane highway with cars on each lane and a rabbit who wants to cross. The rabbit has some freedom of going slower or faster and so do the cars. Can - with the help of the car drivers - the rabbit reach the other side of the highway alive? To model this by a network of timed automata, we choose to model the highway as a checker board of lanes and positions on lanes as indicated in the picture, cars move in the horizontal direction and the rabbit in the vertical direction. Each car and the rabbit is realised by an individual automaton. The freedom of going slower or faster is modeled by a time interval in which the rabbit can advance by one lane and an interval in which the car can advance for one unit length on a discretized highway. If a car and the rabbit are in the same field of the checker board at the same time, an accident occurs.

The UppAal model is indicated by showing the automaton for the rabbit and an instance of the automata for cars.

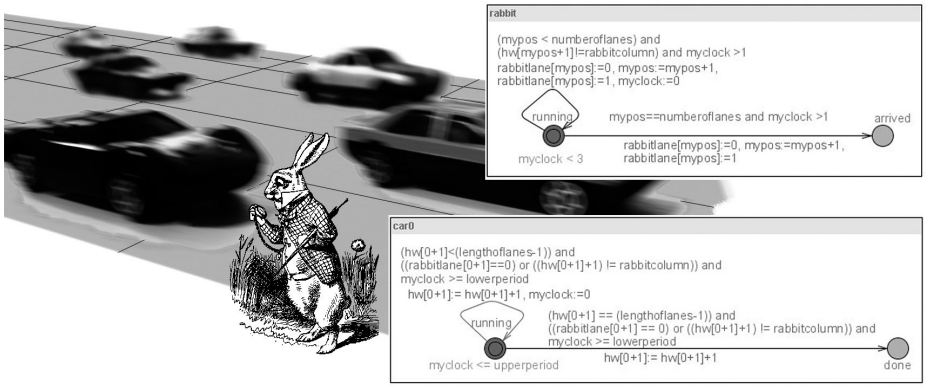

Fig. 1. A real life race condition and its UppAal model with invariants

Without invariants, the cars or the rabbit could just stop (not take a advancing transition), in which case it is obvious that the rabbit will reach its target safely. Hence, the invariants are essential for correct modeling in that the invariants enforce a more or less synchronous progress of the cars and the rabbit, so that everyone has to choose their speed to allow the rabbit to pass unharmed.

A naïve approach to eliminate the invariants is to apply the two step transformation to automata (on-the-fly or offline) and after that have an Alur-Dill automaton without invariants to which we could apply the algorithm shown in [11]. However, the global invariant limits the time progress for the local clock of each car, thus the rewriting will essentially render all transitions dependent! This is precisely a source of combinatorial explosion in this example.

But this additional dependency is not unavoidable, the "independence" of the cars in the example is quite obvious: They don't interfere with each other, they just interfere with the rabbit. In the rest of the article we show that there is a better way of dealing with local invariants, preserving full independence.

3 Timed automata with transition invariants

In this section, we introduce basic notions of timed words, timed languages, as well as their finite representation by timed automata [1].

For an alphabet Σ of actions denoted by $a, b, c \ldots$, let $\$ \notin \Sigma$ be a special symbol, the *snapshot action*[1], and let $\Sigma_\$:= \Sigma \uplus \{\$\}$ denote the extension of Σ by \$. Σ^* (or $\Sigma_\$^*$) is the set of finite sequences $a_1 \ldots a_n$ called words, with ϵ the empty word. The length n of a word $a_1 \ldots a_n$ is denoted by $|a_1 \ldots a_n|$. A *timed word* is a sequence $(a_1, \tau_1) \ldots (a_n, \tau_n)$ of elements in $(\Sigma_\$ \times \mathbb{R}^+)^*$, with $\mathbb{R}^+$ the set of non-negative reals, the τ_i's are *time stamps*. For convenience, we set $\tau_0 = 0$

[1] the rôle of which will become clear later.

to be an additional time stamp for the beginning. A timed word is *normal* if $\tau_i \leq \tau_j$ for $i \leq j$ as in $(a, 3.2)(c, 4.5)(b, 6.3)$ whereas $(a, 3.2)(c, 2.5)(b, 6.3)$ is not normal. Normal timed words represent temporally ordered sequences of events and serve as standard semantics of timed automata in the literature.

In timed systems, events can occur only if certain time constraints are satisfied. In timed automata, a finite set of real valued[2] variables X, called *clocks*, are used to express the time constraints between an event that resets a clock and another event that refers to the clock value at the time of its occurrence. The clock constraints permitted here are conjunctions of *atomic clock constraints*, comparisons between a clock and a numerical constant. To preserve decidability, constants are assumed to be positive rationals and for simplicity in $\mathbb{N}$, the set of natural numbers. For a set of clocks X, the set $\Phi(X)$ of clock constraints ϕ is formally defined by the grammar $\phi := \text{true} | x \bowtie c \ | \phi_1 \wedge \phi_2$, where x is a clock in X, $\bowtie \in \{<, \leq, >, \geq\}$ and c is a constant in $\mathbb{N}$ (true is for transitions without conditions), moreover excluding the trivially false combination "< 0". Another way of looking at clock constraints is sets of atomic constraints that must all be satisfied. A subset of clock constraints result from the restriction to upper bounds $\lhd \in \{<, \leq\}$, so let $\Phi_{upper}(X) \subseteq \Phi(X)$ be defined according to the grammar $\phi := \text{true} | x \lhd c \ | \phi_1 \wedge \phi_2$.

A *clock valuation* $v : X \to \mathbb{R}$ is a function that assigns a real number to each clock. We denote by $v + \tau$ the clock valuation that translates all clock $x \in X$ synchronously by τ such that $(v + \tau)(x) = v(x) + \tau$. For a subset C of clocks, $v[C \leftarrow 0]$ denotes the clock valuation with $v[C \leftarrow 0](x) = 0$ if $x \in C$ and $v[C \leftarrow 0](x) = v(x)$ if $x \notin C$, i.e. the valuation where the clocks in C are reset to 0. The satisfaction of the clock constraint ϕ by the clock valuation v, i.e. the fact that all atomic constraints are satisfied when substituting $v(x)$ for x, is denoted by $v \vDash \phi$.

Definition 1. *Given an alphabet $\Sigma_\$$ and a set of clocks X, a* timed automaton with transition invariants *is a quintuple $\mathcal{A} = (\Sigma_\$, S, s_0, \rightarrow, Inv, F)$ where S is a finite set of locations, $s_0 \in S$ is the initial location, $F \subseteq S$ is the set of final locations and $\rightarrow \subseteq S \times [\Sigma_\$ \times \Phi(X) \times \Phi_{upper}(X) \times 2^X] \times S$ is a set of transitions, $Inv : S \longrightarrow \Phi_{upper}(X)$ is an assignment of* clock invariants *to locations. For a transition $(s, a, \phi, \psi, C, s') \in \rightarrow$, we write $s \xrightarrow{(a,\phi,\psi,C)} s'$, and call a the* label *of the transition, and ψ the invariant of the transition or the transition invariant.*

A state *(s, v) consists of a location s and a clock valuation v. The invariant of location s is also called the state invariant. For a network of automata, the invariant of a global state, i.e.* global state invariant*, is the conjunction of the invariants of local states, i.e.* local state invariants*. We take local state invariants, instead of global state invariants, as transition invariants in the product automaton of the network, in order to be able to apply partial order reduction to verification. The only exception is the snapshot action, which is a single (self-loop) transition $(s, \$, true, Inv(s), \emptyset, s)$ at each $s \in S$. Here s is a global location*

[2] For normal timed words *positive real values* would suffice, see Remark 2.

and $Inv(s)$ is the global state invariant. In this sense, we say that the snapshot action verifies the global state invariant.

For a conjunction of constraints ϕ, let $\phi(x)$ denote the restriction of ϕ to constraints concerning the clock x. For convenience, we consider only a network of automata, which satisfy for all transitions $(s, a, \phi, \psi, C, s')$ that

1. *$Inv(s) \Rightarrow \psi$, i.e. the global state invariant implies the transition invariant;*
2. *For all $x \notin C$ we have $Inv(s)(x) \wedge \phi(x) \Longrightarrow Inv(s')(x)$;*
3. *For all $x \notin C$ we have $Inv(s')(x) \wedge \psi(x) \Longrightarrow Inv(s)(x)$.*

The intuition of the invariant constraint ψ of a transition is to result from the local state invariant of the predecessor state of one automaton in a network. Hence, the first condition says that the global state invariant is (at least as strong as) the conjunction of the invariants of the outgoing transitions. Indeed, it is the case according to the definition of the global state invariant.

Remark 1. A timed automaton is *action deterministic* if for two transitions $s \xrightarrow{(a,\phi_1,\psi_1,C_1)} s_1$ and $s \xrightarrow{(a,\phi_2,\psi_2,C_2)} s_2$, we have that $\phi_1 = \phi_2$, $\psi_1 = \psi_2$, $C_1 = C_2$ and $s_1 = s_2$. Similarly, we call the timed automaton *constraint consistent* if actions determine uniquely clock constraints and resets, i.e. for each pair of transitions $(s_1, a, \phi, \psi, C, s_2)$ and $(s_1', a, \phi', \psi, C', s_2')$ with the same action, we have $\phi = \phi'$, $\psi = \psi'$ and $C = C'$. In that case, given an action a, the unique clock constraint, invariant and reset are denoted by ϕ_a, ψ_a and C_a respectively. In this paper, we will *only consider timed automata that are action deterministic and constraint consistent*. This is no restriction to applications, as this can be easily achieved by renaming (a more detailed discussion is in [11]).

Figure 2 (produced by UppAal) shows a system consisting of two automata. Initial locations are s0 and w0, and final locations are s0, s2, s3 and w0. Clocks are x, y, and z. States invariants are labeled in boldface, e.g., "y<=9" is the variant for local state s1 and "y<=9 ∧ z<=4" is the global variant for (s1,w1).

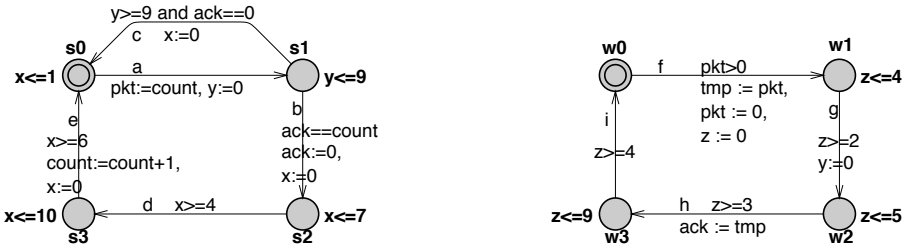

Fig. 2. The example

For our formal development, we introduce three distinct notions of sequences of execution: *paths* (ignoring time constraints), *runs* (paths with time stamps respecting the time constraints), *normal runs* (furthermore the time stamps respect the progress of time):

Definition 2. *A* path *in $\mathcal{A}$ is a finite sequence $s_0 \xrightarrow{(a_1,\phi_1,\psi_1,C_1)} s_1 \ldots \xrightarrow{(a_n,\phi_n,\psi_n,C_n)} s_n$ of consecutive transitions $s_{i-1} \xrightarrow{(a_i,\phi_i,\psi_i,C_i)} s_i$. The word $a_1 \ldots a_n \in \Sigma_{\$}^*$ of transition labels is called the* path labeling. *If $a_n = \$$ (final snapshot) and $s_n \in F$, the path is said to be* accepted. *The set of labelings of accepted paths is called the* untimed language *of $\mathcal{A}$ and denoted $L(\mathcal{A})$.*

Definition 3. *A* run *of a timed automaton is a path extended by time stamps for the transition occurrences satisfying clock constraints and resets:*

$(s_0, v_0) \xrightarrow{(a_1,\phi_1,\psi_1,C_1),\tau_1} (s_1, v_1) \ldots \xrightarrow{(a_n,\phi_n,\psi_n,C_n),\tau_n} (s_n, v_n)$ *where $(a_1, \tau_1) \ldots (a_n, \tau_n)$ is a timed word and $(v_i)_{0 \le i \le n}$ are clock valuations defined by:*

1. *$v_0(x) = 0$ for all $x \in X$, $\tau_0 = 0$*
2. *$v_{i-1} + (\tau_i - \tau_{i-1}) \vDash \phi_i \wedge \psi_i$*
3. *$v_{i-1} + (\tau_i - \tau_{i-1}) \vDash Inv(s_{i-1})(x)$ for $x \in C_i$*
4. *$v_i = (v_{i-1} + (\tau_i - \tau_{i-1}))[C_i \leftarrow 0]$.*

Note that the above conditions imply for $a_i = \$$ that $v_i \vDash Inv(s_i)$, since $s_i = s_{i-1}$. The timed word $(a_1, \tau_1) \ldots (a_n, \tau_n)$ is the timed labeling *of the run. The run is* accepted by $\mathcal{A}$ *if $a_n = \$$, $s_n \in F$.*

Definition 4. *A* normal run *is a run such that its timed labeling $(a_1,\tau_1) \ldots (a_n,\tau_n)$ is a normal timed word i.e. $\tau_1 \le \tau_2 \le \ldots \le \tau_n$.*

Remark 2. It is straightforward to see that for normal runs the valuations always produce positive values: Clocks are either reset to 0 or the translations $v + (\tau_i - \tau_{i-1})$ increase the values since $\tau_i \ge \tau_{i-1}$. In non-normal runs, this need not be the case.

Proposition 1. *In a normal accepted run it holds for all intermediate states s_i that $v_i + (\tau_{i+1} - \tau_i) \vDash Inv(s_i)$, i.e. global invariants are never violated.*

Proof. The proof is by induction on $n-k$, i.e. on the distance from the last state in the sequence.

The basis is to prove that the proposition holds for $k = 1$. Since $a_n = \$$, $s_n = s_{n-1}$ and $\psi_n = Inv(s_n) = Inv(s_{n-1})$. The second condition implies in turn that $v_{n-1} + (\tau_n - \tau_{n-1}) \vDash \psi_n$.

For the induction step, assume that for $k \ge 1$, the proposition is true, i.e., $v_{n-k} + (\tau_{n-k+1} - \tau_{n-k}) \vDash Inv(s_{n-k})$. For the case $k = k + 1$, $v_{n-k-1} + (\tau_{n-k} - \tau_{n-k-1}) \vDash \phi_{n-k} \wedge \psi_{n-k}$ and $v_{n-k-1} + (\tau_{n-k} - \tau_{n-k-1}) \vDash Inv(s_{n-k-1})(x)$ if $x \in C_{n-k-1}$. For $y \notin C_{n-k-1}$, $v_{n-k-1} + (\tau_{n-k} - \tau_{n-k-1}) \vDash Inv(s_{n-k})(y) \wedge \psi_{n-k}$ implies $v_{n-k-1} + (\tau_{n-k} - \tau_{n-k-1}) \vDash Inv(s_{n-k-1})(y)$. Therefore, $v_{n-k-1} + (\tau_{n-k} - \tau_{n-k-1}) \vDash Inv(s_{n-k-1})$, which means the proposition holds for $k + 1$. □

The *timed language* $L_T(\mathcal{A})$ of $\mathcal{A}$ is the set of *normal* timed words $(a_1,\tau_1) \ldots (a_n,\tau_n)$, such that $(a_1, \tau_1) \ldots (a_n, \tau_n)(\$, \tau)$ with $\tau \ge \tau_i$ is the labeling of a (normal) run accepted by $\mathcal{A}$. The path labeling $a_1 \ldots a_n$ is said to be *realizable* if for some time stamps τ_i the normal timed word $(a_1, \tau_1) \ldots (a_n, \tau_n)(\$, \tau)$ (then called the

normal realization of $a_1 \dots a_n$) is the labeling of a normal run. The language of realizable words that are the labeling of an *accepted run* is denoted by $L_N(\mathcal{A})$.

For instance, in Figure 2 (a,0.5)(f,2.5)(g,4.6)(h,7)(i,9.3)(b,9.4)$\in L_T(\mathcal{A})$ is a normal realization of the path labeling afghib, hence afghib$\in L_N(\mathcal{A})$.

4 Independence for Timed Automata

To model concurrency, we use an independence relation between actions such that actions are independent when the order of their occurrence is irrelevant. Formally, an *independence relation* I for an (action deterministic and constraint consistent) timed automaton $\mathcal{A} = (\Sigma_\$, S, s_0, \rightarrow, Inv, F)$ is a symmetric and irreflexive relation $I \subseteq \Sigma \times \Sigma$ such that the following two properties hold for any two $a, b \in \Sigma$ with $a\ I\ b$:

(i) $s \xrightarrow{(a,\phi_a,\psi_a,C_a)} s_1 \xrightarrow{(b,\phi_b,\psi_b,C_b)} s_2$ implies $s \xrightarrow{(b,\phi_b,\psi_b,C_b)} s'_1 \xrightarrow{(a,\phi_a,\psi_a,C_a)} s_2$ for some location s'_1

(ii) $C_a \cap C_b = \emptyset$ and no clock x in C_b belongs to an atomic clock constraint $x \bowtie c$ of $\phi_a \wedge \psi_a$ and conversely no clock x in C_a belongs to an atomic clock constraint $x \bowtie c$ of $\phi_b \wedge \psi_b$.

We also use the dependence relation $D = \Sigma \times \Sigma - I$, which is reflexive and symmetric.

We extend the independence relation to $\Sigma_\$$ by setting $\$Db$ for all $b \in \Sigma_\$$, i.e. we *define* the snapshot to be dependent of every other action. This obviously meets conditions (i) and (ii).

Intuitively, condition (ii) arises from the view of clocks as shared variables in concurrent programming: An action resetting a clock is writing it whereas an action with a clock constraint on this clock is reading it. The restriction states that two actions are dependent if both are writing the same variable, or one is writing a variable and the other one is reading it.

Since $I = \emptyset$ trivially meets (i) and (ii) such a relation always exists. Computing a good (the larger, the better) I meeting (i) and (ii) is a matter of static analysis and is typically done on the level of a network *before* constructing the product timed automaton: Sufficient criteria for (i) may require that two transitions originate from distinct components and do not have conflicts around shared variables and do not synchronize on the same channels. For instance, (b,f) is in the independence relation for the timed automata of Figure 2, while (b,g) in the dependence relation.

The *Mazurkiewicz trace equivalence* associated to the independence relation I is the least congruence $\simeq$ over Σ^* such that $ab \simeq ba$ for any pair of independent actions $a\ I\ b$. A *trace* $[u]$ is the congruence class of a word $u \in \Sigma^*$. By definition, two words are equivalent with respect to $\simeq$ if they can be obtained from each other by a finite number of exchanges of adjacent independent actions. E.g., acf $\simeq$ afc for Figure 2, but acf $\not\simeq$ fac (a and f are dependent). In other words, this permutation of actions between two equivalent words lets the relative order of occurrences of dependent actions unchanged, formally:

Lemma 1. *Let I be an independence relation, $\simeq$ the induced Mazurkiewicz trace equivalence and $a_1 \dots a_n \simeq b_1 \dots b_n$ be two equivalent words. There exists a uniquely determined permutation $\pi : \{1, \dots, n\} \to \{1, \dots, n\}$ such that $a_i = b_{\pi(i)}$ and for $a_i \ D \ a_j$ we have $i < j$ iff $\pi(i) < \pi(j)$.*

Conversely, let $a_1 \dots a_n$ be a word and $\pi : \{1, \dots, n\} \to \{1, \dots, n\}$ be a permutation of indices such that for each pair i, j $a_i \ D \ a_j$ we have $i < j$ iff $\pi(i) < \pi(j)$. Then $a_{\pi(1)} \dots a_{\pi(n)} \simeq a_1 \dots a_n$.

Proof. By induction on the number of exchanges. □

For convenience in applications to timed words, we assume π to be extended to 0 with $\pi(0) = 0$.

The untimed language $L(\mathcal{A})$ of a timed automaton $\mathcal{A}$ is closed under the equivalence $\simeq$ and this is the theoretical foundation of many partial order reduction approaches. For instance, reductions that preserve at least one representative for each equivalence class do preserve non-emptiness of the untimed languages. Moreover the equivalence relation extends to runs when disregarding normality constraints:

Lemma 2. *Let $(a_1, \tau_1) \dots (a_n, \tau_n)$ be the timed labeling of a run,*
$\pi : \{1, \dots, n\} \to \{1, \dots, n\}$ be a permutation with $a_1 \dots a_n \simeq a_{\pi(1)} \dots a_{\pi(n)}$. Then $(a_{\pi(1)}, \tau_{\pi(1)}) \dots (a_{\pi(n)}, \tau_{\pi(n)})$ is also a timed labeling of a run.

Proof. The proof is by induction of the number of exchanges in π, it is sufficient to consider the case of a single exchange.

Let $(a_1, \tau_1) \dots (a_k, \tau_k)(a, \tau_{k+1})(b, \tau_{k+2})(a_{k+3}, \tau_{k+3}) \dots (a_n, \tau_n)$ be the time labeling where $a \ I \ b$ and let $r = (s_0, v_0) \dots (s_n, v_n)$ be the corresponding run. Assume that $s_k \xrightarrow{(a,\phi_a,\psi_a,C_a)} s_{k+1} \xrightarrow{(b,\phi_b,\psi_b,C_b)} s_{k+2}$.

We prove the existence of a unique run $r' = (s'_0, v'_0) \dots (s'_n, v'_n)$ with timed labeling $(a_1, \tau_1) \dots (a_k, \tau_k)(b, \tau_{k+2})(a, \tau_{k+1})(a_{k+3}, \tau_{k+3}) \dots (a_n, \tau_n)$ such that $s'_i = s_i$ for $i \neq k+1$ and $v'_i = v_i$ for $i \notin \{k+1, k+2\}$.

By property (i) of I, $s_k \xrightarrow{(b,\phi_b,\psi_b,C_b)} s'_{k+1} \xrightarrow{(a,\phi_a,\psi_a,C_a)} s_{k+2}$ and all other transitions are unchanged hence $s'_i = s_i$ for $i \neq k+1$.

The sequence r' is a run if the time valuations v'_i satisfy the constraints. We consider two cases:

(1) $i \leq k$ or $i > k+3$. The result holds since r is a run.

(2) $i = k+1$,$i = k+2$ and $i = k+3$.

First observe that since r is a run, we have $v_{k+1} + (\tau_{k+2} - \tau_{k+1}) \vDash \phi_b \wedge \psi_b$: By condition (ii) of independence, no clock mentioned in $\phi_b \wedge \psi_b$ is reset in C_a hence $(v_{k+1} + (\tau_{k+2} - \tau_{k+1}))(x) = v_{k+1}(x) + (\tau_{k+2} - \tau_{k+1}) = (v_k + (\tau_{k+1} - \tau_k))(x) + (\tau_{k+2} - \tau_{k+1}) = (v_k + (\tau_{k+2} - \tau_k))(x)$ for any clock x mentioned in ϕ_b.

Therefore $v_{k+1} + (\tau_{k+2} - \tau_{k+1}) \vDash \phi_b \wedge \psi_b$ iff $v_k + (\tau_{k+2} - \tau_k) \vDash \phi_b \wedge \psi_b$.

Second, for a clock $x \in C_b$ we have $v_{k+1} + (\tau_{k+2} - \tau_{k+1}) \vDash Inv(s_{k+1})(x)$. Then, since due to independence we know that $x \notin C_a$, again $(v_{k+1} + (\tau_{k+2} - \tau_{k+1}))(x) = (v_k + (\tau_{k+2} - \tau_k))(x)$. Therefore the transition $s_k \xrightarrow{(b,\phi_b,\psi_b,C_b)} s'_{k+1}$

is enabled at τ_{k+2} yielding $(s'_{k+1}, v'_{k+1}) \vDash Inv(s_{k+1})(x)$. But $Inv(s_{k+1})(x) \wedge \psi_a(x) \Longrightarrow Inv(s_k)(x)$ and ψ_a does not restrict x (due to independence), hence $Inv(s_{k+1})(x) \Longrightarrow Inv(s_k)(x)$ and we obtain $v_k + (\tau_{k+2} - \tau_k) \vDash Inv(s_k)(x)$. We therefore obtain that r' is a run up to (s'_{k+1}, v'_{k+1}).

Similarly the transition $s'_{k+1} \xrightarrow{(a,\phi_a,\psi_a,C_a)} s_{k+2}$ satisfies the conditions on $v'_{k+1} + (\tau_{k+1} - \tau_{k+2})$: For $x \notin C_b$ we have $(v'_{k+1} + (\tau_{k+1} - \tau_{k+2}))(x) = (v_k + (\tau_{k+1} - \tau_k))(x)$, hence $v'_{k+1} + (\tau_{k+1} - \tau_{k+2}) \vDash \phi_a \wedge \psi_a$. If, on the other hand, $x \in C_a$, then $x \notin C_b$ and we see that $v'_{k+1} + (\tau_{k+1} - \tau_{k+2}) \vDash Inv(s_k)(x)$. But $Inv(s_k)(x) \wedge \phi_b \wedge \psi_b \Longrightarrow Inv(s'_{k+1})(x)$ and since $\phi_b \wedge \psi_b$ do not constrain x due to independence, we obtain that $Inv(s_k)(x) \Longrightarrow Inv(s'_{k+1})(x)$ and hence $v'_{k+1} + (\tau_{k+1} - \tau_{k+2}) \vDash Inv(s'_{k+1})(x)$.

Finally, since $C_a \cap C_b = \emptyset$ we get $v'_{k+2} = v_{k+2} + (\tau_{k+1} - \tau_{k+2})$ which implies $v'_{k+2} + (\tau_{k+3} - \tau_{k+1}) = v_{k+2} + (\tau_{k+3} - \tau_{k+2})$. This guarantees that the transition corresponding to a_{k+3} is still possible at τ_{k+3} and that $v'_{k+3} = v_{k+3}$. □

However, Lemma 2 only claims commutability of runs without taking time progress into account. For the timed language $L_T(\mathcal{A})$ and consequently for $L_N(\mathcal{A})$, the normality condition may exclude some representatives in a trace:

Let afcgahib and acfgahib be two equivalent paths of Figure 2. It is easy to know that the former one is in $L_N(\mathcal{A})$, while the latter not because the constraint $(\tau_5 - \tau_2 \leq 1) \wedge (2 \leq \tau_4 - \tau_3 \leq 4) \wedge (\tau_2 \leq \tau_3) \wedge (\tau_4 \leq \tau_5)$ cannot be satisfied by a normal word (a,τ_1)(c,τ_2)(f,τ_3)(g,τ_4)(a,τ_5)... Therefore we introduce a weaker notion of normality:

A timed word $(a_1, \tau_1) \ldots (a_n, \tau_n)$ is *I-normal* iff for any two letters a_i, a_j with $i \leq j$ *and additionally* $a_i \; D \; a_j$ we have $\tau_i \leq \tau_j$. In Figure 2, the timed word (a,0.5)(c,9.5)(f,5.7)(g,9.6)(a,10.1) is I-normal. The intuition behind this relaxation of constraints is that in practice, actions are dependent if they are executed by the same component in a network of timed automata. This non-decreasing condition on action occurrences models the sequential behavior of each component. In [6], this is modeled by considering a local time for each component. The interaction between components leads to the propagation of time progress to other components (formally due to dependency).

In analogy to realisable words, we say that $a_1 \ldots a_n$ is *I-realisable* iff it is the labelling of a run $(s_0, v_0) \xrightarrow{(a_1,\phi_{a_1},C_{a_1}),\tau_1} (s_1, v_1) \ldots \xrightarrow{(a_n,\phi_{a_n},C_{a_n}),\tau_n} (s_n, v_n)$ in $\mathcal{A}$ such that $(a_1, \tau_1) \ldots (a_n, \tau_n)$ is I-normal. As for L_N, let $L_I(\mathcal{A})$ denote the set of I-realisable words $a_1 \ldots a_n$ such that $a_1 \ldots a_n\$$ is the labelling of an *accepted run* (i.e. $s_n \in F$ and in particular $v_n \vDash Inv(s_n)$). For instance, afghb is I-realisable in Figure 2 as time stamps $0.5, 2.5, 4.6, 7, 9.4, 9.3$ satisfy clock constraints of transitions from (s0,w0) to (s2,w0) and (a,0.5)(f,2.5)(g,4.6)(h,7)(b,9.4)(i,9.3) is I-normal. Moreover, afghbi is also in $L_I(\mathcal{A})$ since (s2,w0) is final.

Obviously $L_N(\mathcal{A}) \subseteq L_I(\mathcal{A})$.

By definition $L_T(\mathcal{A}) = \emptyset$ if and only if $L_N(\mathcal{A}) = \emptyset$. Moreover, the following proposition implies that $L_N(\mathcal{A}) = \emptyset$ iff $L_I(\mathcal{A}) = \emptyset$, so that we can check this emptiness problem equivalently for either language.

Proposition 2. *For every I-normal labelling $(a_1, \tau_1) \dots (a_n, \tau_n)$ of a run in an action deterministic, constraint consistent timed automaton $\mathcal{A}$, there exists $(a_{\pi(1)}, \tau_{\pi(1)}) \dots (a_{\pi(n)}, \tau_{\pi(n)})$ an equivalent normal labelling of an (equivalent) run in $\mathcal{A}$, where π is a permutation as defined in Lemma 1.*

Proof. Consider the following ordering on $\{1, \dots, n\}$: $i \sqsubset j$ iff $\tau_i < \tau_j$ or $\tau_i = \tau_j$ and $i < j$. There is a unique permutation such that $i \sqsubset j$ iff $\pi(i) < \pi(j)$. Moreover, for $a_i D a_j$ and $i < j$, I-normality implies that $\tau_i \leq \tau_j$ and finally $\pi(i) < \pi(j)$, i.e. π yields an equivalent path. By Lemma 2, $(a_{\pi(1)}, \tau_{\pi(1)}) \dots (a_{\pi(n)}, \tau_{\pi(n)})$ is thus a timed labelling of some run and by the construction of $\sqsubset$ it is a normal timed word. □

A sorting algorithm provides an efficient way of computing a normal timed labelling of a run from an I-normal labelling.

A key main feature of $L_I(\mathcal{A})$ is the closure under equivalence that is stated in Theorem 1. In principle this allows to limit exploration of realisable clocked words to representatives of equivalence class:

Theorem 1. *(1) Let $u \simeq v$ and $u \in L_I(\mathcal{A})$ then $v \in L_I(\mathcal{A})$.*
(2) $L_I(\mathcal{A}) = \{u \mid \exists v \simeq u : \ v \in L_N(\mathcal{A})\}$.

Proof. (1) Let $u = a_1 \dots a_n$, $v = b_1 \dots b_n$ and π be the permutation linking $a_1 \dots a_n$ and $b_1 \dots b_n$ according to Lemma 1. Let $(a_1, \tau_1) \dots (a_n, \tau_n)$ an I-normal labelling of some accepting run of $\mathcal{A}$. Then $(b_1, \tau_{\pi(1)}) \dots (b_n, \tau_{\pi(n)})$ is a timed labelling of some accepting run according to Lemma 2 and it inherits I-normality since π preserves the order of occurrences of dependent actions.

(2) "$\supseteq$" follows from $L_N(\mathcal{A}) \subseteq L_I(\mathcal{A})$ (normality implies I-normality) and reflexivity of $\simeq$. "$\subseteq$" is an easy consequence of Proposition 2. □

5 Symbolic analysis for L_I

The goal of this section is to show how the results of the previous section can be used for reachability analysis. A full description would require a lot of space and we refer the reader to [11] for an exhaustive treatment without invariants. Instead, here we concentrate on one aspect, the constraints required to check whether a word belongs to $L_I(A)$.

Let $\mathbb{T} = \{t_0, t_1, \dots\}$ be a set of time stamp variables. An atomic time constraint is a time constraint of the form $t_i - t_j \prec c$, a general constraint a conjunction of atomic constraints, where t_i, t_j are time stamp variables in $\mathbb{T}$, $\prec \in \{<, \leq\}$ and c is a constant in $\mathbb{Z}$. An *interpretation* of a time stamp constraint φ is a function $v : \mathbb{T} \to \mathbb{R}^+$ assigning a non-negative real number τ_i to each time stamp variable t_i. The satisfaction of φ by v is denoted $v \vDash \varphi$ and in that case v is a *model* for φ. We call φ *consistent* iff it has a model otherwise *inconsistent*. As is well known, consistency and a model can be determined with the Bellman-Ford shortest path algorithm.

To express I-realizability in terms of time stamp constraints we need to define special positions in a path. Given a path labeling $a_1 \dots a_n$ we define

$last_a(a_1 \dots a_n)$, the last occurrence of a, to be the maximal k such that $a_k = a$, if such a k exists, otherwise $last_a(a_1 \dots a_n) = 0$. Similarly, we define $last_x(a_1 \dots a_n)$ to be the maximal position k at which x is reset, that is $x \in C_{a_k}$, if such a position exists, otherwise $last_x(a_1 \dots a_n) = 0$ (every clock is reset at the beginning).

With these positions we express that in a word dependent actions are ordered according to their order of occurrence (condition (1) in the following) and that clock constraints are satisfied (conditions (2) (3) and (4)) allowing to check I-realizability on the level of consistency:

For a timed automaton $\mathcal{A}$ and a path labelling $a_1 \dots a_n$ let $\varphi_{a_1 \dots a_n}$ be the *associated time stamp constraint* which is the conjunction of the time stamp constraints satisfying one of four cases :

1. $t_i - t_j \leq 0$ with $i < j$ and $a_i \; D \; a_j$ and $i = last_{a_i}(a_1 \dots a_{j-1})$;
2. $t_j - t_i \prec c$ with $x \prec c$ in ϕ_j, ψ_j, and $i = last_x(a_1 \dots a_{j-1})$
3. $t_i - t_j \prec -c$ with $x \succ c$ in ϕ_j, and $i = last_x(a_1 \dots a_{j-1})$
4. $t_j - t_i \prec c$ with $x \in C_j$ and $x \prec c = Inv(\sigma(a_1 \dots a_{j-1}))(x)$ and $i = last_x(a_1 \dots a_{j-1})$

Proposition 3. *Let $\mathcal{A} = (\Sigma, S, s_0, \rightarrow, F)$ be a deterministic timed automaton and I an independence relation. Moreover, let $a_1 \dots a_n$ be a path labeling of $\mathcal{A}$. The word $a_1 \dots a_n$ is I-realizable iff its associated time stamp constraint $\varphi_{a_1 \dots a_n}$ is consistent.*

In [11], *event zones* were introduced as an incremental way of computing consistency of time stamp constraints: An event zone is a triple $Z = (T, \varphi, Last)$ where T is a set of time stamp variables, φ is a time stamp constraint and $Last : X \cup \Sigma \rightarrow T$ is the *last occurrence function* that assigns to a clock or an action a the time stamps that represents respectively its last reset and the last occurrence of the action. Formally, the event zone $Z_u = (T_u, \varphi_u, Last_u)$ of the path labeling $u = a_1 \dots a_n$ is given by $T_u = \{t_0, \dots, t_n\}$, where φ_u is the time stamp constraint associated to u and $Last_u(a) = t_i$ with $i = last_a(a_1 \dots a_n)$ for all action a, $Last_u(x) = t_i$ with $i = last_x(a_1 \dots a_n)$ for all clock x.

To obtain a symbolic automaton, we consider pairs (s, Z) with s a location from the original timed automaton and Z an event zone, *symbolic states*. Transitions on symbolic states are obtained by *extension* $(s_1, Z_1) \circ a := (s_2, Z_2) = (s_2, (T_2, \varphi_2, Last_2))$ of a symbolic state $(s_1, Z_1) = (s_1, (T_1, \varphi_1, Last_1)$ by an action a is defined if there exists a transition $s_1 \xrightarrow{a, \phi_a, \psi_a, C_a} s_2$, such that $T_2 = T_1 \uplus \{t\}$ with t a fresh time stamp variable not in T_1, and φ_2 is the *consistent* conjunction of φ_1 and

- $t_i - t \leq 0$ for all $t_i = Last(b)$ for b such that $a \; D \; b$,
- $t - t_i \prec c$ with $x \prec c$ in ϕ_a, ψ_a, and $t_i = Last(x)$,
- $t_i - t \prec -c$ with $x \succ c$ in ϕ_a and $t_i = Last(x)$,
- $t - t_i \prec c$ with $x \prec c = Inv(s_1)(x)$ iff $x \in C_a$,

and finally $Last_2$ is such that $Last_2(\alpha) = t$ for α a clock in C_a or $\alpha = a$ otherwise $Last_2(\alpha) = Last_1(\alpha)$.

Following the lines of [11], it is then possible to define an equivalence relation $\simeq_{EZ}$ compatible with the extension, such that if $(s_1, Z_1) \simeq_{EZ} (s_2, Z_2)$ then $(s_1, Z_1) \odot a \simeq_{EZ} (s_2, Z_2) \odot a$. This equivalence is essentially "same constraint up to pointer renaming". It turns out that the independence relation is compatible with $\simeq_{EZ}$, i.e. for $a\ I\ b$ we have $(s_1, Z_1) \odot a \odot b \simeq_{EZ} (s_1, Z_1) \odot b \odot a$, the fundamental reason why event zones reduce the number of symbolic states explored. The equivalence classes are the symbolic states of the event zone automaton, which thus itself respects the independence relation.

The interesting aspect of event zones is that they allow to abstract from time stamps that are not referenced by *Last*. More precisely, the constraint of the event zone is *closed* using the Floyd-Warshall algorithm and then the time stamps not referenced by pointers can be projected away. This allows to limit the dimensions of event zones to the number of pointers (here: clocks and the size of the alphabet). In practice, we use an optimized set of pointers to further reduce the dimensions, which moreover results in event zones for \$ terminated paths that never have more dimensions than the number of clocks plus one. This corresponds to the dimension of classical clock zones.

The snapshot action \$ obtains a special rôle in the state exploration with event zones: When we reach a final state (desired property) with a symbolic state (s, Z), we evaluate the consistency of $(s, Z) \odot \$$ according to Definition 3. We also eliminate intermediate states using the snapshot action, due to the following observation:

Lemma 3. *Let* $(a_1, \tau_1) \dots (a_m, \tau_m)(a_{m+1}, \tau_{m+1}) \dots (a_n, \tau_n)(\$, \tau_n)$ *be a normal run accepted by* $\mathcal{A}$*, then so is* $(a_1,\tau_1) \dots (a_m,\tau_m)(\$,\tau_m)(a_{m+1},\tau_{m+1}) \dots (a_n,\tau_n)(\$,\tau_n)$*. In particular,* $a_1 \dots a_m\$$ *is* I*-realisable.*

Proof. Check definitions and use Proposition 1. □

This observation allows us to restrict our search to paths u such that $u\$$ is I-realisable (we do *not* want to check *realisable*). This reduction ensures that every symbolic state we explore is also reachable by some equivalent interleaving in classical zone automata: We never explore paths (up to commutation) that would not be explored with standard semantics.

Algorithm 1 shows how we search for a state in F. "Waiting" is a set of paths and symbolic states to be explored, whereas "Past" is a set of symbolic states terminated with \$ that have been explored. $\lesssim_C$ is similar to a *zone inclusion test* and assures that we explore only a finite number of symbolic states, for details see [11]. It is essential, that we do not put the same symbolic states in Past and Waiting and that \$ is never used in a symbolic state in Waiting.

6 Experiments and Conclusion

We recently finished a prototype implementation based on a previous implementation of event zones without invariants. We made two experiments to demonstrate the performance of the event zone approach with state invariants. The

Algorithm 1 Generic exploration algorithm

```
Waiting ← {((s_ε, Z_ε), ε)}, Past ← ∅
while Waiting ≠ ∅ do
  Choose ((s, Z), w) ∈ Waiting, Waiting ← Waiting \ {((s, Z), w)}
  for all w' = wa with (s', Z') = (s, Z) ⊙ a consistent do
    if (s', Z'') := (s', Z') ⊙ $ consistent then
      if s' ∈ F then return "witness(w')" end if
      if there exits no (s', Z''') ∈ Past with (s', Z'') ≲_C (s', Z''') then
        Waiting ← Waiting ∪ {((s', Z'), w')}, Past ← Past ∪ {(s', Z'')}
      end if
    end if
  end for
end while
return "empty"
```

experiments were carried out in a machine with two 2.8GHz Xeon CPUs, 2GB memory and Fedora core 4 Linux.

The first experiment is a timed version of dining philosophers. There are a group of philosophers and a timestopper process[3]. Figure 3 shows the results generated by our prototype and UppAal (v3.6 beta 1). The data under the title "No partial order" were obtained by our implementation setting all transitions dependent. They are results in principle in the same basic algorithm used by UppAal, but our implementation is lacking abstraction techniques like [8], which explains the largely superior results obtained by UppAal. The potential of the reduction can thus be seen by comparing the figures with and without "partial order" reduction. However, with reduction - despite the lack of abstractions - even the current implementation outperforms UppAal.

Number of philosophers	No partial order		With partial order		UppAal	
	time	memory	time	memory	time	memory
2	0.02s	16m	0.03s	16m	0.03s	4m
3	0.11s	17m	0.05s	16m	0.04s	5m
4	21.88s	44m	0.53s	17m	0.29s	6m
5	—	—	9.79s	22m	12.86s	36m
6	—	—	175.10s	72m	1523.22s	730m
7	—	—	2909.32s	540m	—	—

Fig. 3. Results of the philosophers example

The second experiment was performed on the highway example of Section 2. The results[4] are listed in Figure 4. The advantage of event zone with state invariants against UppAal in this experiment was more explicit than the first one.

In this paper, we have shown how to add local state invariants to the event zone approach for timed automata reachability. We thus lift the application domain to the full class of reachability analysis that can be done with tools like

[3] Due to page limit, the description of these processes was put in the appendix.

[4] The number labeled "(*)" was not accurate since the swap memory was used automatically by the operating system.

Number of lanes	No partial order		With partial order		UppAal	
	time	memory	time	memory	time	memory
1	0.02s	16m	0.03s	16m	0.02s	6m
2	0.03s	16m	0.03s	16m	0.02s	6m
3	0.06s	16m	0.04s	16m	0.03s	6m
4	1.98s	22m	0.30s	17m	0.23s	7m
5	548.57s	279m	1.29s	19m	20.35s	29m
6	34681.91s(*)	2301m	10.80s	36m	2946.67s	438m
7	—	—	87.35s	119m	—	—
8	—	—	554.35s	466m	—	—

Fig. 4. Results of the highway example

UppAal or Kronos. Moreover, we have implemented the algorithm and shown that it can compete with state of the art timed automata tools.

References

1. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
2. G. Behrmann, P. Bouyer, E. Fleury, and K.G. Larsen. Static guard analysis in timed automata verification. In *In TACAS*, volume 2619 of *Lecture Notes in Computer Science*, pages 254–277. Springer Verlag, 2003.
3. G. Behrmann, A. David, K. G. Larsen, O. Moeller, P. Pettersson, and W. Yi. Uppaal - present and future. In *Proc. of 40*th *IEEE Conference on Decision and Control*. IEEE Computer Society Press, 2001.
4. G. Behrmann, K. Larsen, J. Pearson, C. Weise, W. Yi, and J. Lind-Nielsen. Efficient timed reachability analysis using clock difference diagrams. In *International Conference on Computer Aided Verification*, volume 1633 of *Lecture Notes in Computer Science*, pages 341–353, 1999.
5. W. Belluomini and C. Myers. Verification of timed systems using POSETs. In *International Conference on Computer Aided Verification*, pages 403–415, 1998.
6. J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi. Partial order reductions for timed systems. In *International Conference on Concurrency Theory (CONCUR)*, volume 1466 of *Lecture Notes in Computer Science*, pages 485–500. Springer Verlag, 1998.
7. C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. *Lecture Notes in Computer Science*, 1384:pp. 313, 1998.
8. C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In *IEE Real-Time Systems Symposium*, pages 73–81, December 1996.
9. T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111:193–244, 1994.
10. K. Larsen, P. Pettersson, and W. Yi. Model-checking for real-time systems. In *Fundamentals of Computation Theory*, Lecture Notes in Computer Science, pages 62–88. Springer Verlag, 1995.
11. D. Lugiez, P. Niebert, and S. Zennou. A partial order semantics approach to the clock explosion problem of timed automata. *Theoretical Computer Science*, 345(1):27–59, 2005.
12. S. Yovine. Kronos: A verification tool for real-time systems. *Software Tools for Technology Transfer*, 1(1):123–133, 1997.
13. S. Zennou, M. Yguel, and P. Niebert. Else: A new symbolic state generator for timed automata. In *Proceedings of the 1st International Conference FORMATS*, volume 2791 of *Lecture Notes in Computer Science*, pages 273–280. Springer, 2003.

Appendix: The timed version of dining philosophers

The automata of a philosopher and the timestopper are shown in Figure 5, respectively. Each philosopher has five states: think, hungry, leftfork, eat and dropthefork. The timestopper has two states: onestate and finalstate. This process is used to stop the execution of the system when time progresses to a limit. hungernoticed, patience, starved, eatingtime, concentrated and timelimit are constant; foodless, myclock and time are local clocks; myindex is the process id; afork and done are program variables.

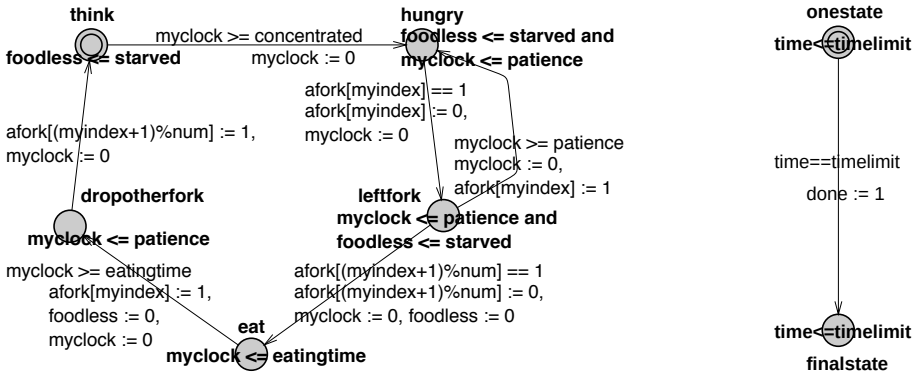

Fig. 5. The automata of a philosopher (left) and the timestopper (right)