Compositional Invariant Generation for Timed Systems

L. Aştefănoaei, S. Ben Rayana, S. Bensalem, M. Bozga, J. Combaz

UJF-Grenoble, CNRS VERIMAG UMR 5104, Grenoble F-38041, France **

Abstract. In this paper we address the state space explosion problem inherent to model-checking timed systems with a large number of components. The main challenge is to obtain pertinent global timing constraints from the timings in the components alone. To this end, we make use of auxiliary clocks to automatically generate new invariants which capture the constraints induced by the synchronisations between components. The method has been implemented as an extension of the D-Finder tool and successfully experimented on several benchmarks.

1 Introduction

Compositional methods in verification have been developed to cope with state space explosion. Generally based on divide et impera principles, these methods attempt to break monolithic verification problems into smaller sub-problems by exploiting either the structure of the system or the property or both. Compositional reasoning can be used in different manners e.g., for deductive verification, assume-guarantee, contract-based verification, compositional generation, etc.

The development of compositional verification for timed systems remains however challenging. State-of-the-art tools [7, 13, 25, 18] for the verification of such systems are mostly based on symbolic state space exploration, using efficient data structures and particularly involved exploration techniques. In the timed context, the use of compositional reasoning is inherently difficult due to the synchronous model of time. Time progress is an action that synchronises continuously all the components of the system. Getting rid of the time synchronisation is necessary for analysing independently different parts of the system (or of the property) but becomes problematic when attempting to re-compose the partial verification results. Nonetheless, compositional verification is actively investigated and several approaches have been recently developed and employed in timed interfaces [2] and contract-based assume-guarantee reasoning [15, 22].

In this paper, we propose a different approach for exploiting compositionality for analysis of timed systems using invariants. In contrast to exact reachability analysis, invariants are symbolic approximations of the set of reachable states of the system. We show that rather precise invariants can be computed compositionally, from the separate analysis of the components in the system and from

** Work partially supported by the European Integrated Projects 257414 ASCENS, 288175 CERTAINTY, and STREP 318772 D-MILS.

their composition glue. This method is proved to be sound for the verification of safety properties. However, it is not complete.

The starting point is the verification method of [9], summarised in Figure 1. The method exploits compositionality as explained next. Consider a system consisting of components B_i interacting by means of a set γ of multi-party interactions, and let Ψ be a system property of interest. Assume that all B_i as well as the composition through γ can be independently characterised by means of component invariants $CI(B_i)$, respectively interaction invariants $II(\gamma)$. The connection between the invariants and the system property Ψ can be intuitively understood as follows: if Ψ can be proved to be a logical consequence of the conjunction of components and interaction invariants, then Ψ holds for the system. In the rule (VR) the symbol " $\vdash$ " is used to underline that the logical implication can be effectively proved (for instance with an SMT solver) and the notation "$B \models \Box\, \Psi$" is to be read as "Ψ holds in every reachable state of B".

$$\frac{\vdash \bigwedge_i CI(B_i) \wedge II(\gamma) \rightarrow \Psi}{\|_\gamma B_i \models \Box\, \Psi} \text{ (VR)}$$

Fig. 1: Compositional Verification

The verification rule (VR) in [9] has been developed for untimed systems. Its direct application to timed systems may be weak as interaction invariants do not capture global timings of interactions between components. The key contribution of this paper is to improve the invariant generation method so to better track such global timings by means of auxiliary *history clocks* for actions and interactions. At component level, history clocks expose the local timing constraints relevant to the interactions of the participating components. At composition level, extra constraints on history clocks are enforced due to the simultaneity of interactions and to the synchrony of time progress.

As an illustration, let us consider as running example the timed system in Figure 2 which depicts a "controller" component serving n "worker" components, one at a time. The interactions between the controller and the workers are defined by the set of synchronisations $\{(a \mid b_i), (c \mid d_i) \mid i \leq n\}$. Periodically, after every 4 units of time, the controller synchronises its action a with the action b_i of any worker i whose clock shows at least $4n$ units of time. Initially, such a worker exists because the controller waits for $4n$ units of time before interacting with workers. The cycle repeats forever because there is always a worker "willing" to do b, that is, the system is deadlock-free. Proving deadlock-freedom of the system requires to establish that when the controller is at location lc_1 there is at least one worker such that $y_i - x \geq 4n - 4$. Unfortunately, this property cannot be shown if we use (VR) as it is in [9]. Intuitively, this is because the proposed invariants are too weak to infer such cross constraints relating the clocks of the

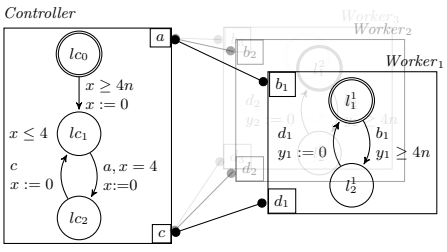

Fig. 2: A Timed System

controller and the clocks of the workers: interaction invariants $II(\gamma)$ relates only locations of components and thus at most eliminates unreachable configurations like $(lc_1, \ldots, l_2^i, \ldots)$, while the component invariants can only state local conditions on clocks such as that $x \leq 4$ at lc_1. Using history clocks allows to recover additional constraints. For example, after the first execution of the loop, each time when the controller is at location lc_1, there exists a worker i whose clock has an equal value as that of the controller. Similarly, history clocks allow to infer that different $(a \mid b_i)$ interactions are separated by at least 4 time units. These constraints altogether are then sufficient to prove the deadlock freedom property.

Related Work. Automatic generation of invariants for concurrent systems is a long-time studied topic. Yet, to our knowledge, specific extensions or applications for timed systems are rather limited. As an exception, the papers [5, 17] propose a monolithic, non-compositional method for finding invariants in the case of systems represented as a single timed automaton.

Compositional verification for timed systems has been mainly considered in the context of timed interface theories [2] and contract-based assume guarantee reasoning [15, 22]. These methods usually rely upon choosing a "good" decomposition structure and require individual abstractions for components to be deterministic timed I/O automata. Finding the abstractions is in general difficult, however, their construction can be automated by using learning techniques [22] in some cases. In contrast to the above, we are proposing a fully automated method generating, in a compositional manner, an invariant approximating the reachable states of a timed system. Abstractions serve also for compositional minimisation, for instance [10] minimises by constructing timed automata quotients with respect to simulation; these quotients are in turn composed for model-checking. Our approach is orthogonal in that we do not compose at all. Compositional deductive verification as in [16] is also orthogonal on our work in that, by choosing a particular class of local invariants to work with, we need not focus on elaborate proof systems but reason at a level closer to intuition.

The use of additional clocks has been considered, for instance, in [8]. There, extra reference clocks are added to components to faithfully implement a partial order reduction strategy for symbolic state space exploration. Time is allowed to progress desynchronised for individual components and re-synchronised only when needed, i.e., for direct interaction within components. Clearly, the history clocks in our work behave in a similar way, however, our use of clocks is as a helper construction in the generation of invariants and we are totally avoiding state space exploration. Finally, another successful application of extra clocks has been provided in [23] for timing analysis of asynchronous circuits. There, specific history clocks are reset on input signals and used to provide a new time basis for the construction of an abstract model of output signals of the circuit.

Organisation of the paper. Section 2 recalls the needed definitions for modelling timed systems and their properties. Section 3 presents our method for compositional generation of invariants. Section 4 describes the prototype implementing the method and some case studies we experimented with. Section 5 concludes.

2 Timed Systems and Properties

In the framework of the present paper, components are timed automata and systems are compositions of timed automata with respect to multi-party interactions. The timed automata we use are essentially the ones from [3], however, sligthly adapted to embrace a uniform notation throughout the paper.

Definition 1 (Syntax of a Component). *A component is a timed automaton* $(L, l_0, A, T, \mathcal{X}, \mathsf{tpc})$ *where* L *is a finite set of locations,* l_0 *is an initial location,* A *a finite set of actions,* $T \subseteq L \times (A \times \mathcal{C} \times 2^{\mathcal{X}}) \times L$ *is a set of edges labeled with an action, a guard, and a set of clocks to be reset,* $\mathcal{X}$ *is a finite set of clocks*[1]*, and* $\mathsf{tpc} : L \to \mathcal{C}$ *assigns a time progress condition*[2] *to each location.* $\mathcal{C}$ *is the set of clock constraints. A clock constraint is defined by the grammar:*

$$C ::= \mathit{true} \mid \mathit{false} \mid x \# ct \mid x - y \# ct \mid C \wedge C$$

with $x, y \in \mathcal{X}$*,* $\# \in \{<, \leq, =, \geq, >\}$ *and* $ct \in \mathbb{Z}$*. Time progress conditions are restricted to conjunctions of constraints as* $x \leq ct$*.*

Before recalling the semantics of a component, we first fix some notation. Let $\mathbf{V}$ be the set of all clock valuation functions $\mathbf{v} : \mathcal{X} \to \mathbb{R}_{\geq 0}$. For a clock constraint C, $C(\mathbf{v})$ denotes the evaluation of C in $\mathbf{v}$. The notation $\mathbf{v} + \delta$ represents a new $\mathbf{v}'$ defined as $\mathbf{v}'(x) = \mathbf{v}(x) + \delta$ while $\mathbf{v}[r]$ represents a new $\mathbf{v}'$ which assigns any x in r to 0 and otherwise preserves the values from $\mathbf{v}$.

Definition 2 (Semantics of a Component). *The semantics of a component* $B = (L, l_0, A, T, X, \mathsf{tpc})$ *is given by the labelled transition system* $(Q, A, \to)$ *where* $Q \subseteq L \times \mathbf{V}$ *denotes the states of* B *and* $\to \subseteq Q \times (A \cup \mathbb{R}_{\geq 0}) \times Q$ *denotes the transitions according to the rules:*

- $(l, \mathbf{v}) \xrightarrow{\delta} (l, \mathbf{v} + \delta)$ *if* $\big(\forall \delta' \in [0, \delta]\big).(\mathsf{tpc}(l)(\mathbf{v} + \delta'))$ *(time progress);*
- $(l, \mathbf{v}) \xrightarrow{a} (l', \mathbf{v}[r])$ *if* $\big(l, (a, g, r), l'\big) \in T$*,* $g(\mathbf{v}) \wedge \mathsf{tpc}(l')(\mathbf{v}[r])$ *(action step).*

Because the semantics defined above is in general infinite, we work with the so called zone graph [19] as a finite symbolic representation. The symbolic states in a zone graph are pairs (l, ζ) where l is a location of B and ζ is a *zone*, a set of clock valuations defined by clock constraints. Given a symbolic state (l, ζ), its successor with respect to a transition t of B is denoted as $\mathsf{succ}(t, (l, \zeta))$ and defined by means of its timed and its discrete successor:

- $\mathsf{time\text{-}succ}((l, \zeta)) = (l, \nearrow \zeta \cap \mathsf{tpc}(l))$
- $\mathsf{disc\text{-}succ}(t, (l, \zeta)) = (l', (\zeta \cap g)[r] \cap \mathsf{tpc}(l'))$ if $t = \big(l, (\_, g, r), l'\big)$
- $\mathsf{succ}(t, (l, \zeta)) = \mathsf{norm}(\mathsf{time\text{-}succ}(\mathsf{disc\text{-}succ}(t, (l, \zeta))))$

[1] Clocks are local. This is essential for avoiding side effects which would break compositionality and local analysis.

[2] To avoid confusion with invariant properties, we prefer to adopt the terminology of "time progress condition" from [11] instead of "location invariants".

where $\nearrow, [r], \mathsf{norm}$ are usual operations on zones: $\nearrow \zeta$ is the forward diagonal projection of ζ, i.e., it contains any valuation $\mathbf{v}'$ for which there exists a real δ such that $\mathbf{v}' - \delta$ is in ζ; $\zeta[r]$ is the set of all valuations in ζ after applying the resets in r; $\mathsf{norm}(\zeta)$ corresponds to normalising ζ such that computation of the set of all successors terminates. Since we are seeking component invariants which are over-approximations of the reachable states, a more thorough discussion on normalisation is not relevant for the present paper. The interested reader may refer to [12] for more precise definitions.

A symbolic execution of a component starting from a symbolic state s_0 is a sequence of symbolic states $s_0, s_1, \ldots, s_n, \ldots$ such that for any $i > 0$ there exists a transition t for which s_i is $\mathsf{succ}(t, s_{i-1})$.

Given a component B with initial symbolic state s_0 and transitions T, the set of reachable symbolic states $\mathit{Reach}(B)$ is $\mathit{Reach}(s_0)$ where Reach is defined recursively for an arbitrary s as:

$$\mathit{Reach}(s) = \{s\} \cup \bigcup_{t \in T} \mathit{Reach}(\mathsf{succ}(t, s)).$$

In our framework, components communicate by means of *interactions*, which are synchronisations between their actions. Given n components B_i, $1 \leq i \leq n$, with disjoint sets of actions A_i, an interaction is a subset of actions $\alpha \subseteq \cup_i A_i$ containing at most one action per component, that is, of the form $\alpha = \{a_i\}_{i \in I}$, with $a_i \in A_i$ for all $i \in I \subseteq \{1, \ldots, n\}$. Given a set of interactions $\gamma \subseteq 2^{\cup_i A_i}$, we denote by $\mathit{Act}(\gamma)$ the set of actions involved in γ, that is, $\mathit{Act}(\gamma) = \cup_{\alpha \in \gamma} \alpha$. A *timed system* is the composition of components B_i for a set of interactions γ such that $\mathit{Act}(\gamma) = \cup_i A_i$.

Definition 3 (Timed System). *For n components $B_i = (L_i, l_0^i, A_i, T_i, \mathcal{X}_i, \mathsf{tpc}_i)$ with $A_i \cap A_j = \emptyset$, $\mathcal{X}_i \cap \mathcal{X}_j = \emptyset$, for any $i \neq j$, the* composition $\|_\gamma B_i$ *w.r.t. a set of interactions γ is defined by a timed automaton $(L, \bar{l}_0, \gamma, T_\gamma, \mathcal{X}, \mathsf{tpc})$ where $\bar{l}_0 = (l_0^1, \ldots, l_0^n)$, $\mathcal{X} = \cup_i \mathcal{X}_i$, $L = \times_i L_i$, $\mathsf{tpc}(\bar{l}) = \wedge_i \mathsf{tpc}(l_i)$, and T_γ is such that for any interaction $\alpha = \{a_i\}_{i \in I}$ we have that $\bar{l} \xrightarrow{\alpha, g, r} \bar{l}'$ where $\bar{l} = (l_1, \ldots, l_n)$, $g = \wedge_{i \in I} g_i$, $r = \cup_{i \in I} r_i$, and $\bar{l}'(i) =$ if $(i \notin I)$ l_i else l_i' for $l_i \xrightarrow{a_i, g_i, r_i} l_i'$.*

In the timed system $\|_\gamma B_i$ a component B_i can execute an action a_i only as part of an interaction α, $a_i \in \alpha$, that is, along with the execution of all other actions $a_j \in \alpha$, which corresponds to the usual notion of multi-party interaction. Notice that interactions can only restrict the behavior of components, i.e. the states reached by B_i in $\|_\gamma B_i$ belong to $\mathit{Reach}(B_i)$. This property is exploited by the verification rule (VR) presented throughout this paper.

To give a logical characterisation of components and interactions we use invariants. An invariant Φ is a state property which holds in every reachable state of a component (or of a system) B, in symbols, $B \models \Box\Phi$. We use $\mathit{CI}(B_i)$ and $\mathit{II}(\gamma)$, to denote component, respectively interaction invariants. For component invariants, our choice is to work with their reachable set. More precisely, for a component B with initial state s_0, $\mathit{CI}(B)$ is the disjunction of $(l \wedge \zeta)$ and where, to ease the reading, we abuse notation and use l as a place holder for a

state predicate "$at(l)$" which holds in any symbolic state with location l, that is, the semantics of $at(l)$ is given by $(l, \zeta) \models at(l)$. As an example, the component invariants for the scenario in Figure 2 with one worker are:

$$CI(Controller) = (lc_0 \wedge x \geq 0) \vee (lc_1 \wedge x \leq 4) \vee (lc_2 \wedge x \geq 0)$$
$$CI(Worker_i) = (l_1^i \wedge y_i \geq 0) \vee (l_2^i \wedge y_i \geq 4).$$

Interaction invariants are over-approximations of global state spaces allowing us to disregard certain tuples of local states as unreachable. Interaction invariants relate locations of different atomic components. They are either boolean e.g., $l_1 \vee l_2 \vee l_3$ or linear e.g., $l_1 + l_2 + l_3 = 1$. These particular examples ensure that at least (resp. exactly) one of the locations l_1, l_2, l_3 are active at any time. Interaction invariants are computed on the synchronization skeleton of the composition, that is, a 1-safe Petri net obtained by composing component behaviours according to the interaction glue. The methods rely on boolean ([9]) / algebraic ([21]) constraint solving and avoid any form of state-space exploration. In the case of the running example, when the controller is interacting with one worker, the interaction invariant $II\big(\{(a \mid b_1), (c \mid d_1)\}\big)$ is $(lc_2 \vee l_1^1) \wedge (l_2^1 \vee lc_0 \vee lc_1)$.

The proposed invariants[3] have the feature that they are inductive. We recall that an invariant Φ is inductive if it holds initially and if for a state s s.t. $s \models \Phi$ we have that $s' \models \Phi$ for any successor s' of s. Moreover, inductive invariants have the property that their conjunction is also an inductive invariant.

3 Timed Invariant Generation

As explained in the introduction, a direct application of (VR) may not be useful in itself in the sense that the component and the interaction invariants alone are usually not enough to prove global properties, especially when the properties involve relations between clocks in different components. More precisely, though component invariants encode timings of local clocks, there is no direct way – the interaction invariant is orthogonal on timing aspects – to constrain the bounds on the differences between clocks in different components. To give a concrete illustration, consider the safety property $\Psi_{Safe} = (lc_1 \wedge l_1^1 \rightarrow x \leq y_1)$ that holds in the running example with one worker. It is not difficult to see that Ψ_{Safe} cannot be deduced from $CI(Controller) \wedge CI(Worker_1) \wedge II\big(\{(a \mid b_1), (c \mid d_1)\}\big)$.

3.1 History Clocks for Actions

In this section we show how we can, by means of some auxiliary constructions, apply (VR) more successfully. To this end, we "equip" components (and later, interactions) with *history clocks*, a clock per action; then, at interaction time, the clocks corresponding to the actions participating in the interaction are reset.

[3] The rule (VR) is generic enough to work with other types of invariants. For example, one could use over-approximations of the reachable set in the case of component invariants, however, this comes at the price of losing precision.

This basic transformation allows us to automatically compute a new invariant of the system with history clocks. This new invariant, together with the component and interaction invariants, is shown to be, after projection of history clocks, an invariant of the initial system.

Definition 4 (Components with History Clocks). *Given a component model* $B = (L, l_0, A, T, \mathcal{X}, \mathsf{tpc})$*, its extension wrt history clocks is a timed automaton* $B^h = (L, l_0, A, T^h, \mathcal{X} \cup \mathcal{H}_A, \mathsf{tpc})$ *where:*

- $\mathcal{H}_A = \{h_a \mid a \in A\} \cup \{h_0\}$ *is the set of history clocks associated to actions and* h_o*, a history clock dedicated to initialisation. Together with the clocks in* $\mathcal{X}$*,* h_0 *is initialised to zero. All other clocks in* $\mathcal{H}_A$ *may be initialised to any arbitrary positive value.*
- $T^h = \big\{\big(l, (a, g, r \cup [h_a := 0]), l'\big) \mid \big(l, (a, g, r), l'\big) \in T\big\}$.

Since there is no timing constraint involving history clocks, these have no influence on the behaviour. The extended model is, in fact, bisimilar to the original model. Moreover, any invariant of the composition of B_i^h corresponds to an invariant of $\|_\gamma B_i$. For the ease of reading, we abuse notation and use $\exists \mathcal{H}_A$ to stand for $\exists h_0 \exists h_{a_1} \exists h_{a_2} \ldots \exists h_{a_n}$ for $A = \{a_1, a_2, \ldots, a_n\}$.

Proposition 1 *Any symbolic execution in* B^h *corresponds to a symbolic execution (where all constraints on history clocks are ignored) in* B*. Moreover, if* $\|_\gamma B_i^h \models \Box\Phi$ *then* $\|_\gamma B_i \models \Box(\exists \mathcal{H}_A).\Phi$.

The only operation acting on history clocks is reset. Its effect is that immediately after an interaction takes place, all history clocks involved in the interaction are equal to zero. All other history clocks preserve their previous values, thus they are at least greater in value than all those being reset. This basic but useful observation is exploited in the following definition, which builds, recursively, all the inequalities that could hold given an interaction set γ.

Definition 5 (Interaction Inequalities for History Clocks). *Given an interaction set* γ*, we define the following interaction inequalities* $\mathcal{E}(\gamma)$*:*

$$\mathcal{E}(\gamma) = \bigvee_{\alpha \in \gamma} \Big(\big(\bigwedge_{\substack{a_i, a_j \in \alpha \\ a_k \notin \alpha}} h_{a_i} = h_{a_j} \leq h_{a_k} \big) \wedge \mathcal{E}(\gamma \ominus \alpha) \Big).$$

where $\gamma \ominus \alpha = \{\beta \setminus \alpha \mid \beta \in \gamma \wedge \beta \not\subseteq \alpha\}$.

Remark 1. We can use the interpreted function "min" as syntactic sugar to have a more compact expression for $\mathcal{E}(\gamma)$:

$$\mathcal{E}(\gamma) = \bigvee_{\alpha \in \gamma} \Big(\bigwedge_{a_i, a_j \in \alpha} h_{a_i} = h_{a_j} \leq \min_{a_k \notin \alpha} h_{a_k} \wedge \mathcal{E}(\gamma \ominus \alpha) \Big).$$

Example 1. For $\gamma = \{(a \mid b_1), (c \mid d_1)\}$, corresponding to the interactions between the controller and one worker in Figure 2, the compact form of $\mathcal{E}(\gamma)$ is:

$$\big(h_a = h_{b_1} \leq \min(h_c, h_{d_1}) \wedge h_c = h_{d_1}\big) \vee \big(h_c = h_{d_1} \leq \min(h_a, h_{b_1}) \wedge h_a = h_{b_1}\big).$$

$\mathcal{E}(\gamma)$ characterises the relations between history clocks during any possible execution of a system. It can be shown, by induction, that this characterisation is, in fact, an inductive invariant of the extended system.

Proposition 2 *$\mathcal{E}(\gamma)$ is an inductive invariant of $\|_\gamma B_i^h$.*

By Proposition 2, and using the fact that component and interaction invariants are inductive, we have that also their conjunction is an inductive invariant of the system with history clocks. As a consequence of Proposition 1, we can eliminate the history clocks from $\wedge_i CI(B_i^h) \wedge II(\gamma) \wedge \mathcal{E}(\gamma)$ and obtain an invariant of the original system. This invariant is usually stronger than $CI(B_i) \wedge II(\gamma)$ and yields more successful applications of the rule (VR).

Example 2. We reconsider the sub-system of a controller and a worker from Figure 2. We illustrate how the safety property ψ_{Safe} introduced in the beginning of the section can be shown to hold by using the newly generated invariant. The invariants for the components with history clocks are:

$$\begin{aligned} CI(Controller^h) = {} & (lc_0 \wedge h_0 = x) \vee \\ & (lc_1 \wedge x \leq 4 \wedge h_a \leq h_0 \wedge (h_a = h_c \geq 4 + x \ \vee x = h_c \leq h_a)) \vee \\ & (lc_2 \wedge x = h_a \wedge h_c \leq h_0 \wedge (h_c \geq h_a + 8 \vee h_c = h_a + 4)) \\ CI(Worker_1^h) = {} & (l_1^1 \wedge (y_1 = h_0 \vee y_1 = h_{d_1} \leq h_{b_1} \leq h_0)) \vee \\ & (l_2^1 \wedge h_0 \geq y_1 = h_{d_1} \geq 4 + h_{b_1}) \end{aligned}$$

By using the interaction invariant described in Section 2 and the equality constraints $\mathcal{E}(\gamma)$ from Example 1, after the elimination of the existential quantifiers in $\big(\exists h_0.\exists h_a.\exists h_{b_1}.\exists h_c.\exists h_{d_1}\big)\big(CI(Controller^h) \wedge CI(Worker_1^h) \wedge II(\gamma) \wedge \mathcal{E}(\gamma)\big)$ we obtain the following invariant Φ:

$$\begin{aligned} \Phi = {} & (l_1^1 \wedge lc_0 \wedge (\boldsymbol{y_1 \leq x})) \vee \big(l_1^1 \wedge lc_1 \wedge (\boldsymbol{y_1 = x \ \vee \ y_1 \geq x + 4})\big) \vee \\ & \big(l_2^1 \wedge lc_2 \wedge \boldsymbol{(y_1 = x + 4 \ \vee \ y_1 \geq x + 8)}\big). \end{aligned}$$

It can be easily checked that $\Phi \wedge \neg \Psi_{Safe}$ has no satisfying model and this proves that Ψ_{Safe} holds for the system. We used bold fonts in Φ to highlight relations between x and y_1 which are not in $CI(Controller) \wedge CI(Worker_1) \wedge II(\gamma)$.

To sum up, the basic steps described so far are: (1) extend the input components B_i to components with history clocks B_i^h; (2) compute component invariants $CI(B_i^h)$ and (3) equality constraints $\mathcal{E}(\gamma)$ from the interactions γ; (4) finally, eliminate the history clocks in $\wedge_i CI(B_i^h) \wedge \mathcal{E}(\gamma) \wedge II(\gamma)$, and obtain a stronger invariant by means of which the application of (VR) is more successful.

We conclude the section with a remark on the size of $\mathcal{E}(\gamma)$. Due to the combination of recursion and disjunction, $\mathcal{E}(\gamma)$ can be large. Much more compact formulae can be obtained by exploiting non-conflicting interactions, i.e., interactions that do not share actions.

Proposition 3 *For $\gamma = \gamma_1 \cup \gamma_2$ with $Act(\gamma_1) \cap Act(\gamma_2) = \emptyset$, $\mathcal{E}(\gamma) \equiv \mathcal{E}(\gamma_1) \wedge \mathcal{E}(\gamma_2)$.*

Corollary 4 *If the interaction model γ has only disjoint interactions, i.e., for any $\alpha_1, \alpha_2 \in \gamma$, $\alpha_1 \cap \alpha_2 = \emptyset$, then $\mathcal{E}(\gamma) \equiv \bigwedge_{\alpha \in \gamma} \Big(\bigwedge_{a_i, a_j \in \alpha} h_{a_i} = h_{a_j} \Big)$.*

Example 3. The interaction set γ in Example 1 is not conflicting. Thus, by applying Corollary 4, we can simplify the expression of $\mathcal{E}(\gamma)$ to $(h_a = h_{b_1}) \wedge (h_c = h_{d_1})$.

3.2 History Clocks for Interactions

The equality constraints on history clocks allow to relate the local constraints obtained individually on components. In the case of non-conflicting interactions, the relation is rather "tight", that is, expressed as conjunction of equalities on history clocks. In contrast, the presence of conflicts lead to a significantly weaker form. Intuitively, every action in conflict can be potentially used in different interactions. The uncertainty on its exact use leads to a disjunctive expression as well as to more restricted equalities and inequalities amongst history clocks.

Nonetheless, the presence of conflicts themselves can be additionally exploited for the generation of new invariants. That is, in contrast to equality constraints obtained from interaction, the presence of conflicting actions enforce disequalities (or separation) constraints between all interactions using them. In what follows, we show a generic way to automatically compute such invariants enforcing differences between the timings of the interactions themselves. To effectively implement this, we proceed in a similar manner as in the previous section: we again make use of history clocks and corresponding resets but this time we associate them to interactions, at the system level.

Definition 6 (Systems with Interaction History Clocks). *Given a system $\|_\gamma B_i$, its extension wrt history clocks for interactions is $\|_{\gamma^h} B_i^h, \Gamma^*$ where:*

- *Γ^* is an auxiliary TA having one location l with no invariant, and for each interaction α in γ a clock h_α, i.e., $\Gamma^* = (\{l^*\}, A_\gamma, T, \mathcal{H}_\gamma, \emptyset)$ where:*
  - *the set of actions $A_\gamma = \{a_\alpha \mid \alpha \in \gamma\}$*
  - *the set of clocks $\mathcal{H}_\gamma = \{h_\alpha \mid \alpha \in \gamma\}$*
  - *$T = \{(l^*, a_\alpha, true, h_\alpha := 0, l^*) \mid \alpha \in \gamma\}$*
- *$\gamma^h = \{(a_\alpha \mid \alpha) \mid \alpha \in \gamma\}$ with $(a_\alpha \mid \alpha)$ denoting $\{a_\alpha\} \cup \{a \mid a \in \alpha\}$.*

Using a similar argument as for Proposition 1, it can be shown that any invariant of $\|_{\gamma^h} B_i^h, \Gamma^*$ corresponds to an invariant of $\|_\gamma B_i$ by first showing that any execution of $\|_{\gamma^h} B_i^h, \Gamma^*$ corresponds to an execution of $\|_\gamma B_i$.

Proposition 5 *Any execution in $\|_{\gamma^h} B_i^h, \Gamma^*$ corresponds to an execution in $\|_\gamma B_i$. Moreover, if $\|_{\gamma^h} B_i^h, \Gamma^* \models \Box\, \Phi$ then $\|_\gamma B_i \models \Box\, \exists \mathcal{H}_\gamma \exists \mathcal{H}_A . \Phi$ where the new notation $\exists \mathcal{H}_\gamma$ stands for $\exists h_{\alpha_1} \exists h_{\alpha_2} \ldots \exists h_{\alpha_n}$ when $\gamma = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$.*

We use history clocks for interactions to express additional constraints on their timing. The starting point is the observation that when two conflicting interactions compete for the same action a, no matter which one is first, the other one must wait until the component which owns a is again able to execute a. This is referred to as a "separation constraint" for conflicting interactions.

Definition 7 (Separation Constraints for Interaction Clocks). *Given an interaction set γ, the induced separation constraints, $\mathcal{S}(\gamma)$, are defined as follows:*

$$\mathcal{S}(\gamma) = \bigwedge_{a \in Act(\gamma)} \bigwedge_{\substack{\alpha \neq \beta \in \gamma \\ a \in \alpha \cap \beta}} \mid h_\alpha - h_\beta \mid \geq k_a$$

where $\mid\ \mid$ stands for absolute values and k_a denotes the minimum between the first occurrence time of a and the minimal time elapse between two consecutive occurrences of a. It is computed[4] *locally on the component executing a.*

Example 4. In our running example the only shared actions are a and c within the controller, and both k_a and k_c are equal to 4, thus the expression of the separation constraints reduces to:

$$\mathcal{S}(\gamma) \equiv \bigwedge_{i \neq j} |h_{c|d_i} - h_{c|d_j}| \geq 4 \wedge \bigwedge_{i \neq j} |h_{a|b_i} - h_{a|b_j}| \geq 4.$$

Proposition 6 *$\mathcal{S}(\gamma)$ is an inductive invariant of $\|_{\gamma^h} B_i^h, \Gamma^*$.*

Proof. By induction on the length of computations. For the base case, we assume that the initial values of the history clocks for interactions in Γ^* are such that they satisfy $\mathcal{S}(\gamma)$. Obviously, such a satisfying initial model always exists: it suffices to take all h_α with a minimal distance between them greater than the maximum k_a, in an arbitrary order.

For the inductive step, let s be the state reached after i steps, s' a successor, α an interaction such that $s \xrightarrow{\alpha} s'$, a an arbitrary action and $\beta \in \gamma$ such that $a \in \beta$. For any $\beta' \neq \alpha$, $\mid h_\beta - h_{\beta'} \mid \geq k_a$ is unchanged from s to s' (α is the only interaction for which h_α is reset from s to s') and thus holds by induction. We now turn to $\mid h_\beta - h_\alpha \mid$ which at s' evaluates to h_β. Let s_a be the most recent state reached by an interaction containing a. If no such interaction exists, that is, if a has no appearance in the i steps to s, let s_a be the initial state. On the path from s_a to s', h_β could not have been reset (otherwise, s_a would not be the most recent one). Thus $h_\beta \geq k_a$ by the definition of k_a. □

The invariant $\mathcal{S}(\gamma)$ is defined over the history clocks for interactions. Previously, the invariant $\mathcal{E}(\gamma)$ has been expressed using history clocks for actions. In order to "glue" them together in a meaningful way, we need some connection between history and interaction clocks. This connection is formally addressed by the constraints $\mathcal{E}^*$ defined below.

Definition 8 ($\mathcal{E}^*$). *Given an interaction set γ, we define $\mathcal{E}^*(\gamma)$ as follows:*

$$\mathcal{E}^*(\gamma) = \bigwedge_{a \in Act(\gamma)} h_a = \min_{\alpha \in \gamma, a \in \alpha} h_\alpha.$$

[4] For instance, by reduction to a shortest path problem in weighted graphs [14].

By a similar argument as the one in Proposition 2, it can be shown that $\mathcal{E}^*(\gamma)$ is an inductive invariant of the extended system. The connection between $\mathcal{E}$ and $\mathcal{E}^*$ is given in Proposition 7.

Proposition 7 *$\mathcal{E}^*(\gamma)$ is an inductive invariant of $\|_{\gamma^h} B_i^h, \Gamma^*$. Moreover, the equivalence $\exists \mathcal{H}_\gamma.\mathcal{E}^*(\gamma) \equiv \mathcal{E}(\gamma)$ is a valid formula.*

Proof. To see that $\mathcal{E}^*(\gamma)$ is an invariant it suffices to note that, for any action a, there is always an interaction α containing a such that h_a and h_α are both reset in the same time.

The connection between $\mathcal{E}$ and $\mathcal{E}^*$ is shown by induction on the number of interactions in γ. We only present the base case, $\gamma = \{\alpha\}$, (the inductive one as well as all proofs can be found in [4]):

$$\mathcal{E}(\gamma) = \bigwedge_{a_i, a_j \in \alpha} h_{a_i} = h_{a_j} \equiv \exists h_\alpha.\big(\bigwedge_{a_i \in \alpha} h_{a_i} = h_\alpha\big) \equiv$$
$$\exists \mathcal{H}_\gamma.\big(\bigwedge_{a_i \in Act(\gamma)} h_{a_i} = \min_{\alpha \in \gamma, a_i \in \alpha} h_\alpha\big) \equiv \exists \mathcal{H}_\gamma.\mathcal{E}^*(\gamma).$$

□

From Proposition 7, together with Propositions 5 and 6, it follows that $\exists \mathcal{H}_A \exists \mathcal{H}_\gamma.(\wedge_i CI(B_i^h) \wedge II(\gamma) \wedge \mathcal{E}^*(\gamma) \wedge \mathcal{S}(\gamma))$ is an invariant of $\|_\gamma B_i$. This new invariant is in general stronger than $\wedge_i CI(B_i^h) \wedge II(\gamma) \wedge \mathcal{E}(\gamma)$ and it provides better state space approximations for timed systems with conflicting interactions.

Example 5. To get some intuition about the invariant generated using separation constraints, let us reconsider the running example with two workers. The subformula which we emphasise here is the conjunction of $\mathcal{E}^*$ and $\mathcal{S}$. The interaction inequalities for history clocks are:

$$\mathcal{E}^*(\gamma) \equiv h_{b_1} = h_{a|b_1} \wedge h_{b_2} = h_{a|b_2} \wedge\ h_a = \min_{i=1,2}(h_{a|b_i}) \wedge$$
$$h_{d_1} = h_{c|d_1} \wedge h_{d_2} = h_{c|d_2} \wedge\ h_c = \min_{i=1,2}(h_{c|d_i})$$

by recalling the expression of $\mathcal{S}(\gamma)$ from Example 4 we obtain that:

$$\exists \mathcal{H}_\gamma.\mathcal{E}^*(\gamma) \wedge \mathcal{S}(\gamma)\ \equiv |h_{b_2} - h_{b_1}| \geq 4 \wedge |h_{d_2} - h_{d_1}| \geq 4$$

and thus, after quantifier elimination in $\exists \mathcal{H}_A \exists \mathcal{H}_\gamma.(CI(Controller^h) \wedge_i CI(Worker_i^h) \wedge II(\gamma) \wedge \mathcal{E}^*(\gamma) \wedge \mathcal{S}(\gamma))$, we obtain the following invariant Φ:

$$\begin{aligned}\Phi = &\big(l_1^1 \wedge l_1^2 \wedge lc_0 \wedge\ x = y_1 = y_2\big) \vee \\ &\big(l_1^1 \wedge l_1^2 \wedge lc_1 \wedge x \leq 4 \wedge ((y_1 = x\ \wedge \boldsymbol{y_2 - y_1 \geq 4} \vee\ y_1 \geq x + 8 \vee \\ &\qquad\qquad y_2 = x\ \wedge \boldsymbol{y_1 - y_2 \geq 4} \vee\ y_2 \geq x + 8))\big) \vee \\ &\big(l_2^1 \wedge l_1^2 \wedge lc_2 \wedge (y_1 \geq x + 8 \vee (y_2 = x + 4\ \wedge\ \boldsymbol{y_1 - y_2 \geq 4}))\big) \vee \\ &\big(l_1^2 \wedge l_2^1 \wedge lc_2 \wedge (y_2 \geq x + 8 \vee (y_1 = x + 4\ \wedge\ \boldsymbol{y_2 - y_1 \geq 4}))\big)\end{aligned}$$

We emphasised in the expression of Φ the newly discovered constraints. All in all, Φ is strong enough to prove that the system is deadlock free.

4 Implementation and Experiments

The method has been implemented in a Scala (`scala-lang.org/`) prototype (`www-verimag.imag.fr/~lastefan/tas`) which is currently being integrated with the D-Finder tool [9] for verification of Real-Time BIP systems [1]. The prototype takes as input components B_i, an interaction set γ and a global safety property Ψ and checks whether the system satisfies Ψ. Internally, it uses PPL (`bugseng.com/products/ppl`) to manipulate zones (essentially polyhedra) and to compute component invariants. It generates Z3 (`z3.codeplex.com`) Python code to check the satisfiability of the formula $\wedge_i CI(B_i) \wedge II(\gamma) \wedge \Phi^* \wedge \neg\Psi$ where Φ^*, depending on whether γ is conflicting, stands for $\mathcal{E}(\gamma)$ or $\mathcal{E}^*(\gamma) \wedge \mathcal{S}(\gamma)$. If the formula is not satisfiable, the prototype returns `no solution`, that is, the system is guaranteed to satisfy Ψ. Otherwise, it returns a substitution for which the formula is satisfiable, that is, the conjunction of invariants is true while Ψ is not. This substitution may correspond to a false positive in the sense that the state represented by the substitution could be unreachable.
For experiments, we chose three classical benchmarks which we discuss below.

Train gate controller: This is a classical example from [3]. The system is composed of a controller, a gate and a number of trains. For simplicity, Figure 3 depicts only one train interacting with the controller and the gate. The controller lowers and raises the gate when a train enters, respectively exits. The safety property of interest is that when a train is at location `in`, the gate has been lowered: $\wedge_i(in_i \rightarrow g_2)$. When there is only one train in the system, $\mathcal{E}(\gamma)$ is enough to show safety. When there are more trains, we use the separation constraints.

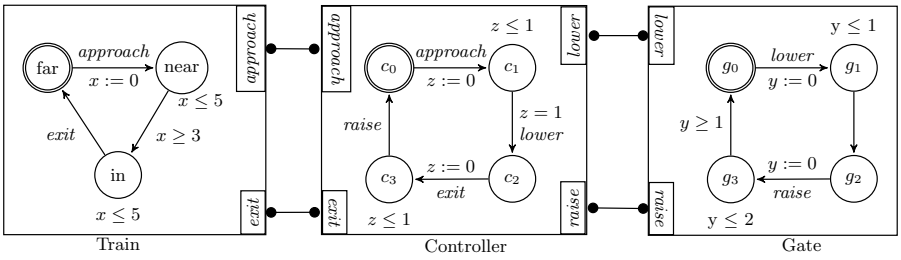

Fig. 3: A Controller Interacting with a Train and a Gate

Fischer Protocol: This is a well-studied protocol for mutual exclusion [20]. The protocol specifies how processes can share a resource one at a time by means of a shared variable to which each process assigns its own identifier number. After θ time units, the process with the id stored in the variable enters the critical state and uses the resource. We use an auxiliary component `Id Variable` to mimic the role of the shared variable. To keep the size of the generated invariants manageable, we restrict to the acyclic version. The system with two concurrent processes is represented in Figure 4. The property of interest is mutual exclusion: $(cs_i \wedge cs_j) \rightarrow i = j$. The component `Id Variable` has combinatorial behavior and a large number of actions $(2n + 1)$, thus the generated invariant

is huge except for very small values of n. To overcome this issue, we extracted from the structure of the generated invariant a weaker inductive one which we verified for validity locally with Uppaal. Basically, it encodes information like $h_{eq_i} < h_{set_i} \rightarrow h_{eq_i} < h_{eq_0} \wedge h_{set_i} < h_{eq_0}$ for any index i. This invariant, together with the component invariants for the processes and together with $\mathcal{E}(\gamma)$ is sufficient to show that mutual exclusion holds.

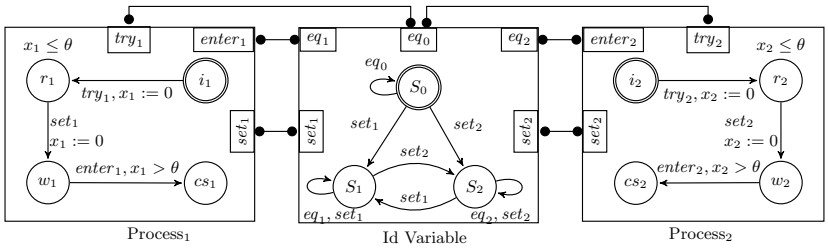

Fig. 4: The Fischer Protocol

Temperature controller: This example is an adaptation from [9]. It represents a simplified model of a nuclear plant. The system consists of a controller interacting with an arbitrary number n of rods (two, in Figure 5) in order to maintain the temperature between the bounds 450 and 900: when the temperature in the reactor reaches 900 (resp. 450), a rod must be used to cool (resp. heat) the reactor. The rods are enabled to cool only after $900n$ units of time. The global property of interest is the absence of deadlock, that is, the system can run continuously and keep the temperature between the bounds. To express this property in our prototype, we adapt from [24] the definition of *enabled* states, while in Uppaal, we use the query `A[] not deadlock`. For one rod, $\mathcal{E}(\gamma)$ is enough to show the property. For more rods, because interactions are conflicting, we need the separation constraints which basically bring as new information conjunctions as $\wedge_i(h_{rest_{\pi(i)}} - h_{rest_{\pi(i-1)}} \geq 1350)$ for π an ordering on rods.

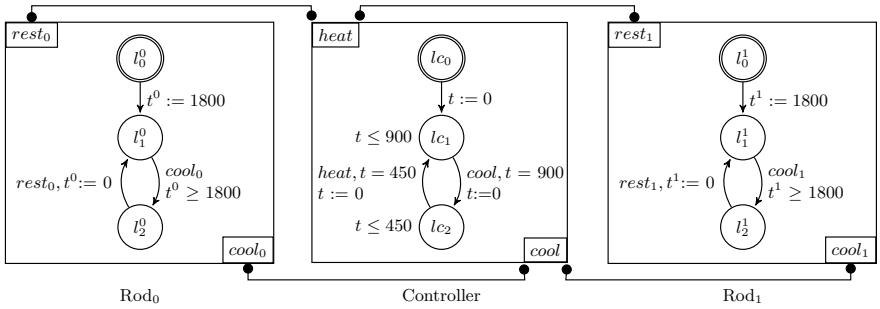

Fig. 5: A Controller Interacting with 2 Rods

The experiments were run on a Dell machine with Ubuntu 12.04, an Intel(R) Core(TM)i5-2430M processor of frequency 2.4GHz×4, and 5.7GiB memory. The results, synthesised in Table 1, show the potential of our method in terms of

accuracy (no false positives) and scalability. For larger numbers of components, the size of the resulting invariants was not problematic for Z3. However, it may be the case that history clocks considerably increase the size of the generated formulae. It can also be observed that Uppaal being highly optimised, it has better scores on the first example in particular and on smaller systems in general. The timings for our prototype are obtained with the `Unix` command `time` while the results for Uppaal come from the command `verifyta` which comes with the Uppaal 4.1.14 distribution.

Model & Property	Size	Time/Space	
		$\mathcal{E}^* \wedge \mathcal{S}$	Uppaal
Train Gate Controller & mutual exclusion	1*	0m0.156s/2.6kB+140B	0ms/8 states
	2	0m0.176s/3.2kB+350B	0ms/13 states
	64	0m4.82s/530kB+170kB	0m0.210s/323 states
	124	0m17.718s/700kB+640kB	0m1.52s/623 states
Fischer & mutual exclusion	2*	0m0.144/3kB	0m0.008s/14 states
	4*	0m0.22s/6.5kB	0m0.012s/156 states
	6*	0m0.36s/12.5kB	0m0.03s/1714 states
	14*	0m2.840s/112kB	no result in 4 hours
Temperature Controller & absence of deadlock	1*	0m0.172s/840B+60B	0m0.01s/4 states
	8	0m0.5s/23kB+2.4kB	11m0.348s/57922 states
	16	0m2.132s/127kB+9kB	no result in 6 hours
	124	0m19.22s/460kB+510kB	idem

Table 1: Results from Experiments. The marking "*" highlights the cases when $\mathcal{E}$ alone was enough to prove the property. The expressions of form "$x + y$" are to be read as "the formula $\wedge_i CI(B_i) \wedge II(\gamma) \wedge \mathcal{E}(\gamma)$, resp. $\mathcal{E}^*(\gamma) \wedge \mathcal{S}(\gamma)$, has length x, resp. y".

5 Conclusion and Future Work

We presented a fully automated compositional method to generate global invariants for timed systems described as parallel compositions of timed automata components using multi-party interactions. The soundness of the method proposed has been proven. In addition, it has been implemented and successfully tested on several benchmarks. The results show that our method may outperform existing exhaustive exploration-based techniques for large systems, thanks to the use of compositionality and over-approximations.

This work is currently being extended in several directions. First, in order to achieve a better integration within D-Finder tool [9] and the Real-Time BIP framework [1] we are working on handling *urgencies* [6] on transitions. Actually, urgencies provide an alternative way to constrain time progress, which is more intuitive to use by programmers but much difficult to handle in a compositional way. A second extension concerns the development of heuristics to reduce the size of the generated invariants. As an example, symmetry-based reduction is potentially interesting for systems containing identical, replicated components. Finally, we are considering specific extensions to particular classes of timed systems and properties, in particular, for schedulability analysis of systems with mixed-critical tasks.

References

1. T. Abdellatif, J. Combaz, and J. Sifakis. Model-based implementation of real-time applications. In *EMSOFT*, 2010.
2. L. D. Alfaro, T. A. Henzinger, and M. Stoelinga. Timed interfaces. In *EMSOFT*, 2002.
3. R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 1994.
4. L. Astefanoaei, S. B. Rayana, S. Bensalem, M. Bozga, and J. Combaz. Compositional invariant generation for timed systems. Technical Report TR-2013-5, Verimag Research Report, 2013.
5. B. Badban, S. Leue, and J.-G. Smaus. Automated invariant generation for the verification of real-time systems. In *WING@ETAPS/IJCAR*, 2010.
6. A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in BIP. In *SEFM*, 2006.
7. G. Behrmann, A. David, K. G. Larsen, J. Håkansson, P. Pettersson, W. Yi, and M. Hendriks. UPPAAL 4.0. In *QEST*, 2006.
8. J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi. Partial order reductions for timed systems. In *CONCUR*, 1998.
9. S. Bensalem, M. Bozga, J. Sifakis, and T.-H. Nguyen. Compositional verification for component-based systems and application. In *ATVA*, 2008.
10. J. Berendsen and F. W. Vaandrager. Compositional abstraction in real-time model checking. In *FORMATS*, 2008.
11. S. Bornot and J. Sifakis. An algebraic framework for urgency. *Information and Computation*, 1998.
12. P. Bouyer. Forward analysis of updatable timed automata. *Form. Methods Syst. Des.*, 2004.
13. M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A model-checking tool for real-time systems. In *CAV*, 1998.
14. C. Courcoubetis and M. Yannakakis. Minimum and maximum delay problems in real-time systems. *Formal Methods in System Design*, 1992.
15. A. David, K. G. Larsen, A. Legay, M. H. Møller, U. Nyman, A. P. Ravn, A. Skou, and A. Wasowski. Compositional verification of real-time systems using Ecdar. *STTT*, 2012.
16. F. S. de Boer, U. Hannemann, and W. P. de Roever. Hoare-style compositional proof systems for reactive shared variable concurency. In *FSTTCS*, 1997.
17. A. Fietzke and C. Weidenbach. Superposition as a decision procedure for timed automata. *Mathematics in Computer Science*, 2012.
18. G. Gardey, D. Lime, M. Magnin, and O. H. Roux. ROMEO: A tool for analyzing time petri nets. In *CAV*, 2005.
19. T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Inf. Comput.*, 1994.
20. L. Lamport. A fast mutual exclusion algorithm. *ACM Trans. Comput. Syst.*, 1987.
21. A. Legay, S. Bensalem, B. Boyer, and M. Bozga. Incremental generation of linear invariants for component-based systems. In *ACSD*, 2013.
22. S.-W. Lin, Y. Liu, P.-A. Hsiung, J. Sun, and J. S. Dong. Automatic generation of provably correct embedded systems. In *ICFEM*, 2012.
23. R. B. Salah, M. Bozga, and O. Maler. Compositional timing analysis. In *EMSOFT*, 2009.
24. S. Tripakis. Verifying progress in timed systems. In *ARTS*, 1999.
25. F. Wang. Redlib for the formal verification of embedded systems. In *ISoLA*, 2006.