

COMPOSITIONAL VERIFICATION FOR TIMED SYSTEMS BASED ON AUTOMATIC INVARIANT GENERATION *

SOUHA BEN RAYANA ^a, LĂCRĂMIOARA AȘTEFĂNOAEI ^b, SADDEK BENSALEM ^c,
MARIUS BOZGA ^d, AND JACQUES COMBAZ

^{a--d} Univ. Grenoble Alpes, VERIMAG, F-38000 Grenoble
e-mail address: {Souha.BenRayana,lastefan,Saddek.Bensalem}@imag.fr

^{c,d} CNRS, VERIMAG, F-38000 Grenoble, France
e-mail address: {Marius.Bozga,Jacques.Combaz}@imag.fr

ABSTRACT. We propose a method for compositional verification to 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 in the RTD-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 [8, 16, 35, 25] 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

2012 ACM CCS: [**Software and its engineering**]: Software organization and properties—Software functional properties—Formal methods—Software verification; [**Theory of computation**]: Semantics and reasoning—Program reasoning—Program verification.

Key words and phrases: compositional verification, timed automata, invariants, component invariants, interaction invariants, interactions.

* Research supported by the European Integrated Project 257414 ASCENS and ICT Collaborative Project 288175 CERTAINTY.

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 [18, 30].

In this paper, we propose a different approach for exploiting compositionality for analysis of timed systems. The driving principle is to use invariants as approximations to exact reachability analysis, the default technique in model-checking. We show that rather precise invariants can be computed compositionally, from the separate analysis of the components in the system and from their composition glue. This method is proved to be sound for the verification of safety state properties. However, it is not complete.

The starting point is the verification method of [12], 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 φ 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 invariant $II(\gamma)$. The connection between the invariants and the system property φ can be intuitively understood as follows: if φ can be proved to be a logical consequence of the conjunction of components and interaction invariants, then φ holds for the system.

$$\frac{\vdash (\bigwedge_i CI(B_i)) \wedge II(\gamma) \rightarrow \varphi}{\parallel_{\gamma} B_i \models \square \varphi} \quad (VR)$$

Figure 1: Compositional verification

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 “ $\parallel_{\gamma} B_i \models \square \varphi$ ” is to be read as “ φ holds in every reachable state of $\parallel_{\gamma} B_i$ ”.

The verification rule (VR) in [12] 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 [12]. Intuitively, this is because the proposed invariants are too weak to infer cross constraints relating the clocks of the controller and those of the workers: interaction invariants $II(\gamma)$

relates only locations of components and thus at most eliminates unreachable configurations like $(lc_1, \dots, l_{2i}, \dots)$, while the component invariants can only state local conditions on clocks such as $x \leq 4$ at lc_1 . Using history clocks allows to recover additional constraints. For example, after the controller returns from lc_2 to lc_1 for the first time, whenever it reaches lc_1 again, 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 sufficient to prove the deadlock freedom property.

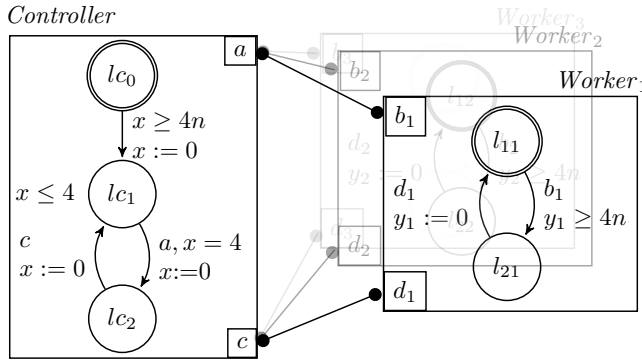


Figure 2: A timed system

Organisation of the paper. This paper is essentially an extended version of the conference paper [5]. The extension is threefold with respect to (1) incorporating proofs, (2) detailing technicalities about handling initial states, and (3) formalising three heuristics to speed up and simplify invariant generation. 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 heuristics while Section 5 shows their use in the case studies we experimented with in our implementation. Section 6 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, slightly adapted to embrace a uniform notation throughout the paper.

Definition 2.1 (Syntax). A component is a timed automaton $(L, A, \mathcal{X}, T, \text{tpc}, s_0)$ where L is a finite set of locations, A a finite set of actions, \mathcal{X} is a finite set of local¹ clocks, $T \subseteq L \times (A \times \mathcal{C} \times 2^{\mathcal{X}}) \times L$ is a set of edges labelled with an action, a guard, and a set of clocks to be reset, $\text{tpc} : L \rightarrow \mathcal{C}$ assigns a time progress condition² to each location. \mathcal{C} is the

¹Locality is essential for avoiding side effects which would break compositionality and local analysis.

²To avoid confusion with invariant properties, we prefer to adopt the terminology of “time progress condition” from [14] instead of “location invariants”.

set of clock constraints and $s_0 \in L \times \mathcal{C}$ provides the initial configuration. A clock constraint is defined by the grammar:

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

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 $\mathbf{v} : \mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$. For a clock constraint C , $\mathbf{v} \models C$ 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 2.2 (Semantics). The semantics of a component $B = (L, A, \mathcal{X}, T, \text{tpc}, s_0)$ is given by the labelled transition system (Q, A, \rightarrow, Q_0) where $Q \subseteq L \times \mathbf{V}$ denotes the states of B , $\rightarrow \subseteq Q \times (A \cup \mathbb{R}_{\geq 0}) \times Q$ denotes the transitions according to the rules:

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

and $Q_0 = \{(l_0, \mathbf{v}_0) \mid s_0 = (l_0, c_0) \wedge c_0(\mathbf{v}_0)\}$ denotes the initial states.

Because the semantics defined above is in general infinite, we work with the so called zone graph [27] 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. The initial configuration $s_0 = (l_0, c_0)$ corresponds trivially to a symbolic state (l_0, ζ_0) . Given a symbolic state (l, ζ) , its successor with respect to a transition t of B is denoted as $\text{succ}(t, (l, \zeta))$ and defined by means of its timed and its discrete successor:

- $\text{time_succ}((l, \zeta)) = (l, \nearrow \zeta \cap \text{tpc}(l))$
- $\text{disc_succ}(t, (l, \zeta)) = (l', (\zeta \cap g)[r] \cap \text{tpc}(l'))$ if $t = (l, (-, g, r), l')$
- $\text{succ}(t, (l, \zeta)) = \text{norm}(\text{time_succ}(\text{disc_succ}(t, (l, \zeta))))$

where $\nearrow, [r], \text{norm}$ 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 ; $\text{norm}(\zeta)$ corresponds to normalising ζ such that all bounds on clocks and clock differences are either bounded by some finite value or infinite. Since our use of invariants is only as 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 [10, 15] for more precise definitions.

A symbolic execution of B is a sequence of symbolic states s_0, \dots, s_i, \dots ³ such that for any $i > 0$, there exists a transition t for which s_i is $\text{succ}(t, s_{i-1})$. The set of reachable symbolic states of B is $\text{Reach}_B(s_0)$ where Reach_B is defined recursively as:

$$\text{Reach}_B(s) = \{s\} \cup \bigcup_{t \in T} \text{Reach}_B(\text{succ}(t, s))$$

for an arbitrary s and T the set of transitions in B . We remind that the set $\text{Reach}_B(s_0)$ can be shown finite knowing that the number of normalised zones is finite. In general, the symbolic zone graph provides an over-approximation of the set of reachable states. This over-approximation is exact only for timed automata without diagonal constraints [10, 15].

³We tacitly assume that s_0 is such that $s_0 = \text{time_succ}(s_0)$. If this is not the case, one can always consider $\text{time_succ}(s_0)$ instead of s_0 for the definition of symbolic executions and reachable states.

In our framework, components communicate by means of *interactions*, which are synchronisations between actions. Given n components $(B_i)_{i=1,\dots,n}$, with disjoint sets of actions A_i , an interaction is a subset $\alpha \subseteq \cup_i A_i$ containing at most one action per component. We denote interactions α as sets $\{a_i\}_{i \in I}$, with $a_i \in A_i$ for all $i \in I \subseteq \{1, \dots, n\}$. For readability, in examples, we use the alternative notation $(a_1 \mid a_2 \mid \dots \mid a_i)$ instead. Given a set of interactions γ , we denote by $Act(\gamma)$ the set of actions involved in γ , that is, $Act(\gamma) = \cup_{\alpha \in \gamma} \alpha$.

Definition 2.3 (Timed System). For a given n and $i \in \{1, \dots, n\}$ let $B_i = (L_i, A_i, \mathcal{X}_i, T_i, \mathbf{tpc}_i, s_{0i})$ be n components with disjoint sets of actions and initial states $s_{0i} = (l_{0i}, c_{0i})$. Let γ be a set of interactions constructed from $\cup_i A_i$. The *timed system* $\parallel_{\gamma} B_i$ is defined as the component $(L, \gamma, \mathcal{X}, T_{\gamma}, \mathbf{tpc}, s_0)$ where $L = \times_i L_i$, $\mathcal{X} = \cup_i \mathcal{X}_i$, $\mathbf{tpc}(\bar{l}) = \bigwedge_i \mathbf{tpc}(l_i)$, $s_0 = ((l_{01}, \dots, l_{0n}), \bigwedge_i c_{0i})$ and

$$T_{\gamma} = \left\{ (\bar{l}, (\alpha, g, r), \bar{l}') \mid \begin{array}{l} \bar{l} = (l_1, \dots, l_n) \in L, \bar{l}' = (l'_1, \dots, l'_n) \in L \\ \alpha = \{a_i\}_{i \in I} \in \gamma, \forall i \in I. (l_i, (a_i, g_i, r_i), l'_i) \in T_i, \forall i \notin I. l_i = l'_i \\ g = \bigwedge_{i \in I} g_i, r = \bigcup_{i \in I} r_i \end{array} \right\}$$

In the timed system $\parallel_{\gamma} B_i$, 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 $a_j \in \alpha$ ⁴. This corresponds to the usual notion of multi-party interaction. We note that interactions can only restrict the behaviour of components, i.e., the states reached by B_i in $\parallel_{\gamma} B_i$ belong to $Reach_{B_i}(s_{0i})$. This is a property which is exploited in the verification rule (VR) in Figure 1.

To give a logical characterisation of components and their properties, we use invariants. An invariant Φ is a state predicate which holds in every reachable state of B , in symbols, $B \models \Box \Phi$. We use $CI(B)$ and $II(\gamma)$, to denote **component**, respectively **interaction invariants**. For component invariants, our choice is to work with their reachable symbolic set. More precisely, for component B , its associated component invariant $CI(B)$ is the disjunction of $(l \wedge \zeta)$ for all symbolic states (l, ζ) in $Reach_B(s_0)$. To ease the reading, we abuse of 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 example in Figure 2 with one worker are:

$$\begin{aligned} CI(Contoller) &= (lc_0 \wedge x \geq 0) \vee (lc_1 \wedge 4 \geq x \geq 0) \vee (lc_2 \wedge x \geq 0) \\ CI(Worker_1) &= (l_{11} \wedge y_1 \geq 0) \vee (l_{21} \wedge y_1 \geq 4). \end{aligned}$$

The interaction invariants are computed by the method explained in [12]. Interaction invariants are over-approximations of the global state space allowing us to disregard certain tuples of local states as unreachable. As an illustration, consider the interactions invariant for the running example when the controller is interacting with one worker:

$$II(\{(a \mid b_1), (c \mid d_1)\}) = (l_{11} \vee lc_2) \wedge (l_{21} \vee lc_0 \vee lc_1).$$

The invariant is given in conjunctive normal form to stick to the formalism in [12, 11]. Every disjunction corresponds to the so called notion of “initially marked traps” in an underlying Petri net associated to our model. Intuitively, a trap in Petri nets is a set of places which always contains tokens if they have tokens initially.

⁴To simplify the notation, we omit unary interactions and the actions for transitions involved in them. For example, in Figure 2, the initial transition in *Contoller* does not have an explicit action associated.

We note that the proposed⁵ component and interaction invariants are inductive invariants. A state predicate is called *inductive* for a component or system B if, whenever it holds for a state s of B it equally holds for any of its successors s' . That is, the validity of an inductive predicate is preserved by executing any transition, timed or discrete. An inductive predicate which moreover holds at initial states is an (inductive) invariant. Trivially, such a predicate holds in all reachable states.

As for **component properties**, we are interested in arbitrary invariant state properties that can be expressed as boolean combinations of “ $at(l)$ ” predicates and clock constraints. Invariant properties include generic properties such as mutual exclusion, absence of deadlock, unreachability of “bad” states, etc. As a simple illustration consider the property $lc_1 \rightarrow \bigvee_i (y_i - x \geq 4n - 4)$, discussed for our running example introduced in Section 1. As a more sophisticated example, consider *absence of deadlock*. Intuitively, a timed system with a set of interactions γ is *deadlocked* when no interaction in γ is enabled. Absence of deadlock is therefore expressed as the disjunction $\bigvee_{\alpha \in \gamma} \text{enabled}(\alpha)$. As for the enabledness predicate, we borrow it from [34] where it is essentially constructed from the syntactic definition of the timed system. More precisely, for an interaction α , $\text{enabled}(\alpha)$ is $\bigvee_t \text{enabled}(t)$, with t being a transition triggered by α . In turn, for $t = (\bar{l}, (\alpha, g, r), \bar{l}')$, $\text{enabled}(t)$ is defined using elementary operations on zones as $\bar{l} \wedge \swarrow (g \cap [r] \text{tpc}(\bar{l}') \cap \text{tpc}(\bar{l}))$, where $\swarrow \zeta$ is the backward diagonal projection of ζ , $[r]\zeta$ is the set of valuations \mathbf{v} such that $\mathbf{v}[r]$ is in ζ .

3. TIMED INVARIANT GENERATION

As explained in the introduction, a direct application of the compositional verification rule (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 such 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 to timing aspects – to constrain the bounds on the differences between clocks in different components. To give a concrete illustration, consider the property $\varphi_{\text{Safe}} = (lc_1 \wedge l_{11} \rightarrow x \leq y_1)$ that holds in the running example with one worker. We note that if this property is satisfied, it is guaranteed that the global system is not deadlocked when the controller is at location lc_1 and the worker is at location l_{11} . It is not difficult to see that φ_{Safe} cannot be deduced from $CI(\text{Controller}) \wedge CI(\text{Worker}_1) \wedge II(\{(a \mid b_1), (c \mid d_1)\})$ as no relation can be established between x and y_1 .

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. 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.

⁵The rule (VR) is generic enough to work with other types of invariants. For example, one could use any over-approximation of the reachable set in the case of component invariants, however, this comes at the price of losing precision.

Definition 3.1 (Components with History Clocks). Given component $B = (L, A, \mathcal{X}, T, \text{tpc}, s_0)$, its extension with history clocks is the component $B^h = (L, A, \mathcal{X} \cup \mathcal{H}_A, T^h, \text{tpc}, s_0^h)$ where

- $\mathcal{H}_A = \{h_0\} \cup \{h_a \mid a \in A\}$ is the set of history clocks,
- $T^h = \{(l, (a, g, r \cup \{h_a\}), l') \mid (l, (a, g, r), l') \in T\}$,
- $s_0^h = (l_0, c_0^h)$, where $c_0^h = (c_0 \wedge h_0 = 0 \wedge \bigwedge_{a \in A} h_a > 0)$, given $s_0 = (l_0, c_0)$.

The clock h_0 measures the time from the initialisation. This clock equals 0 in s_0^h and is never tested or reset. Due to this very restricted use, the same clock h_0 can be consistently used (shared) by all components B^h and consequently, allows to capture clock constraints derived from the common system initialisation time.

Every history clock h_a measures the time passed from the last occurrence of action a . These history clocks are initially strictly greater than 0 and are reset when the corresponding action is executed. As a side effect, whenever h_a is strictly bigger than h_0 , we can infer that the action a has not been (yet) executed. This initialisation scheme allows a more refined analysis precisely because we can distinguish between actions which were executed and those which were not.

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 extended model of B^h corresponds to an invariant of original component. By abuse of notation, given set of actions $A = \{a_1, \dots, a_m\}$ use $\exists \mathcal{H}_A$ to stand for $\exists h_{a_1} \exists h_{a_2} \dots \exists h_{a_m} \exists h_0$.

Proposition 3.2.

- (1) If Φ^h is an invariant of B^h then $\Phi = \exists \mathcal{H}_A. \Phi^h$ is an invariant of B .
- (2) If Φ^h is an invariant of B^h and Ψ^h an inductive assertion of B^h expressed on history clocks $\mathcal{H}_A \setminus \{h_0\}$ then $\Phi = \exists \mathcal{H}_A. (\Phi^h \wedge \Psi^h)$ is an invariant of B .

Proof. (1) It suffices to notice that any symbolic state (l, ζ^h) in the reachable set $\text{Reach}_{B^h}(s_0^h)$ corresponds to a symbolic state (l, ζ) in the reachable set $\text{Reach}_B(s_0)$ such that ζ is the projection of ζ^h to clocks in \mathcal{X} , that is $\zeta \equiv \exists \mathcal{H}_A. \zeta^h$. Henceforth, $\exists \mathcal{H}_A. \text{Reach}_{B^h}(s_0^h) \equiv \text{Reach}_B(s_0)$. Moreover, for any invariant Φ^h of B^h it holds $\exists \mathcal{H}_A. \text{Reach}_{B^h}(s_0^h) \subseteq \exists \mathcal{H}_A. \Phi^h$. By combining the two facts, we obtain that Φ is an invariant of B .

(2) Consider the modified component with history clocks B_Ψ^h defined as B^h but with initial configuration $(l_0, c_0^h \wedge \Psi^h)$. This initial configuration is valid, as Ψ^h constrain exclusively clocks in \mathcal{H}_A whereas c_0^h leaves all of them unconstrained. Now, it can be easily shown that $\Phi^h \wedge \Psi^h$ is an invariant of B_Ψ^h . Then, following the same reasoning as for point (1) we obtain that $\exists \mathcal{H}_A. (\Phi^h \wedge \Psi^h)$ is an invariant of B . \square

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 the remaining ones preserve their previous values, thus they are greater than or equal to those being reset. This basic observation is exploited in the following definition, which builds, recursively, all the inequalities that could hold given an interaction set γ .

Definition 3.3 (Interaction Inequalities for History Clocks). Given an interaction set γ , we define the following interaction inequalities $\mathcal{E}(\gamma)$:

$$\mathcal{E}(\gamma) = \bigvee_{\alpha \in \gamma} \left(\bigwedge_{\substack{a_i, a_j \in \alpha \\ a_k \in \text{Act}(\gamma \ominus \alpha)}} h_{a_i} = h_{a_j} \leq h_{a_k} \right) \wedge \mathcal{E}(\gamma \ominus \alpha).$$

where $\gamma \ominus \alpha = \{\beta \setminus \alpha \mid \beta \in \gamma \wedge \beta \not\subseteq \alpha\}$ and $\mathcal{E}(\emptyset) = \text{true}$.

The mechanism of history clocks is as follows. When an interaction α takes place, the history clocks h_a associated to any action $a \in \alpha$ are reset. Thus they are all equal and smaller than any other clocks and measure the time passed from the last occurrence of a .

The operation $\gamma \ominus \alpha$ eliminates in any interaction β the actions from α . As an illustration, for $\beta = (a \mid a_1 \mid a_2)$, $\alpha = (a_1 \mid a_2)$, $\gamma = \{\alpha, \beta\}$, $\gamma \ominus \alpha = \{a\}$.

We can use the interpreted function “min” as syntactic sugar to have a slightly more compact expression for $\mathcal{E}(\gamma)$ as follows:

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

As an example, 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 is:

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

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

Proposition 3.4. $\mathcal{E}(\gamma)$ is an inductive predicate of $\|_{\gamma} B_i^h$.

Proof. Assume $\mathcal{E}(\gamma)$ holds in some arbitrary state s of $\|_{\gamma} B_i^h$. We have two categories of successor states for s , namely time successors and discrete successors. Obviously $\mathcal{E}(\gamma)$ holds for all time successors s' , as all clocks progress uniformly and henceforth all the relations between them are preserved. Let now s' be a discrete successor of s by an arbitrary interaction α . As all the history clocks for actions in α have just been reset, s' satisfies

$$\bigwedge_{\substack{a_i, a_j \in \alpha \\ a_k \in \text{Act}(\gamma \ominus \alpha)}} 0 = h_{a_i} = h_{a_j} \leq h_{a_k} \quad (3.1)$$

To conclude the proof, we need to show that moreover, for the remaining clocks of actions in $\text{Act}(\gamma \ominus \alpha)$, they satisfy $\mathcal{E}(\gamma \ominus \alpha)$ in s' . Actually, we can show the additional fact that for any set of interactions γ and for any interaction α the implication $\mathcal{E}(\gamma) \rightarrow \mathcal{E}(\gamma \ominus \alpha)$ is valid in any reachable state. This fact can be simply proven by induction on the size of the set interactions γ following the definition of \mathcal{E} . Consequently, assuming that $\mathcal{E}(\gamma)$ holds at s , it follows that $\mathcal{E}(\gamma \ominus \alpha)$ holds at s . Then $\mathcal{E}(\gamma \ominus \alpha)$ also holds at s' because α does not modify any clock involved in $\gamma \ominus \alpha$ and this concludes the proof. \square

By using Proposition 3.4 and Proposition 3.2, we can safely combine the component and interaction invariants of the system with history clocks with the interaction inequalities. We can eliminate the history clocks from $\bigwedge_i CI(B_i^h) \wedge II(\gamma) \wedge \mathcal{E}(\gamma)$ and obtain an invariant of the original system. This invariant is usually stronger than $\bigwedge_i CI(B_i) \wedge II(\gamma)$ and yields more successful applications of the rule (VR).

Corollary 3.5. $\Phi = \exists \mathcal{H}_A. (\bigwedge_i CI(B_i^h) \wedge II(\gamma) \wedge \mathcal{E}(\gamma))$ is an invariant of $\|\gamma B_i$.

Example 3.6. We reconsider the model of a controller and a worker from Figure 2. We show how the generated invariants are enough to prove the safety property $\varphi_{Safe} = (lc_1 \wedge l_{11} \rightarrow x \leq y_1)$ from Section 1. The invariants for the components with history clocks are computed precisely as illustrated in Section 1, that is, they represent zone graphs:

$$\begin{aligned} CI(Controller^h) = & (lc_0 \wedge x = h_0 < h_a \wedge h_0 < h_c) \vee \\ & (lc_1 \wedge x \leq h_0 - 4 \wedge x \leq 4 \wedge h_0 < h_a \wedge h_0 < h_c) \vee \\ & (lc_1 \wedge x \leq 4 \wedge x = h_c \leq h_a \leq h_0 - 8) \vee \\ & (lc_2 \wedge x \leq h_0 - 8 \wedge h_a = x \wedge h_0 < h_c) \vee \\ & (lc_2 \wedge x = h_a \wedge h_c = h_a + 4 \leq h_0 - 8) \end{aligned}$$

$$\begin{aligned} CI(Worker_1^h) = & (l_{11} \wedge y_1 = h_0 < h_{d_1} \wedge h_0 < h_{b_1}) \vee \\ & (l_{11} \wedge y_1 = h_{d_1} \leq h_{b_1} \leq h_0 - 4) \vee \\ & (l_{21} \wedge h_{b_1} + 4 \leq y_1 = h_0 < h_{d_1}) \vee \\ & (l_{21} \wedge y_1 = h_{d_1} \leq h_0 - 4 \wedge h_{b_1} \leq h_{d_1} - 4) \end{aligned}$$

By using the interaction invariant described in Section 2 and the inequality constraints $\mathcal{E}((a \mid b_1), (c \mid d_1))$, after the elimination of the existential quantifiers in

$$(\exists h_a. \exists h_{b_1}. \exists h_c. \exists h_{d_1}. \exists h_0) CI(Controller^h) \wedge CI(Worker_1^h) \wedge II(\gamma) \wedge \mathcal{E}(\gamma)$$

we obtain the following invariant Φ :

$$\begin{aligned} \Phi = & (l_{11} \wedge lc_0 \wedge \mathbf{x = y_1}) \vee \\ & (l_{11} \wedge lc_1 \wedge (\mathbf{y_1 = x} \vee \mathbf{x + 4 \leq y_1})) \vee \\ & (l_{21} \wedge lc_2 \wedge (\mathbf{y_1 = x + 4} \vee \mathbf{x + 8 \leq y_1})). \end{aligned}$$

We used bold fonts in Φ to highlight relations between x and y_1 which are not in $CI(Controller) \wedge CI(Worker_1) \wedge II(\gamma)$. It can be easily checked now that $\Phi \rightarrow \varphi_{Safe}$ holds and consequently, this proves that φ_{Safe} holds for the system.

To sum up, the basic steps of our invariant generation method described so far are:

- (1) compute the interaction invariant $II(\gamma)$;
- (2) extend the components B_i to components with history clocks B_i^h ;
- (3) compute component invariants $CI(B_i^h)$;
- (4) compute inequality constraints $\mathcal{E}(\gamma)$ for interactions γ ;
- (5) finally, eliminate the history clocks in $\bigwedge_i CI(B_i^h) \wedge II(\gamma) \wedge \mathcal{E}(\gamma)$.

We note that, 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 3.7. If $\gamma = \gamma_1 \cup \gamma_2$ such that $Act(\gamma_1) \cap Act(\gamma_2) = \emptyset$ then $\mathcal{E}(\gamma) \equiv \mathcal{E}(\gamma_1) \wedge \mathcal{E}(\gamma_2)$.

Proof. By induction on the number of interactions in γ . In the base case, γ has a single interaction and the property trivially holds. For the induction step, for the ease of reading, we introduce $eq(\alpha)$ and $leq(\alpha, \gamma)$ to denote respectively $\bigwedge_{a_i, a_j \in \alpha} h_{a_i} = h_{a_j}$ and $\bigwedge_{\substack{a_i \in \alpha \\ a_k \in Act(\gamma \ominus \alpha)}} h_{a_i} \leq h_{a_k}$. $\mathcal{E}(\gamma)$ can be rewritten as follows:

$$\begin{aligned}
\mathcal{E}(\gamma) &= \bigvee_{\alpha \in \gamma_1} eq(\alpha) \wedge leq(\alpha, \gamma) \wedge \mathcal{E}((\gamma_1 \cup \gamma_2) \ominus \alpha) \vee \bigvee_{\alpha \in \gamma_2} eq(\alpha) \wedge leq(\alpha, \gamma) \wedge \mathcal{E}((\gamma_1 \cup \gamma_2) \ominus \alpha) \\
&\quad (\text{using } \gamma_2 \ominus \alpha = \gamma_2 \text{ for } \alpha \in \gamma_1 \text{ and by ind. for } \gamma' = (\gamma_1 \ominus \alpha) \cup \gamma_2) \\
&\equiv \bigvee_{\alpha \in \gamma_1} eq(\alpha) \wedge leq(\alpha, \gamma) \wedge \mathcal{E}(\gamma_1 \ominus \alpha) \wedge \mathcal{E}(\gamma_2) \vee \bigvee_{\alpha \in \gamma_2} eq(\alpha) \wedge leq(\alpha, \gamma) \wedge \mathcal{E}(\gamma_1) \wedge \mathcal{E}(\gamma_2 \ominus \alpha) \\
&\quad (\text{using } \bigvee_{\alpha \in \gamma_i} eq(\alpha) \wedge leq(\alpha, \gamma_i) \wedge \mathcal{E}(\gamma_i \ominus \alpha) = \mathcal{E}(\gamma_i) \text{ for } i \in \{1, 2\}) \\
&\equiv \mathcal{E}(\gamma_1) \wedge \mathcal{E}(\gamma_2) \wedge \left(\bigvee_{\alpha \in \gamma_1} leq(\alpha, \gamma_2) \vee \bigvee_{\alpha \in \gamma_2} leq(\alpha, \gamma_1) \right) \\
&\quad (\text{using totality of "}\leq\text{" and disjointness of } \gamma_i) \\
&\equiv \mathcal{E}(\gamma_1) \wedge \mathcal{E}(\gamma_2)
\end{aligned}$$

The following corollary is an immediate consequence of Proposition 3.7. \square

Corollary 3.8. *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 $\mathcal{E}(\gamma) \equiv \bigwedge_{\alpha \in \gamma} \left(\bigwedge_{a_i, a_j \in \alpha} h_{a_i} = h_{a_j} \right)$.*

The two interactions in $\gamma = \{(a \mid b_1), (c \mid d_1)\}$ are disjoint. Thus, 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 interactions, the presence of conflicting actions enforce disequalities (or separation) constraints between all interactions using them. In what follows, we show a generic way of automatically computing 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 3.9 (System with Interaction History Clocks). Given a timed system $\|_{\gamma} B_i$, its extension with history clocks for interactions is the timed system $B^* \|_{\gamma, h} B_i^h$ where:

- B^* is an auxiliary component $(\{l^*\}, A_{\gamma}, \mathcal{H}_{\gamma}, T, (l^* \mapsto true), (l^*, true))$ where:
 - the set of actions $A_{\gamma} = \{a_{\alpha} \mid \alpha \in \gamma\}$

- the set of interaction history clocks $\mathcal{H}_\gamma = \{h_\alpha \mid \alpha \in \gamma\}$
- the set of transitions $T = \{(l^*, (a_\alpha, true, \{h_\alpha\}), l^*) \mid \alpha \in \gamma\}$
- $\gamma^h = \{(a_\alpha \mid \alpha) \mid \alpha \in \gamma\}$ with $(a_\alpha \mid \alpha)$ denoting $\{a_\alpha\} \cup \{a \mid a \in \alpha\}$.

As before, it can be shown that any invariant of $B^* \parallel_{\gamma^h} B_i^h$ corresponds to an invariant of $\parallel_\gamma B_i$. The history clocks for interactions do not impact the behaviour and henceforth the two systems are bisimilar.

Proposition 3.10.

- (1) If Φ^h is an invariant of $B^* \parallel_{\gamma^h} B_i^h$, then $\Phi = \exists \mathcal{H}_A \exists \mathcal{H}_\gamma. \Phi^h$ is an invariant of $\parallel_\gamma B_i$.
- (2) If Φ^h is an invariant of $B^* \parallel_{\gamma^h} B_i^h$ and Ψ^h an inductive predicate of $B^* \parallel_{\gamma^h} B_i^h$ expressed on history clocks for actions and interactions $\mathcal{H}_\gamma \cup \mathcal{H}_A \setminus \{h_0\}$ then $\Phi = \exists \mathcal{H}_A \exists \mathcal{H}_\gamma. (\Phi^h \wedge \Psi^h)$ is an invariant of $\parallel_\gamma B_i$.

Proof. Similar to Proposition 3.2. □

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 latter 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 3.11 (Separation Constraints for Interaction Clocks). Given an interaction set γ , the induced separation constraints, $\mathcal{S}(\gamma)$, are defined as follows:

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

where $|x|$ denotes the absolute value of x and k_a is a constant computed locally on the component executing a , and representing the minimum elapsed time between two consecutive executions of a .

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

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

Proposition 3.12. *Let*

$$\mathcal{S}^*(\gamma) = \bigwedge_{a \in \text{Act}(\gamma)} \bigwedge_{\substack{\alpha \neq \beta \in \gamma \\ a \in \alpha \cap \beta}} (h_a \leq h_\alpha \leq h_\beta - k_a \vee h_a \leq h_\beta \leq h_\alpha - k_a)$$

We have that:

- (1) $\mathcal{S}^*(\gamma)$ is an inductive predicate of $B^* \parallel_{\gamma^h} B_i^h$.
- (2) The equivalence $\mathcal{S}(\gamma) \equiv \exists \mathcal{H}_A. \mathcal{S}^*(\gamma)$ is a valid formula.

Proof. (1) Let us fix an arbitrary term $S(a, \alpha, \beta)$ defined as

$$S(a, \alpha, \beta) = (h_a \leq h_\alpha \leq h_\beta - k_a \vee h_a \leq h_\beta \leq h_\alpha - k_a)$$

Assume $S(a, \alpha, \beta)$ holds in an arbitrary state s of $B^* \parallel_{\gamma^h} B_i^h$. Then, it obviously holds for any time successors as well as for any discrete successors by interactions not containing the action a . For an interaction involving a , but different than α and β , h_a is reset to zero

whereas h_α and h_β are unchanged. Henceforth, $S(a, \alpha, \beta)$ remains valid as only h_a changes to 0. Let consider the situation α is executed (the case of β is perfectly dual). In this case, both h_a and h_α are reset to 0, whereas h_β is unchanged. Two situations can happen:

- (a) $h_a \leq h_\alpha \leq h_\beta - k_a$ holds in s . Then, obviously, the same holds in s' where h_a and h_α are reset.
- (b) $h_a \leq h_\beta \leq h_\alpha - k_a$ holds in s . This is the interesting case where we need the assumption about the separation time k_a . As consecutive executions of a are separated by k_a , to execute α it must actually hold that $h_a \geq k_a$ in s . Consequently, $h_\beta \geq k_a$ in s , as well as in s' (because h_β does not change from s to s'). Then, knowing that $h_a = h_\alpha = 0$ in s' we have that $h_a \leq h_\alpha \leq h_\beta - k_a$ in s' .

(2) We can equivalently write

$$\begin{aligned} \mathcal{S}^*(\gamma) &\equiv \bigwedge_{a \in \text{Act}(\gamma)} \bigwedge_{\substack{\alpha \neq \beta \in \gamma \\ a \in \alpha \cap \beta}} (h_a \leq h_\alpha \wedge h_a \leq h_\beta \wedge |h_\alpha - h_\beta| \leq k_a) \\ &\equiv \mathcal{S}(\gamma) \wedge \bigwedge_{a \in \text{Act}(\gamma)} \bigwedge_{\substack{\alpha \neq \beta \in \gamma \\ a \in \alpha \cap \beta}} (h_a \leq h_\alpha \wedge h_a \leq h_\beta) \end{aligned}$$

and this concludes our proof. \square

The predicate $\mathcal{S}(\gamma)$ is expressed over history clocks for interactions. Component invariants $CI(B_i^h)$ are however expressed using history clocks for actions. In order to “glue” them together in a meaningful way, we need some tighter connection between action and interaction history clocks. This aspect is addressed by the constraints \mathcal{E}^* defined below.

Definition 3.13 (\mathcal{E}^*). Given an interaction set γ , we define $\mathcal{E}^*(\gamma)$ as follows:

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

By a similar argument as the one in Proposition 3.4, it can be shown that $\mathcal{E}^*(\gamma)$ is an inductive predicate of the extended system $B^* \parallel_{\gamma^h} B_i^h$. Moreover, there exists a tight connection between \mathcal{E} and \mathcal{E}^* as given in Proposition 3.14.

Proposition 3.14.

- (1) $\mathcal{E}^*(\gamma)$ is an inductive predicate of $B^* \parallel_{\gamma^h} B_i^h$.
- (2) The equivalence $\exists \mathcal{H}_\gamma. \mathcal{E}^*(\gamma) \equiv \mathcal{E}(\gamma)$ is a valid formula.

Proof. (1) To see that $\mathcal{E}^*(\gamma)$ is an inductive predicate it suffices to note that the predicate is preserved by time progress transitions and for any discrete action a , there is always an interaction α containing a such that h_a and h_α are both reset in the same time.

(2) The proof follows directly from the definitions of $\mathcal{E}(\gamma)$ and $\mathcal{E}^*(\gamma)$. Consider that

$\gamma = \{\alpha_1, \alpha_2, \dots, \alpha_m\}$. We have the following equivalences:

$$\begin{aligned}
 \exists \mathcal{H}_\gamma. \mathcal{E}^*(\gamma) &\equiv \exists \mathcal{H}_\gamma. \bigvee_{\alpha_{k_1} \prec \alpha_{k_2} \prec \dots \prec \alpha_{k_m}} (h_{\alpha_{k_1}} \leq h_{\alpha_{k_2}} \leq \dots \leq h_{\alpha_{k_m}} \wedge \mathcal{E}^*(\gamma)) \\
 &\quad \text{(by choosing an arbitrary ordering } \prec \text{ on interactions)} \\
 &\equiv \exists \mathcal{H}_\gamma. \bigvee_{\alpha_{k_1} \prec \alpha_{k_2} \prec \dots \prec \alpha_{k_m}} (h_{\alpha_{k_1}} \leq h_{\alpha_{k_2}} \leq \dots \leq h_{\alpha_{k_m}} \wedge \\
 &\quad \bigwedge_{a \in \alpha_{k_1}} (h_a = h_{\alpha_{k_1}}) \wedge \bigwedge_{a \in \alpha_{k_2} \setminus \alpha_{k_1}} (h_a = h_{\alpha_{k_2}}) \wedge \dots \bigwedge_{a \in \alpha_{k_m} \setminus \alpha_{k_1} \dots \alpha_{k_{m-1}}} (h_a = h_{\alpha_{k_m}})) \\
 &\quad \text{(by expanding the definition of } \mathcal{E}^*(\gamma) \text{ along the chosen order)} \\
 &\equiv \exists \mathcal{H}_\gamma. \bigvee_{\alpha_{k_1} \prec \alpha_{k_2} \prec \dots \prec \alpha_{k_m}} (h_{\alpha_{k_1}} \leq h_{\alpha_{k_2}} \leq \dots \leq h_{\alpha_{k_m}} \wedge \bigwedge_{\ell=1}^m \bigwedge_{a \in \alpha_{k_\ell} \setminus \alpha_{k_1} \dots \alpha_{k_{\ell-1}}} (h_a = h_{\alpha_{k_\ell}})) \\
 &\quad \text{(by rewriting to a more compact form)} \\
 &\equiv \bigvee_{\alpha_{k_1} \prec \alpha_{k_2} \prec \dots \prec \alpha_{k_m}} \exists \mathcal{H}_\gamma. (h_{\alpha_{k_1}} \leq h_{\alpha_{k_2}} \leq \dots \leq h_{\alpha_{k_m}} \wedge \bigwedge_{\ell=1}^m \bigwedge_{a \in \alpha_{k_\ell} \setminus \alpha_{k_1} \dots \alpha_{k_{\ell-1}}} (h_a = h_{\alpha_{k_\ell}})) \\
 &\quad \text{(by distributing the existential quantifiers over the disjunction)} \\
 &\equiv \bigvee_{\alpha_{k_1} \prec \alpha_{k_2} \prec \dots \prec \alpha_{k_m}} \bigwedge_{\ell=1}^m \bigwedge_{\substack{a_i, a_j \in \alpha_{k_\ell} \setminus \alpha_{k_1} \dots \alpha_{k_{\ell-1}} \\ a_k \notin \alpha_{k_1} \dots \alpha_{k_\ell}}} (h_{a_i} = h_{a_j} \leq h_{a_k}) \equiv \mathcal{E}(\gamma) \\
 &\quad \text{(by eliminating the existential quantifiers)} \quad \square
 \end{aligned}$$

From Propositions 3.14, 3.10, and 3.12, it follows that $\exists \mathcal{H}_A \exists \mathcal{H}_\gamma. (\bigwedge_i CI(B_i^h) \wedge II(\gamma) \wedge \mathcal{E}^*(\gamma) \wedge \mathcal{S}(\gamma))$ is an invariant of $\|\gamma B_i$. This new invariant is in general stronger than $\exists \mathcal{H}_A. (\bigwedge_i CI(B_i^h) \wedge II(\gamma) \wedge \mathcal{E}(\gamma))$ and it provides better state space approximations for timed systems with conflicting interactions.

Corollary 3.15. $\Phi = \exists \mathcal{H}_A \exists \mathcal{H}_\gamma. (\bigwedge_i CI(B_i^h) \wedge II(\gamma) \wedge \mathcal{E}^*(\gamma) \wedge \mathcal{S}(\gamma))$ is an invariant of $\|\gamma B_i$.

Example 3.16. 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 invariant is:

$$II(\gamma) = (l_{11} \vee l_{c1} \vee l_{c2}) \wedge (l_{12} \vee l_{c1} \vee l_{c2}) \wedge (l_{c2} \vee l_{11} \vee l_{12}) \wedge (l_{c0} \vee l_{c1} \vee l_{21} \vee l_{22})$$

The components invariants are:

$$\begin{aligned}
 CI(\text{Controller}^h) &= (l_{c0} \wedge x = h_0 \wedge h_0 < h_a \wedge h_0 < h_c) \vee \\
 &\quad (l_{c1} \wedge x \leq h_0 - 8 \wedge x \leq 4 \wedge h_0 < h_a \wedge h_0 < h_c) \vee \\
 &\quad (l_{c1} \wedge x \leq 4 \wedge x = h_c \leq h_a \leq h_0 - 12) \vee \\
 &\quad (l_{c2} \wedge x \leq h_0 - 12 \wedge h_a = x \wedge h_0 < h_c) \vee \\
 &\quad (l_{c2} \wedge x = h_a \wedge h_c = h_a + 4 \leq h_0 - 12)
 \end{aligned}$$

$$\begin{aligned}
CI(Worker_i^h) = & (l_{1i} \wedge y_i = h_0 \wedge h_0 < h_{d_i} \wedge h_0 < h_{b_i}) \vee \\
& (l_{1i} \wedge y_i = h_{d_i} \leq h_{b_i} \leq h_0 - 8) \vee \\
& (l_{2i} \wedge y_i \geq h_{b_i} + 8 \leq h_0 < h_{d_i}) \vee \\
& (l_{2i} \wedge y_i = h_{d_i} \leq h_0 - 8 \wedge h_{b_i} \leq h_{d_i} - 8)
\end{aligned}$$

The inequalities for action and interaction history clocks are:

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

By recalling the expression of $\mathcal{S}(\gamma)$ we obtain that:

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

and thus, after quantifier elimination in

$$\exists \mathcal{H}_A \exists \mathcal{H}_\gamma. (CI(Controller^h) \wedge \bigwedge_i CI(Worker_i^h) \wedge II(\gamma) \wedge \mathcal{E}^*(\gamma) \wedge \mathcal{S}(\gamma))$$

after simplification, we obtain the following invariant Φ :

$$\begin{aligned}
\Phi = & (l_{11} \wedge l_{12} \wedge lc_0 \wedge x = y_1 = y_2) \vee \\
& (l_{11} \wedge l_{12} \wedge lc_1 \wedge x \leq 4 \wedge (y_1 = y_2 \geq x + 8 \vee \\
& \quad (y_1 = x \wedge \mathbf{y}_2 - \mathbf{y}_1 \geq 4) \vee \\
& \quad (y_1 \geq x + 8 \wedge y_1 - y_2 \geq 8) \vee \\
& \quad (y_2 = x \wedge \mathbf{y}_1 - \mathbf{y}_2 \geq 4) \vee \\
& \quad (y_2 \geq x + 8 \wedge y_2 - y_1 \geq 8))) \vee \\
& (l_{21} \wedge l_{12} \wedge lc_2 \wedge y_1 \geq x + 8 \wedge ((y_2 \geq x + 4 \wedge |\mathbf{y}_1 - \mathbf{y}_2| \geq 4) \vee \\
& \quad y_2 \geq x + 12)) \vee \\
& (l_{11} \wedge l_{22} \wedge lc_2 \wedge y_2 \geq x + 8 \wedge ((y_1 \geq x + 4 \wedge |\mathbf{y}_1 - \mathbf{y}_2| \geq 4) \vee \\
& \quad y_1 \geq x + 12))
\end{aligned}$$

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.

We conclude the section with a discussion about the computation of the separation constants k_a . A simple but incomplete heuristics to test that a given value k_a is a correct separation constraint for an action a is as follows. Consider all paths connecting two transitions (not necessarily distinct) labelled by a . If on every such path, there exists a clock x which is reset and then tested in a guard $x \geq ct$, with $ct \geq k_a$ then, it is safe to conclude that actually k_a is a correct separation value. Nonetheless, alternative methods to exactly compute k_a have been already proposed in the literature. For details, the interested reader can refer, for instance, to [17] which reduces this computation to finding a shortest path in a weighted graph built from the zone graph associated to the component.

4. IMPROVING (VR) - THREE HEURISTICS

We describe and elaborate on heuristics allowing to strengthen the generated invariants and to reduce the generation time. These heuristics have been successfully applied on our case studies considered later in Section 5.

4.1. Refining conflicting interactions. The initialisation of the history clock h_0 provides a convenient way to express and reason about invariants relating occurrences of various actions and interactions at execution. The assertion $h_\alpha \leq h_0$ has the intuitive meaning that “ α has been executed”. We describe below a new family of invariants providing a finer characterisation for the execution of conflicting interactions and related actions.

We fix a as a potential conflicting action within some component $B = (L, A, T, \mathcal{X}, \text{tpc})$. We define the set of preceding actions $Prec(a)$ as all actions of B that can immediately precede a in an execution, formally $Prec(a) = \{a' \in A \mid \exists l, l', l'' \in L. l \xrightarrow{a'} l', l' \xrightarrow{a} l''\}$. For any two conflicting interactions α_1, α_2 involving a , the following assertion:

$$h_{\alpha_1} \leq h_0 \wedge h_{\alpha_2} \leq h_0 \Rightarrow \bigvee_{a' \in Prec(a)} h_{a'} \leq h_0$$

is an invariant. Intuitively, the assertion states that whenever α_1 and α_2 have both been executed (implying that a has also been executed two or more times), at least one of the preceding actions of a must also have been executed. We remark that the invariant above is rather weak and can be implied by the component invariant $CI(B)$ and the glue invariant \mathcal{E}^* in many situations. In fact, whenever a is an action which *is not enabled at the initial location* of B , the component invariant $CI(B)$ implies that

$$h_a \leq h_0 \Rightarrow \bigvee_{a' \in Prec(a)} h_{a'} \leq h_0.$$

This states that whenever a has been executed, at least one of its preceding actions has been executed as well. Knowing moreover that $h_a = \min_{\alpha \in \alpha} h_\alpha$, we can then infer the invariant above.

Nonetheless, if a is an action that is enabled at the initial location, the newly proposed invariant is stronger and cannot be derived as shown before. In this case, a can be actually executed once while none of its predecessors has been executed yet. The component invariant alone does not relate anymore the execution of a to the execution of its preceding actions. Moreover, the component invariant considers always the last occurrence of a and has no means of distinguishing cases where a has been executed only once or more often. This information can sometimes be re-discovered when interaction history clocks $h_{\alpha_1}, h_{\alpha_2}$ are taken into account, henceforth, leading to the proposed invariant. A concrete illustration is provided later in Section 5.

4.2. Invariant computation using regular expressions. There exist situations where the computation of component invariants can be extremely costly. In particular, for untimed components extended with history clocks, their zone graphs will most likely have an exponential size. In fact, due to history clocks, the zones will record the order of (the last) occurrences of actions, and there could be exponentially many of them, reachable at different locations. We note that, in timed components, clocks *restrict* the dynamics of the components, consequently, it cannot be the case that *all* the orders are possible.

The above observation suggests (and was confirmed by our experiments) that applying the same methodology for computing component invariants (based on the reachability graph of the corresponding components with history clocks) regardless of the components being timed or not leads to large formulae when possibly shorter ones exist.

Example 4.1. Consider the untimed component presented in Figure 3 (left) and its extension with history clocks (right). The entire zone graph reachable from $\langle l_0, \zeta_0 \rangle$, with $\zeta_0 = (h_0 = 0, h_{a,b,c} > 0)$ has 6 symbolic states. Therefore, the component invariant is expressed as a disjunction of 16 terms, 9 of them are related to location l_0 and 7 are related to location l_1 .



Figure 3: An untimed component (left) and its extension with history clocks (right).

We recall that untimed automata have elegant and compact encodings as regular expressions. This basic fact can be exploited in order to provide an alternative computation method for component invariants. More concretely, given an untimed component $B = (L, A, T)$ we show how to automatically compute the invariant describing the relations between the history clocks of B^h at some location ℓ , from the language accepted by B at some designated location ℓ . The first key observation is that only the last occurrence of each action should be retained. This implies that it is safe to abstract, with respect to last occurrences, the regular expression characterising the language accepted at the chosen control location. The second key observation is that, regular expressions in some restricted form, can be used to directly generate less constraints on the history clocks. Our regular expression based method can be therefore summarised as follows:

- (1) construct the regular expression E_ℓ representing the language accepted by B at location ℓ ,
- (2) abstract E_ℓ with respect to the last occurrence retention towards some *restricted form* $E_\ell^\sharp = \sum_i e_i^\sharp$ where, every e_i^\sharp contains each action at most once, and does not contain nested $*$ -operators,
- (3) generate from every e_i^\sharp a characteristic formula on history clocks $\phi(e_i^\sharp)$ and obtain as invariant for B the assertion $\ell \Rightarrow \bigvee_i \phi(e_i^\sharp)$.

The first step is well known for finite automata and will not be detailed here. For the second abstraction step, the key ingredients are the simplification rules in Figure 4.

Rule 1 [Last Occurrence Retention]:	$E \cdot a \longrightarrow (E \setminus a) \cdot a$
Rule 2 [Back-unfolding]:	$E^* \longrightarrow (E^* \cdot E) + \varepsilon$

Figure 4: Simplification Rules

Rule 1 eliminates all but the last occurrence of the trailing a symbol from a regular expression of the form $E \cdot a$. The “ \setminus ” denotes a syntactic *elimination operator* defined structurally on expressions as follows. Let a and x be two symbols and E , E_1 and E_2 be arbitrary regular expressions.

$$\begin{aligned} \epsilon \setminus a &= \epsilon \\ x \setminus a &= \begin{cases} \epsilon & \text{if } x = a \\ x & \text{if } x \neq a \end{cases} \\ (E_1 + E_2) \setminus a &= (E_1 \setminus a) + (E_2 \setminus a) \\ (E_1.E_2) \setminus a &= (E_1 \setminus a).(E_2 \setminus a) \\ E^* \setminus a &= (E \setminus a)^* \end{aligned}$$

Rule 2 simply unfolds $*$ -expressions once. By using this rule and other basic manipulation of regular expressions, further simplification opportunities for Rule 1 are enabled.

Example 4.2. Let us consider again the example presented in Figure 3. The language accepted at l_1 is defined as $(a + bc^*b)^*bc^*$. This expression is progressively abstracted into the restricted form as follows:

$$\begin{aligned} (a + bc^*b)^*bc^* &\rightsquigarrow (a + c^*)^*bc^* && \text{(by Rule 1)} \\ &\equiv (a + c^*)^*b(c^*c + \epsilon) && \text{(by Rule 2)} \\ &\equiv (a + c^*)^*bc^*c + (a + c^*)^*b && \text{(by splitting the last +)} \\ &\rightsquigarrow (a + \epsilon)^*bc + (a + c^*)^*b && \text{(by Rule 1)} \\ &\equiv a^*bc + (a + c)^*b && \text{(by standard transformation)} \end{aligned}$$

In the example above, we have applied the iterative strategy consisting of (1) choosing symbols from right to left and applying Rule 1 until no longer possible and then (2) applying Rule 2 to unfold the rightmost $*$ -expression and split the incoming $+$. It can be shown that such a strategy always terminates with expressions in the restricted form. Intuitively, what happens is that Rule 2 splits larger expressions into smaller ones and, further, for each of these Rule 1 eliminates repetitions of symbols.

For the third step, we construct from a regular expression e^\sharp in restricted form an equivalent formula $\phi(e^\sharp)$ on history clocks. This formula represents *exactly* the set of orders on actions (the strings) encoded by the regular expression:

$$\phi(e^\sharp) \equiv \bigvee_{\substack{a_1 \dots a_n \in L(e^\sharp) \\ \text{distinct } a_1, \dots, a_n}} (h_0 \geq h_{a_1} \geq \dots \geq h_{a_n} \wedge \bigwedge_{c \neq a_1, \dots, a_n} h_c > h_0)$$

where $L(e^\sharp)$ is the language of e^\sharp . We note that since we only consider words with distinct symbols, they are finitely many and the disjunction is finite as well.

As an illustration, let e^\sharp be the regular expression in the restricted form $a^*bc + (a + c)^*b$ obtained in Example 4.2. The finite words on which $\phi(e^\sharp)$ builds upon are abc and bc (from

a^*bc) and acb, cab, cb, ab, b from $(a + c)^*b$. By applying the above encoding, we obtain:

$$\begin{aligned}
(h_0 \geq h_a \geq h_b \geq h_c) \vee (h_a > h_0 \geq h_b \geq h_c) &\vee && \text{(corr. to } abc, \text{ resp. } bc) \\
(h_0 \geq h_a \geq h_c \geq h_b) \vee (h_0 \geq h_c \geq h_a \geq h_b) &\vee && \text{(corr. to } acb, \text{ resp. } cab) \\
(h_a > h_0 \geq h_c \geq h_b) \vee (h_c > h_0 \geq h_a \geq h_b) &\vee && \text{(corr. to } cb, \text{ resp. } ab) \\
(h_0 \geq h_b \wedge h_c, h_a > h_0) &&& \text{(corr. to } b)
\end{aligned}$$

Such encodings are, in fact, invariants. Intuitively, the inequalities in $\phi(e^\sharp)$ reflect precisely the order in which the last action occurrences have taken place.

Proposition 4.3. *Let B be an untimed component, E_l the regular expression characterising the language accepted by B at location l , and E_l^\sharp be the result of applying the simplification rules. We have that $\bigvee_l (l \wedge \phi(E_l^\sharp))$ is an invariant of B^h .*

Proof. (sketch) The local component invariant at some location l is precisely characterised by the orders of the last occurrences of actions on traces reaching l . To show that these orders are captured by $\phi(E_l^\sharp)$, it suffices to note that, on the one hand, E_l and E_l^\sharp preserve the language of the last occurrences of actions. This follows from the simplification rules. As for regular expressions e^\sharp in restricted form we can prove the following property. For every word w in $L(e^\sharp)$, the restricted sub-word w_{loc} obtained from w by removing all but last occurrences of every symbol belongs to $L(e^\sharp)$ as well. Henceforth, one can enumerate over all last occurrence words w_{loc} by simply considering all accepted words of $L(e^\sharp)$ having distinct symbols. To conclude the proof we only need to note that the inequalities in $\phi(E_l^\sharp)$ encode the enumeration of all possible words corresponding to traces of B^h ending at l . \square

We can exploit the structure of regular expressions in restricted form to optimise the technique described above even further. To illustrate this, we consider the regular expression $(b_1 + \dots + b_m)^*a_1\dots a_n$ in restricted form (whenever $a_1, \dots, a_n, b_1, \dots, b_m$ are distinct). The corresponding formula on history clocks is

$$h_0 \geq h_{a_1} \geq \dots \geq h_{a_n} \wedge h_{b_1} \geq h_{a_1} \wedge \dots \wedge h_{b_m} \geq h_{a_1} \wedge \bigwedge_{c \neq a_i, b_j} h_c > h_0.$$

The first part encodes the ordering constraints on the *mandatory* string $a_1\dots a_n$. All these actions occur (consequently, their history clocks are smaller than h_0) in this precise order. The second part considers constraints on occurrences of b_j actions, which are *optional*: if some occur, their executions are unconstrained by each other, however, they take place before a_1 . Finally, the last part deals with actions c which do not appear in the regular expression. For all of them, their history clocks should be strictly greater than h_0 . We remark that, for this particular example, the obtained formula has linear size with respect to the size of the regular expression. In contrast, the number of strings encoded (i.e., whenever restricted to last occurrences of symbols) is exponential, with respect to the number of b actions. The construction above can be generalised for arbitrary restricted regular expressions without much difficulty. The resulting formula remains of polynomial size (at worst quadratic) with respect to the size of the restricted regular expression provided as input.

Example 4.4. Following the approach described above, the regular expression in the restricted form $a^*bc + (a + c)^*b$ translates into:

$$(h_0 \geq h_b \geq h_c \wedge h_a \geq h_b) \vee (h_0 \geq h_b \wedge h_a \geq h_b \wedge h_c \geq h_b)$$

We note this expression is significantly smaller, yet logically equivalent to the disjunction of 7 distinct terms corresponding to symbolic zones reached at l_1 as initially presented in Example 4.1.

To sum up, we described a heuristic which can be applied to untimed components to automatically compute an invariant with a reasonable enough size to be handled by existing SMT solvers. Given an untimed component B , our heuristic makes use of the regular expressions characterizing the language accepted by B to avoid a direct construction of the zone graph of B^h which would result in considerably large invariants.

4.3. Exploiting Symmetry. At a closer examination of the definition of separation constraints in Section 3.2, it can be noticed that it characterises all possible orderings of conflicting interactions with respect to permutations. The size of the corresponding search space is exponential in the number of conflicting interactions and this, in turn, may be a bottleneck for the solver. Such situations can and must be avoided especially in the case of symmetric systems. What we show next is how the inherent symmetry in the formula can be eliminated such that the search space becomes considerably smaller.

The use of symmetry has long been addressed, mostly with the intention of making model-checking more feasible and especially in the context of parameterised systems [22, 23, 21, 32]. There the goal is to show the existence of a small cutoff bound which allows the reduction of the verification problem from an arbitrary number of instances to a small, fixed one. Our context is different, that is, breaking the symmetry in some of the generated invariants, for an a priori known number of components.

The types of systems we consider next are formed of a fixed number, be it n , of isomorphic components interacting with a controller, thus the interactions are binary. Isomorphic components are obtained from a generic component B by attaching an index i (from 1 to n) to all symbols in B . The resulting component is denoted by B_i . For any i, j , B_i and B_j are isomorphic⁶. For the ease of reference, we denote systems like $C \parallel_{\gamma}^n B_i$ by the letter M and we use *Exec* to denote the set of their global executions.

In this framework, the notion of symmetry is intrinsically related to permutations. Let Π_n denote the group of permutations of n . The application of permutations is defined on the structure of systems and properties. For a system M as $C \parallel_{\gamma}^n B_i$, and a permutation π , $\pi(M)$ is defined as $C \parallel_{\pi(\gamma)}^n \pi(B_i)$ where $\pi(B_i)$ is defined as $B_{\pi(i)}$ and $\pi(\gamma)$ as $\{\pi(\alpha) \mid \alpha \in \gamma\}$ with $\pi(a_c \mid a_i) = a_c \mid a_{\pi(i)}$ for α an arbitrary binary interaction between an action a_c of C and an action a_i of a B_i . For an execution $\sigma = \alpha_1, \dots, \alpha_i, \dots, \alpha_k$, $\pi(\sigma)$ is defined as $\pi(\alpha_1), \pi(\alpha_2), \dots, \pi(\alpha_i), \dots, \pi(\alpha_k)$. For a global state $s = (s_c, s_1, \dots, s_n)$, $\pi(s)$ is defined as $(s_c, s_{\pi(1)}, \dots, s_{\pi(n)})$. As for system properties φ , we restrict to those built (with the usual logical connectors) from clock constraints and locations, and define:

$$\pi(\varphi) = \begin{cases} x_{\pi(i)} \text{ rop } x_{\pi(j)} & \text{if } \varphi = x_i \text{ rop } x_j \text{ and } \text{rop} \in \{<, \leq, =, >, \geq\} \\ l_{\pi(i)} & \text{if } \varphi = l_i \\ \neg \pi(\varphi_1) & \text{if } \varphi = \neg \varphi_1 \\ \pi(\varphi_1) \text{ op } \pi(\varphi_2) & \text{if } \varphi = \varphi_1 \text{ op } \varphi_2 \text{ and } \text{op} \in \{\wedge, \vee\} \end{cases}$$

where l_i, x_i denote a location, respectively, a clock in B_i .

⁶We note that, by construction, isomorphic components cannot have clock constraints involving indices: any constraint in a worker B_i is obtained from those in B which are oblivious to indices i .

The symmetric systems we consider are symmetric in a “strong” sense, i.e., they are *fully symmetric*. A system M is fully symmetric if for any $\pi \in \Pi_n$, $\pi(M)$ is syntactically identical to M . Similarly, a property φ is fully symmetric if for any permutation π , $\pi(\varphi)$ is equivalent to φ . A property like $l_1 \wedge l_2 \wedge \dots \wedge l_n$ is symmetric. On the contrary, $G = x_1 \leq x_2$ is not as for the permutation $\pi(1) = 2, \pi(2) = 1$, $\pi(G) = x_{\pi(1)} \leq x_{\pi(2)} = x_2 \leq x_1$ which is not equivalent to G .

Symmetric systems have the convenient property that, whenever started in a symmetric state, for any of its executions $\sigma \in Exec$, $\pi(\sigma)$ is itself an execution, that is, $\pi(\sigma) \in Exec$. To see why this is indeed the case, let γ be the interaction set and $\alpha = (a_c \mid a_i)$ an interaction in γ . It suffices to note that if α is possible after σ , then it is also the case for $\pi(\alpha)$ after $\pi(\sigma)$. Note also that, thanks to symmetry, $\pi(\alpha)$ is in γ .

The idea behind simplifying the separation constraints \mathcal{S} is to break the symmetry by replacing the constraints on absolute values $|h_{\alpha_i} - h_{\alpha_j}|$. More precisely, given a conflicting (controller) action a_c , in an execution where interaction $\alpha_i = a_c \mid a_i$ executes before $\alpha_j = a_c \mid a_j$ for $j > i$, we can naturally replace $|h_{\alpha_i} - h_{\alpha_j}|$ by $h_{\alpha_i} - h_{\alpha_j}$. As for an execution which violates this natural ordering (or “canonicity”), we show that we can make use of symmetry to rearrange it. First, we formalise what we mean more precisely by *canonicity*. Given an execution σ and an interaction $\alpha_i = a_c \mid a_i$ we denote by $lpos(\sigma, \alpha_i)$ the last position of α_i in σ . An execution σ is canonical with respect to a_c if $lpos(\sigma, \alpha_i) < lpos(\sigma, \alpha_j)$ for any $i < j$. Let $Exec^c$ be the set of canonical executions. Thanks to symmetry, any execution has a corresponding canonical execution. Assume σ is such that there is a conflicting a_c and for $i > j$ the last occurrence of $\alpha_i = a_c \mid a_i$ appears latter than that of $\alpha_j = a_c \mid a_j$. Let π be such that $\pi(i) = j$ and $\pi(j) = i$. Then $\pi(\sigma)$ is itself an execution and is canonical.

For a canonical execution with a_c being the action of interest \mathcal{S} simplifies to:

$$\mathcal{S}^c(\gamma) = \bigwedge_{\substack{i < j \\ a_c \in \alpha_i \cap \alpha_j}} h_{\alpha_i} - h_{\alpha_j} \geq k_{a_c} \wedge \bigwedge_{\substack{b \neq a_c \\ b \in \beta_i \cap \beta_j}} |h_{\beta_i} - h_{\beta_j}| \geq k_b$$

We note that \mathcal{S}^c reduces \mathcal{S} by $n!$. This is the best we can get in general. However, under particular conditions, \mathcal{S} can be further reduced. For instance, if the controller is such that it considers components one by one and moreover, requires the use of some designated action a_c , then \mathcal{S} further reduces to:

$$\bigwedge_{a \in Act(C)} \bigwedge_{\substack{i < j \\ a \in \alpha_i \cap \alpha_j}} h_{\alpha_i} - h_{\alpha_j} \geq k_{a_c}$$

This is because by considering components one by one, all conflicting interactions involving the controller follow the same order as defined for the designated action a_c . We anticipate and note that such a scenario is the “temperature controller” case study from Section 5.

Finally, we show that for symmetric systems and properties it is correct to consider \mathcal{S}^c instead of \mathcal{S} .

Proposition 4.5. *Let M be a symmetric system, φ be a symmetric property and Φ the global invariant as defined in Section 3.2. We have that if $\vdash \Phi[\mathcal{S} \leftarrow \mathcal{S}^c] \rightarrow \varphi$ then $M \models \square\varphi$.*

Proof. (sketch) It suffices to show that $\vdash \Phi[\mathcal{S} \leftarrow \mathcal{S}^c] \rightarrow \varphi$ iff $\vdash \Phi \rightarrow \varphi$.

“ \Leftarrow ”: trivial. “ \Rightarrow ”: It boils down to show that if φ is an invariant of $Exec^c$ then it is also an invariant of the remaining executions σ in $Exec \setminus Exec^c$. If σ does not have a conflicting

action, we are done, as \mathcal{S}^c is an invariant by default. Else, we make use of the fact that σ has a canonical representation and that φ is symmetric. \square

An immediate application of the above reduction results in the simplification we make use of in the temperature controller example from Section 5. Naturally, the results can be extended also to systems with less symmetry by adapting the standard constructions of automorphisms from, for example, [23]. More precisely, for a system M for which $Aut(M) = \{\pi \mid \pi(M) = M\}$ is a proper subgroup of Π_n , we need to restrict to canonical executions which are consistent with the permutations in $Aut(M)$. However, though such a generalisation is possible, it is not clear if it is also useful: as it is well pointed out in the literature about symmetries, determining $Aut(M)$ is, in itself, a hard problem. This, together with the goal of keeping the presentation as clear as possible, were the reasons why we strictly considered only *fully* symmetric systems.

5. IMPLEMENTATION AND EXPERIMENTS

The method has been implemented in the RTD-Finder tool designed to check safety properties for real-time component-based systems modelled in the RT-BIP language [1]. The tool and the examples are available at <http://www-verimag.imag.fr/RTD-Finder>.

In RT-BIP, components are modelled as timed automata and synchronise by means of n -ary multi-party interactions. The tool takes as input a real-time BIP model and a file containing the safety property. It subsequently generates a Yices [20] output file where the invariants are expressed together with the property. RTD-Finder proceeds by the following steps. It extends the components with history clocks and computes their local invariants. The computation of those invariants requires the implementation of several operations on zones. For this purpose, we developed a DBM (Difference Bound Matrices) library. RTD-Finder subsequently computes the history clocks constraints and the interaction invariant. It writes all these invariants to a file and calls Yices to check the satisfiability of $GI \wedge \neg\Psi$. If $GI \wedge \neg\Psi$ is unsatisfiable, the property is valid. Otherwise, Yices generates a counter-example. We note that, at present, the tool cannot conclude if it is a valid counter-example, however, a guided backward analysis module is currently under development. The benchmarks we used in our experiments with RTD-Finder are described in what follows.

5.1. Train gate controller (TGC). This is a classical example from [3]. The system is composed of a controller, a gate and a number of trains. For simplicity, Figure 5 depicts only one train interacting with the controller and the gate. The controller lowers and raises the gate when a train enters, respectively exits. We propose to check that when all the trains are at *far* location, the gate cannot be going down (g_2 location). The results are presented in Table. 1. When there are more than one train, be it n , the interactions $approach_i \mid approach$ (respectively $exit_i \mid exit$), for $1 \geq i \geq n$ are in conflict on $approach$ (respectively $exit$) of the controller. In this case, in addition to the separation constraints, we made use of the first heuristic presented in Section 4.1. More precisely, the invariant generated by the heuristic is as follows:

$$\bigwedge_{i \neq j} ((h_{approach_i} \leq h_0 \wedge h_{approach_j} \leq h_0) \rightarrow h_{raise} \leq h_0)$$

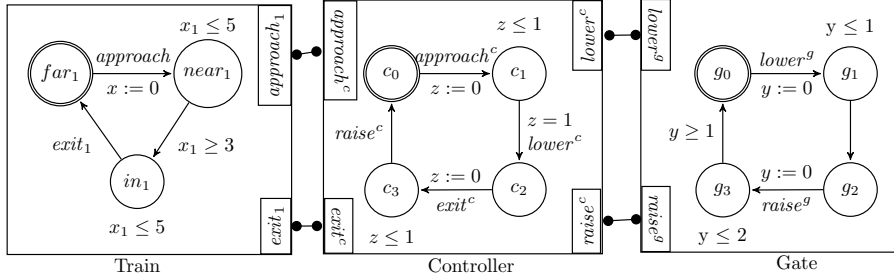


Figure 5: A controller interacting with a train and a gate

5.2. Fischer protocol. This is a well-studied protocol for mutual exclusion [29]. 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. The system with two concurrent processes is represented in Figure 6. 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 made use of the second heuristic presented in Section 4.2. To simplify, we write s_i instead of set_i and e_i instead of eq_i . We construct the regular expression corresponding to location l_i and project it for actions e_i, e_j, s_i, s_j , respectively e_i, e_0, s_i, s_0 . The latter projection leads to the following regular expression in restricted form:

$$\mathbf{R}_i = (e_0 + s_0)^* e_i . s_i + (e_0 + s_0)^* s_i . e_i + (e_0 + e_i)^* s_0 s_i + (e_i + s_0)^* e_0 s_i + s_i$$

This regular expression translates into the following constraint on history clocks:

$$\begin{aligned} \phi(\mathbf{R}_i) = & (h_{e_0} \geq h_{e_i} \wedge h_{s_0} \geq h_{e_i} \wedge h_{e_i} \geq h_{s_i} \wedge h_{e_i} \leq h_0) \vee \\ & (h_{e_0} \geq h_{s_i} \wedge h_{s_0} \geq h_{s_i} \wedge h_{e_i} \leq h_{s_i} \wedge h_{s_i} \leq h_0) \vee \\ & (h_{e_0} \geq h_{s_0} \wedge h_{e_i} \geq h_{s_0} \wedge h_{s_0} \geq h_{s_i} \wedge h_{s_0} \leq h_0) \vee \\ & (h_{s_0} \geq h_{e_0} \wedge h_{e_i} \geq h_{e_0} \wedge h_{e_0} \geq h_{s_i} \wedge h_{e_0} \leq h_0) \vee \\ & (h_{s_i} \leq h_0 \wedge h_{s_0}, h_{e_0}, h_{e_i} > h_0) \end{aligned}$$

We deduce that $at(l_i) \rightarrow \phi(R_i)$ is an invariant of the **Id Variable**, for any i . These invariants in addition to component invariants of processes and inequality constraints $\mathcal{E}(\gamma)$ are sufficient to show that mutual exclusion holds.

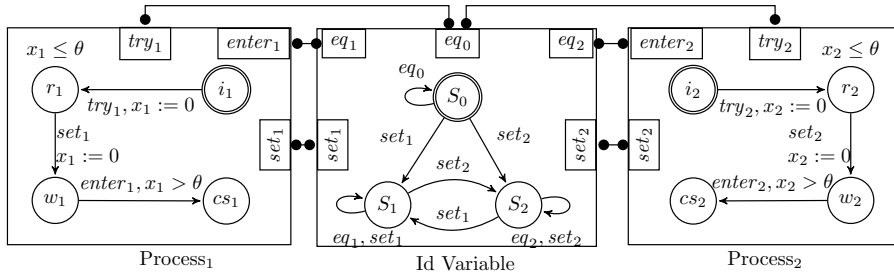


Figure 6: The Fischer protocol

5.3. Gear controller system. Our third example is taken from [31]. There it is described a model of gear controller components in embedded systems operating inside vehicles. A gear controller system is composed of five components: an interface, a controller, a clutch, an engine and a gear-box. The interface sends signals to the controller to change the gear. In turn, the controller interacts with the engine, the clutch and the gear-box. The engine is either regulating the torque or synchronising the speed. The gear-box sets the gear between some fixed bounds. The clutch works as the gear-box and it is used whenever the engine is not able to function correctly (under difficult driving conditions, for instance). One requirement that such a system should satisfy in order to be correct is *predictability*. This requirement ensures a strict order between components. For instance, it ensures that when the engine is regulating the torque, the clutch is closed and the gear-box sets the gear. Another property of interest that we checked is that the controller is in an error location only when one of the other four components is in an error location also.

5.4. Temperature controller (TC). This example is an adaptation from [12]. 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 7) 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. When the controller should take the *cool* action, at least one of the rods is ready to synchronise with it. 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. Recalling the discussion from Section 4.3, such a reduction is correct because the system enjoys the particularly helpful property of being symmetric.

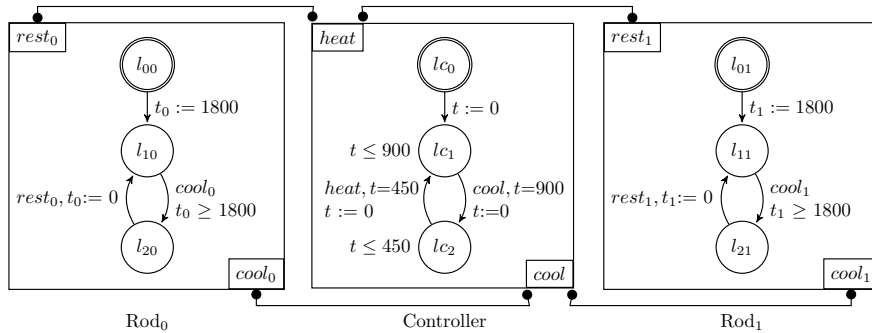


Figure 7: A Controller interacting with two rods

5.5. Dual chamber implantable pacemaker. As a last benchmark, we consider the verification of a dual chamber implantable pacemaker presented in [28]. A pacemaker is a device for the management of the cardiac rhythm. It paces both the atrium and the ventricle of the heart, and based on sensing both chambers it can activate or inhibit further pacing. The model of pacemakers we experimented with has five components, for (1) keeping the heart rate above a minimum value, (2) maintaining delays between atrial and ventricular

activation, (3) preventing pacing the ventricle too fast, filtering noise after (4) ventricular and (5) atrial events. In our experiments, we considered the *upper rate limit (URI)* property stating that the ventricles of the heart should not be paced beyond a maximum rate, equal to a constant called *TURI*. The property states the existence of a minimum time elapse between a ventricular sense (*VS*) event and the following ventricular pace (*VP*) event. As in [28], we verified the property by translating it into a monitor component which is shown in Figure 8. The actions *VS* and *VP* of the monitor are synchronised with those of the other components. We verified that when the monitor reaches the location *interval*, its clock *t* is greater than *TURI*. The corresponding property is $interval \rightarrow t \geq TURI$.

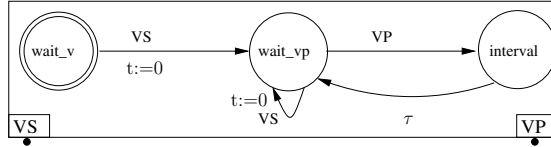


Figure 8: Monitor for the upper rate limit property: the interval between a *VS ventricular event* and a *VP ventricular event* should be longer than *TURI*

Our method offers an additional way to check this property without resorting to the monitor. We expressed it by means of the introduced history clocks. The difference between the history clocks relative to those two events is longer than the required time elapse:

$$(h_{VP} \leq h_{VS} \wedge h_{VS} \leq h_0) \rightarrow h_{VS} - h_{VP} \geq TURI$$

5.6. Results. We ran our experiments on a Linux machine with Intel Core 3.20 GHz \times 4 and 15.6 GiB memory. The results, synthesised in Table 1, show the potential of our method in terms of accuracy and scalability. In Table 1, n is the number of components, q is the total number of control locations, c (resp. h) is the number of system clocks (resp. history clocks), i is the number of interactions, while t shows the total verification time and t_{yices} is the timed taken by Yices for satisfiability checking of $GI \wedge \neg\Psi$.

To the best of our knowledge, there are no tools to compositionally verify safety properties of timed systems. Consequently, there are no relevant tools to compare RTD-Finder with. Nevertheless, we did a small comparison with Uppaal [8]. Uppaal is a well-known model-checking tool which is highly optimised. For instance, thanks to some reduction techniques, it has better scores on the first example (the TGC system) in particular and on smaller systems in general. Nonetheless, generally, state space exploration is costly. This can be illustrated by means of the temperature controller example: for 10 rods, Uppaal generated no results after five hours and 436519 explored states. On the other hand, RTD-Finder checked the property for 300 rods in few minutes, as shown in Table 1. The timings for the RTD-Finder tool are obtained by the java command `getCpuTime` called to compute the total verification time, while the results for Uppaal come from the command `verifyta` which comes with the Uppaal 4.1.14 distribution.

Model	n	q	c	i	h	t	t_{yices}
Train gate controller (50 trains)	52	158	52	102	106	0.5s	0.3s
Train gate controller (100 trains)	102	308	102	202	206	5.3s	0.6s
Train gate controller (200 trains)	202	608	202	402	406	1m33s	5s
Train gate controller (300 trains)	302	908	302	602	606	9m8s	20s
Train gate controller (500 trains)	502	1508	502	1002	1006	1h13m20s	2m52s
Temperature controller (20 rods)	21	42	21	40	42	0.07s	0.01s
Temperature controller (50 rods)	51	102	51	100	102	0.35s	0.04s
Temperature controller (100 rods)	101	204	102	200	204	3.7s	0.08s
Temperature controller (300 rods)	301	602	302	600	602	5m47s	0.9s
Fischer protocol (100 processes)	101	400	101	300	501	2.7s	0.06s
Fischer protocol (200 processes)	201	800	201	600	1001	0m47s	0.22s
Fischer protocol (300 processes)	301	1200	301	900	1501	4m27s	0.5s
Gear controller	5	65	4	17	32	15.1s	0.14s
Pacemaker (with monitor)	7	19	11	6	21	15.23s	0.044s
Pacemaker (without monitor)	6	16	9	6	19	15s	0.032s

Table 1: Results from experiments

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 [6, 24] 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 [18, 30, 4]. 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 [30] 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 [13] 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 [19] 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 [9, 26]. 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 totally avoid global state space exploration. Finally, another successful application of extra clocks

has been provided in [33] 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.

6. CONCLUSIONS

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 successfully tested on several benchmarks. This method has been implemented in the RTD-Finder tool. The results show that it may outperform the existing exhaustive exploration-based techniques for large systems, thanks to the use of compositionality and over-approximations. Nonetheless, the generated invariant is an over-approximation of the reachable states set and false-positives may raise. To remedy this, we are working on a guided backward analysis module to decide upon their validity.

In order to achieve a better integration, we are working on handling richer classes of systems, including systems with data variables and *urgencies* [7] on transitions. Actually, urgencies provide an alternative way to constrain time progress, which is more intuitive to use by programmers but very difficult to handle in a compositional way. A second direction of research which is potentially interesting for systems containing identical, replicated components and closely related to the symmetry-based reduction is the application of our method to the verification of parameterised timed systems. 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.

Acknowledgement. We are grateful to the anonymous referees for their constructive input and for their thorough feedback. We would also like to thank our colleague Mahieddine Dellabani for his help with two benchmarks.

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] R. Alur, T. A. Henzinger, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S. Tasiran. MOCHA: modularity in model checking. In *CAV*, 1998.
- [5] L. Astefanoaei, S. B. Rayana, S. Bensalem, M. Bozga, and J. Combaz. Compositional invariant generation for timed systems. In *TACAS*, 2014.
- [6] B. Badban, S. Leue, and J.-G. Smaus. Automated invariant generation for the verification of real-time systems. In *WING@ETAPS/IJCAR*, 2010.
- [7] A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in BIP. In *SEFM*, 2006.
- [8] G. Behrmann, A. David, K. G. Larsen, J. Håkansson, P. Pettersson, W. Yi, and M. Hendriks. UPPAAL 4.0. In *QEST*, 2006.
- [9] J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi. Partial order reductions for timed systems. In *CONCUR*, 1998.
- [10] J. Bengtsson and W. Yi. On clock difference constraints and termination in reachability analysis of timed automata. In *ICFEM*, 2003.

- [11] S. Bensalem, M. Bozga, T. Nguyen, and J. Sifakis. Compositional verification for component-based systems and application. *IET Software*, 4, 2010.
- [12] S. Bensalem, M. Bozga, J. Sifakis, and T.-H. Nguyen. Compositional verification for component-based systems and application. In *ATVA*, 2008.
- [13] J. Berendsen and F. W. Vaandrager. Compositional abstraction in real-time model checking. In *FORMATS*, 2008.
- [14] S. Bornot and J. Sifakis. An algebraic framework for urgency. *Information and Computation*, 1998.
- [15] P. Bouyer. Forward analysis of updatable timed automata. *Form. Methods Syst. Des.*, 2004.
- [16] 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.
- [17] C. Courcoubetis and M. Yannakakis. Minimum and maximum delay problems in real-time systems. *Formal Methods in System Design*, 1992.
- [18] 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.
- [19] F. S. de Boer, U. Hannemann, and W. P. de Roever. Hoare-style compositional proof systems for reactive shared variable concurrency. In *FSTTCS*, 1997.
- [20] B. Dutertre and L. de Moura. The Yices SMT solver. Technical report, SRI International, 2006.
- [21] E. A. Emerson and V. Kahlon. Reducing model checking of the many to the few. In *CADE*, 2000.
- [22] E. A. Emerson and K. S. Namjoshi. Reasoning about rings. In *POPL*, 1995.
- [23] E. A. Emerson and A. P. Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1/2), 1996.
- [24] A. Fietzke and C. Weidenbach. Superposition as a decision procedure for timed automata. *Mathematics in Computer Science*, 2012.
- [25] G. Gardey, D. Lime, M. Magnin, and O. H. Roux. ROMEO: A tool for analyzing time Petri nets. In *CAV*, 2005.
- [26] J. Håkansson and P. Pettersson. Partial order reduction for verification of real-time components. In *FORMATS*, 2007.
- [27] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Inf. Comput.*, 1994.
- [28] Z. Jiang, M. Pajic, S. Moarref, R. Alur, and R. Mangharam. Modeling and verification of a dual chamber implantable pacemaker. In *TACAS*, 2012.
- [29] L. Lamport. A fast mutual exclusion algorithm. *ACM Trans. Comput. Syst.*, 1987.
- [30] S.-W. Lin, Y. Liu, P.-A. Hsiung, J. Sun, and J. S. Dong. Automatic generation of provably correct embedded systems. In *ICFEM*, 2012.
- [31] M. Lindahl, P. Pettersson, and W. Yi. Formal design and analysis of a gear controller. In *TACAS*, 1998.
- [32] K. S. Namjoshi. Symmetry and completeness in the analysis of parameterized systems. In *VMCAI*, 2007.
- [33] R. B. Salah, M. Bozga, and O. Maler. Compositional timing analysis. In *EMSOFT*, 2009.
- [34] S. Tripakis. Verifying progress in timed systems. In *ARTS*, 1999.
- [35] F. Wang. Redlib for the formal verification of embedded systems. In *ISoLA*, 2006.