A Compositional Approach to the Verification of Hybrid Systems

Lacramioara Astefanoaiei, Saddek Bensalem, Marius Bozga

A Compositional Approach to the Verification of Hybrid Systems

Lăcrămioara Aştefănoaei[1], Saddek Bensalem[2], Marius Bozga[2]

[1] fortiss - An-Institut TUM, Guerickestr. 25, 80805 München, Germany
[2] UJF-Grenoble, CNRS VERIMAG UMR 5104, Grenoble F-38041, France

Abstract. The increase of complexity in modelling systems and the chances of success when model-checking them tend to be inversely proportional. This mere observation justifies plainly the need to investigate alternative ways for verification. In this paper we present such an alternative which uses a compositional verification rule. The basic idea is to automatically compute local properties and combine them such that together they are strong enough to prove global safety properties of systems. In [2] we showed how such a rule works in the framework of timed systems with a fixed number of components and in [3] how the whole approach can be extended to the parameterised case. The application of the compositional verification rule can be pushed even further with respect to two directions: (1) hybrid and (2) parametric systems. This is the subject of the present paper.

1 Introduction

This paper spiraled from three concepts: *compositionality*, *safety* and *hybrid systems*. On compositionality, we would like to recall a short text from Dijkstra's "On Understanding Programs":

> *On a number of occasions I have stated the requirement that if we ever want to be able to compose really large programs reliably, we need a discipline such that the intellectual effort E (measured in some loose sense) needed to understand a program does not grow more rapidly than proportional to the program length L (measured in an equally loose sense) and that if the best we can attain is a growth of E proportional to, say L^2, we had better admit defeat. As an aside I used to express my fear that many programs were written in such a fashion that the functional dependence was more like an exponential growth.*

Despite being frequently used over the years, we feel that the fragment in particular and Dijkstra's remarks in general have not lost their savoury and charm.

Compositionality and safety, together with time, have already been the main characters in our previous work [2, 3] where we orchestrated a compositional method for verifying timed automata. *Timed automata* is an expressive formalism for modelling timing constraints. It would be not an easy task to ignore "time" as a less important concept. Deadlines, delays... are everywhere. Correct scheduling, to name but one time related application, is crucial for critical systems. In this paper we propose to go a bit further, to the "realm of hybridity" and show how our method scales to the verification of state safety properties in the context of *(parametric) hybrid systems* interacting by means of multi-party interactions. This is just but a first step in a more ambitious project on verifying programable controller programs (PLCs) in the context of cyber-physical systems in the domain of industrial automation. "Cyber-physical systems" is all about interconnecting devices, sensors, actuators, all of these being distributed. Such dynamic systems with both discrete and continuous components fit well the class of hybrid systems: PLCs play the role of discrete components while the external environment sensed or impacted by devices such as valves, sensors, or activators exhibits a continuous behaviour. Lately, results on the application of hybrid systems are documented in projects such as COMPASS[3], Veriware[4], or in CPS-VO groups such as ARCH[5] and UncoVerCPS[6] to name but a few. These recent successes suggest that the use of the formalism in industry is growing and this brings opportunities for verification to be put into practice especially in domains where safety is a most critical aspect. We note that, in academia, the verification of hybrid systems has been studied since the early nineties. Central to verification, the reachability problem has been shown to be undecidable for hybrid automata except for few cases such as variations on timed automata and we refer to [23] as a classical reference. Nevertheless, this is not a reason to discourage as by means of abstraction, the fixpoint computation behind reachability converges and in fact there is quite a variety of approaches and tools for model-checking [16, 19, 17, 10, 30, 42, 8, 15, 21, 37, 41, 31, 9, 12, 32], to cite the most recent results. Compositional approaches are fewer. The relevant references we are aware of are [24, 18, 39, 26, 6, 14, 29] and they are either based on assume guarantee or rely on user interaction as it is the case with the interactive prover KeYmaera [39, 29]. Consequently, we find that it is worth-while investigating *automatic* compositional approaches and this offers us enough justification to motivate our work. Our methodology follows the one introduced in [2]. The building blocks can be summed up as the following steps:

- generate local invariants for individual components as over approximations of their symbolic state space (possibly in a property-driven manner *à la* IC3 [40])
- generate interaction invariants to express relations between the different components and/or auxiliary variables
- assemble all invariants into one formula (possibly quantify it existentially on unknown parameters), and feed it to the SMT-solver Z3 [35].

We emphasise that all the computations are completely automatic. We note that these basic building blocks are not new. The novelty is more in bringing them

[3] compass.informatik.rwth-aachen.de
[4] veriware.org
[5] cps-vo.org/group/ARCH
[6] cps-vo.org/group/UnCoVerCPS

together in a coherent methodology. Its simplicity should encourage its use as especially a preliminary step in verification. Behind "preliminary" is the fact that our method is sound, but not complete. Consequently, if Z3, when given as input the formula corresponding to our verification rule, yields "no solution", or in other words, that the "bad" states are not reachable, then we are done. Otherwise, auxiliary techniques, as for instance, counterexample refinement, are needed to prove the system safe.

Organisation of the paper. Section 2 recalls the classical definitions used in our framework. Section 3 presents how to effectively verify linear hybrid systems compositionally. Section 4 discusses extensions and Section 5 concludes.

2 Model

In our setup, components are linear hybrid automata and systems are compositions of components with respect to multi-party interactions. The definitions for hybrid automata are adopted from [1, 21]. Before recalling them, we first fix some notation.

We use $\mathcal{X}$ to denote real-valued variables. A valuation $\mathbf{v}$ is a function that assigns a real-value $\mathbf{v}(x) \in \mathbb{R}$ to each variable $x \in \mathcal{X}$. It is useful to note that a valuation $\mathbf{v}$ can be identified with the point $(\mathbf{v}(x_1), \dots, \mathbf{v}(x_n)) \in \mathbb{R}^n$. We denote by $\mathbf{V}$ the set of valuations. Given a set of variables $\mathcal{X}$, a linear inequality has the form $\sum_{i=1}^{n} \alpha_i x_i \# \beta_i$ with $x_i \in \mathcal{X}$, $\alpha_i, \beta \in \mathbb{Z}$, $\# \in \{<, \leq, =, \geq, >\}$. A convex linear constraint is a finite conjunction of linear inequalities. The set of convex linear constraints over $\mathcal{X}$ is denoted by $\mathcal{L}(\mathcal{X})$. The geometrical interpretation of a convex linear constraint is that of as a convex polyhedron.

Definition 1. *A component is a hybrid automaton $(L, l_0, \mathcal{X}, A, T, \mathsf{tpc}, \mathcal{D})$ where:*

- *L is a finite set of locations and l_0 is an initial location;*
- *$\mathcal{X}$ is a finite set of real-valued variables;*
- *A a finite set of actions;*
- *T is a set of transitions: each transition $\tau = (l, a, g, \mu, l')$ consists of a source location $l \in L$, a target location $l' \in L$, an action $a \in A$, g is a guard condition in $\mathcal{L}(\mathcal{X})$, and a jump relation $\mu \in \mathcal{L}(\mathcal{X} \cup \mathcal{X}')$ with $\mathcal{X}'$ denoting the variables at l';*
- *$\mathsf{tpc} : L \to \mathcal{L}(\mathcal{X})$ assigns a convex linear time progress condition to each location;*
- *$\mathcal{D} : L \to (\mathbb{R}^n \to \mathbb{R}^n)$ assigns activities to each location. The activities D_l describe how the continuous variables evolve within each location l.*

The class of hybrid automata with linear dynamics is called *linear* hybrid automata (LHA). By linear dynamics it is meant that the activities are given by convex linear constraints over the time derivatives of the variables, that is, $\mathcal{D}_l$ is in $\mathcal{L}(\dot{\mathcal{X}})$ for each l in L.

We restrict to linear dynamics for two reasons. The first one is to simplify the presentation: technically, it makes little difference had the dynamics been more complicated. Our second reason is of a more pragmatic type: after investigating the existing tools, the ones answering best our needs handle only LHAs. As a side remark, we note also that though there has been a considerable amount of work and advancement on SMT solvers for nonlinear arithmetics (Z3, SMT-RAT, CVC3, miniSMT, RAHD, hydlogic, dReal, iSAT to name a few cited in [10]), their current performance is still not satisfactory and their scalability is problematic [36].

The semantics of a component B modelled as an LHA $(L, l_0, \mathcal{X}, A, T, \mathsf{tpc}, \mathcal{D})$ is given as a labelled transition system $(Q, A, \rightarrow)$ where $Q \subseteq \{(l, \mathbf{v}) \in L \times \mathbf{V} \mid \mathbf{v} \in \mathsf{tpc}(l)\}$ denotes the states of B and $\rightarrow \subseteq Q \times (A \cup \mathbb{R}_{\geq 0}) \times Q$ denotes the transitions according to the rules:

- $(l, \mathbf{v}) \xrightarrow{\delta} (l, \mathbf{v}')$ if $\exists k \in \mathcal{D}(l). \mathbf{v}' = \mathbf{v} + \delta k$ (time progress);
- $(l, \mathbf{v}) \xrightarrow{a} (l', \mathbf{v}')$ if $\big(l, (a, g, \mu), l'\big) \in T$, $\mathbf{v} \in g$ and $(\mathbf{v}, \mathbf{v}') \in \mu$ (action step).

Since this semantics yields an infinite state space, to effectively compute the states of a component, symbolic representations are used instead. A symbolic state is a pair (l, ζ) of a location l and a constraint ζ over variables. It has been shown that the reachable states of an LHA can be effectively represented by convex polyhedra [1]. Consequently, the operations corresponding to the delay and action transitions are performed on convex polyhedra rather than on concrete valuations. As discussed in [18], what is crucial is implementing them efficiently. Here, we only recall their definitions from [27] however adapted slightly as in [21]. The operation $\mathsf{time\_succ}$ for letting time progress within a symbolic state is defined as $\mathsf{time\_succ}((l, \zeta)) = (l, \zeta \uparrow_q)$ where $\uparrow_q$ is the time-elapse operator defined in turn as follows:

$$\mathbf{v}' \in \zeta \uparrow_q \text{ iff } \exists \mathbf{v} \in \zeta, \delta \in \mathbb{R}_{\geq 0}, k \in \mathcal{D}(l). \mathbf{v}' = \mathbf{v} + \delta k \wedge \mathbf{v}' \in \mathsf{tpc}(l).$$

The successor with respect to a discrete transition $t = \big(l, (\_, g, \mu), l'\big)$ is defined as $\mathsf{disc\_succ}(t, (l, \zeta)) = (l', \zeta')$ where

$$\mathbf{v}' \in \zeta' \text{ iff } \exists \mathbf{v} \in \zeta \cap \mathsf{tpc}(l) \cap g. (\mathbf{v}, \mathbf{v}') \in \mu \wedge \mathbf{v}' \in \mathsf{tpc}(l').$$

With these two operations, the successor operator, succ, is defined simply as $\mathsf{succ}(t, (l, \zeta)) = \mathsf{time\_succ}(\mathsf{disc\_succ}(t, (l, \zeta)))$.

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

Given a component B with initial symbolic state s_0 and transitions T, the set of reachable symbolic states $\mathit{Reach}(B)$ is $\mathit{Reach}(s_0)$ where Reach is defined recursively for an arbitrary s as $\{s\} \cup \bigcup\limits_{t \in T} \mathit{Reach}(\mathsf{succ}(t, s))$.

In our framework, components communicate by means of *interactions*, which are synchronisations between their actions. Given n components B_i, $1 \leq i \leq n$, with disjoint sets of actions A_i, an interaction is a subset of actions $\alpha \subseteq \cup_i A_i$ containing at most one action per component, that is, of the form $\alpha = \{a_i\}_{i \in I}$, with $a_i \in A_i$ for all $i \in I \subseteq \{1, \ldots, n\}$. Given a set of interactions $\gamma \subseteq 2^{\cup_i A_i}$, we denote by $Act(\gamma)$ the set of actions involved in γ, that is, $Act(\gamma) = \cup_{\alpha \in \gamma} \alpha$. A *hybrid system* is the composition of components B_i for a set of interactions γ such that $Act(\gamma) = \cup_i A_i$. For n components $B_i = (L_i, l_0^i, \mathcal{X}_i, A_i, T_i, \mathsf{tpc}_i, \mathcal{D}_i)$ with $A_i \cap A_j = \emptyset$, $\mathcal{X}_i \cap \mathcal{X}_j = \emptyset$, for any $i \neq j$, the *composition* $\|_\gamma B_i$ with respect to a set of interactions γ is defined by an LHA $(L, \bar{l}_0, \mathcal{X}, \gamma, T_\gamma, \mathsf{tpc}, \mathcal{D})$ where $\bar{l}_0 = (l_0^1, \ldots, l_0^n)$, $\mathcal{X} = \cup_i \mathcal{X}_i$, $L = \times_i L_i$, $\mathsf{tpc}(\bar{l}) = \cap_i \mathsf{tpc}_i(l_i)$, $\mathcal{D}(l) = \cap_i \mathcal{D}_i(l_i)$ and T_γ is such that for any interaction $\alpha = \{a_i\}_{i \in I}$ we have that $\bar{l} \xrightarrow{\alpha, g, \mu} \bar{l}'$ where $\bar{l} = (l_1, \ldots, l_n)$, $g = \cap_{i \in I} g_i$, $\mu = \cap_{i \in I} \mu_i$, and $\bar{l}'(i) = \text{if } (i \notin I)\ l_i \text{ else } l_i'$ for $l_i \xrightarrow{a_i, g_i, r_i} l_i'$. In a system $\|_\gamma B_i$ a component B_i can execute action a_i only as part of an interaction α containing it, that is, along with the execution of all other actions from α. This corresponds to the usual notion of multi-party interaction.

Remark 1. Our method being compositional, allowing shared variables is error-prone. Consequently, we require that the sets of local variables are disjoint. However, we note that, in principle, we could deal with a certain "amount" of sharing by adopting a strategy as follows: components share variables in a read-only fashion, while updates can take place in the "owner" component.

Example 1. As a working example we take a classic one, that of a temperature control system which was described first in [28]. We, however, use the model from [1]. The system maintains the coolant temperature inside a reactor tank within given bounds 3 and 15 by moving two rods. When the temperature reaches 15, the controller uses a rod to refrigerate the tank. The temperature rises and decreases at the rate of 6, respectively 2. A rod can be reused only after 6 time units. If the temperature cannot decrease because no rod is available the system is shutdown. Figure 1 shows the corresponding hybrid system with t measuring the temperature and x_0, x_1 the clocks counting the time elapsed since the last use of rod 0 and 1. The set of interactions γ is $\{heat \mid rest_i, cool \mid cool_i\}$ with $i \in \{0, 1\}$. For clarity, the activities and the time progress conditions associated with the locations of the controller are depicted in blue. We denote the trigger for shutdown as *shutdown* $:= (t = 15 \wedge_i x_i < 6)$. We say that the system is *safe* when it is not shutdown and, for the ease of reference, we denote this property by Ψ, i.e., $\Psi := \neg$*shutdown*.

Remark 2. We note that in [1] the system is presented as one component (representing the composition of the controller and the two rods) while we manually decomposed it into three components. At what extent can such decompositions be automated is a research topic on its own.

The separate executions of the controller and of the rods respectively are as follows:

$$\begin{tikzcd}
(lc_0, 15 \geq t \geq 0) \arrow[r, "cool"] & (lc_1, 15 \geq t \geq 3) \arrow[r, "heat"] & (lc_0, 15 \geq t \geq 3) \arrow[l, bend left, "cool"]
\end{tikzcd}
\qquad
\begin{tikzcd}
(l_{0i}, x_i \geq 6) \arrow[r, "cool_i"] & (l_{1i}, x_i \geq 6) \arrow[r, "rest_i"] & (l_{0i}, x_i \geq 0) \arrow[l, bend left, "cool_i"]
\end{tikzcd}$$

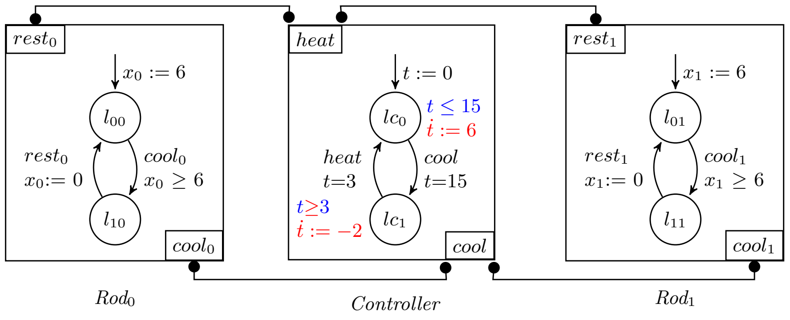

Fig. 1. Temperature Controller System

The executions of the system are longer. For illustration, we show the first three steps as obtained with the tool Hymitator [21]:

$$
\begin{aligned}
&(lc_0, l_{00}, l_{01}, 15 \geq t \geq 0 \wedge t + 36 = 6x_0 \wedge t + 36 = 6x_1) \xrightarrow{cool|cool_0} \\
&(lc_1, l_{10}, l_{01}, 15 \geq t \geq 3 \wedge t + 2x_0 = 32 \wedge t + 2x_1 = 32) \xrightarrow{heat|rest_0} \\
&(lc_0, l_{00}, l_{01}, 15 \geq t \geq 3 \wedge t = 3 + 6x_0 \wedge t + 84 = 6x_1)
\end{aligned}
$$

We note that within these steps the system is safe, and, in fact, it is never the case that the temperature reaches the value 15 with both x_i being less than 6.

3 Compositional Verification

At the heart of our method is the verification rule (VR) from [5]. Its beauty is in its simplicity and genericity. Assume that a system consists of n components B_i interacting by means of an interaction set γ, and that the property that the system should satisfy is Ψ. If components B_i and interactions γ can be locally characterised by means of invariants (here denoted $CI(B_i)$, resp. $II(\gamma)$), and if Ψ can be proved to be a logical consequence of the conjunction of the local invariants, then Ψ is a global invariant. In Figure 2, the symbol $\vdash$ is used to underline that the logical implication can be effectively proved (for instance with an SMT solver) and the notation "$B \models \square\, \Psi$" is to be read as "Ψ holds in every reachable state of B".

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

Fig. 2. Compositional Verification

Thanks to its genericity, (VR) can be instantiated with respect to different modelling frameworks. In [2] we took timed systems as a case of study. Next, we make the move towards (parametric) hybrid systems.

3.1 Component Invariants

Component invariants characterise the reachable states of components when considered alone. More precisely, given that the set of the reachable symbolic states

(l_j, ζ_j) of an arbitrary component B is finite, its invariant is defined by the disjunction $\vee_j (l_j \wedge \zeta_j)$, where by abuse of notation l_j is used to denote the predicate that holds whenever B is at location l_j.

Example 2. As an illustration, after simplifications, the component invariants for the controller and for the rods from our running example are as follows:

$$\begin{aligned} CI(\mathit{Controller}) &= (lc_0 \wedge 15 \geq t \geq 0) \vee (lc_1 \wedge 15 \geq t \geq 3) \\ CI(\mathit{Rod}_i) &= (l_{0i} \wedge x_i \geq 0) \vee (l_{1i} \wedge x_i \geq 6) \end{aligned} \tag{1}$$

We note the correspondence between these formulae and the local executions as shown in Section 2.

3.2 Interaction Invariants

Interaction invariants are over-approximations of the global state space allowing us to disregard certain tuples of local states as unreachable. Interaction invariants $II(\gamma)$ are induced by the synchronisations and have the form of global conditions involving control locations of components. Previous work considered boolean conditions [5] as well as linear constraints [33] as methods for generating $II(\gamma)$.

Example 3. For simplicity, we show the interaction invariant corresponding to the set of interactions between the controller and rod$_0$:

$$II(\{\mathit{heat} \mid \mathit{rest}_0, \mathit{cool} \mid \mathit{cool}_0\}) = (l_{00} \vee lc_1) \wedge (l_{10} \vee lc_0). \tag{2}$$

The invariant is given in conjunctive normal form to stick to the formalism in [5]. The disjunctions represent the so called "initially marked traps" in a Petri net which corresponds to the synchronisation skeleton of our model. Intuitively, a trap can be seen as a set of places which always contains tokens if they have tokens initially. To better "see" Formula (2), the reader can transform it in disjunctive normal form, and after eliminating conjunctions such as $l_{00} \wedge l_{10}$ (a component cannot be at two locations simultaneously), what remains is the disjunction $(l_{00} \wedge lc_0) \vee (l_{10} \wedge lc_1)$. In this particular case, the invariant is an exact characterisation of the global state space of the untimed (sub)system $\mathit{Controller} \| \mathit{Rod}_0$.

3.3 History Clocks & Auxiliary Constraints

As argued in [2], a direct application of the rule (VR) may be too weak in the sense that the component and the interaction invariants derived from the traps are usually not enough to prove global properties, especially when such properties involve relations between clocks in different components. For instance, in the temperature controller scenario, we cannot show that shutdown does not hold by only having at hand the invariants for components and interactions: any valuation such that $t = 15$ and $x_i < 6$ satisfies $CI(\mathit{Controller}) \wedge_i CI(\mathit{Rod}_i) \wedge \wedge II(\gamma) \wedge \mathit{shutdown}$. History clocks allow to decouple the analysis for components and for their composition. They make it possible to derive new global constraints from the simultaneity of interactions and the synchrony of time progress.

Adding History Clocks. History clocks are associated with actions and interactions. For a component B we use B^h to denote its extension with history clocks. The extension of the system is obtained from the extensions of the components alone together with the history clocks for interactions. As an illustration, Figure 3 shows the extension of the system in Figure 1.

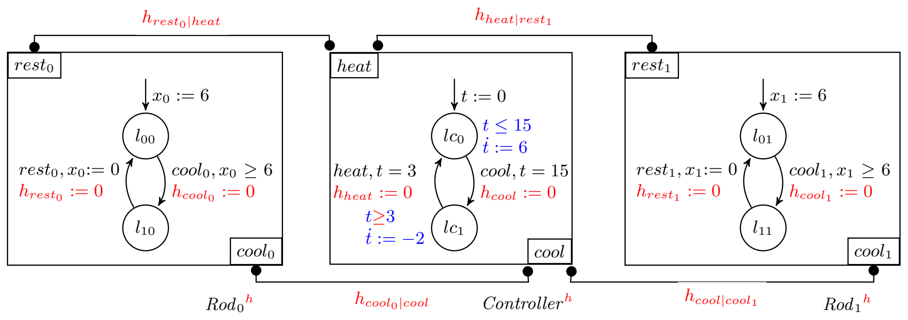

Fig. 3. Illustrating Components with History Clocks for (Inter)Actions

The mechanism of history clocks can be understood as follows. When an interaction α takes place, the history clocks h_α and h_a associated to α and to any action $a \in \alpha$ are reset. Thus they measure the time passed from the last occurrence of α, respectively of a. We note that, since there is no timing constraint involving history clocks, the behaviour of the components is not changed by the addition of history clocks.

Generating Interaction Equalities from History Clocks. The starting point is the following basic fact: a history clock h_a for an action a from a last executed interaction α is necessarily *less* than any h_β with β another interaction containing a. This is because the clocks of the actions in α are the last ones being reset. Consequently, given a common action a of $\alpha_1, \alpha_2, \dots, \alpha_p$, h_a is the minimum of h_{α_i}, $h_a = \min\limits_{i \in [p]} h_{\alpha_i}$. The resulting invariant for a given interaction set γ is denoted as $\mathcal{E}(\gamma)$ and defined as follows:

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

Example 4. For our running example, $\mathcal{E}(\gamma)$ is given by the conjunction:

$$h_{heat} = \min_{i \in \{0,1\}} h_{rest_i} \wedge h_{cool} = \min_{i \in \{0,1\}} h_{cool_i} . \tag{3}$$

Generating Inequalities from Conflicting Interactions. The (in)equality constraints shown previously allow to relate local constraints obtained separately from the component invariants. Without conflicts, that is, when interactions do not share any action, the generated invariants are quite tight in the sense that $\mathcal{E}(\gamma)$ is essentially a conjunction of equalities. However, $\mathcal{E}(\gamma)$ is weaker in the presence of conflicts because any action in conflict can be used in different interactions. The disjunctions (implicit in the definition of min) in $\mathcal{E}(\gamma)$ reflect precisely this uncertainty. History clocks on interactions are introduced to capture the time lapses between conflicting interactions. The basic information we can exploit is that when two conflicting interactions compete for the same action a, no matter which one is first, the other one must wait until the component which owns a is again able to execute a. This is referred to as a "separation constraint" for conflicting interactions and is defined as the following invariant:

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

where $\mid x \mid$ 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.

Remark 3. If in the case of timed automata exact methods to compute k_a exist[7], in the case of hybrid systems, we are not aware of such approaches. However, a simple but incomplete heuristics to determine a correct value for k_a is to guess and do a local model-check.

Example 5. For our running example, we have that $\mathcal{S}(\gamma)$ is given by:

$$\mid h_{heat|rest_0} - h_{heat|rest_1} \mid \geq k_{heat} \wedge \mid h_{cool|cool_0} - h_{cool|cool_1} \mid \geq k_{cool}. \tag{4}$$

By inspecting the model, one can note that the constants k_{cool} and k_{heat} are both equal to the sum of the time lapses at lc_0 and lc_1 which reduces to 8.

3.4 Revisiting (VR)

With the new the clock constraints $\mathcal{E}$ and $\mathcal{S}$, the generalisation of the rule (VR) from Section 2 boils down to checking the validity of the following formula:

$$\underbrace{\wedge_i CI(B_i^h) \wedge II(\gamma) \wedge \mathcal{E}(\gamma) \wedge \mathcal{S}(\gamma)}_{GI} \rightarrow \Psi \tag{5}$$

or equally the unsatisfiability of $GI \wedge \neg\Psi$.

The soundness of (VR) follows from the basic fact that the conjunction of invariants is in turn an invariant and from the observation that each of the constituting elements of GI is an invariant.

[7] We refer to [13] for an approach which reduces the computation to finding a shortest path in a weighted graph built from the zone graph associated of a timed automaton.

Example 6. As an illustration, we work through our running example from head to tail. We recall that we take, as a safety state property, $\Psi := \neg \mathit{shutdown}$ with *shutdown* being $t = 15 \wedge_i x_i < 6$. We first reproduce the component invariants for the rods and the controller extended with history clocks as provided by Hymitator.

$$\begin{aligned}
CI(Rod_i^h) =& (l_{0i} \wedge x_i \geq 6) \vee \\
& (l_{1i} \wedge x_i \geq 6 + h_{cool_i}) \vee \\
& (l_{0i} \wedge h_{cool_i} \geq h_{rest_i} \wedge h_{rest_i} \geq 0 \wedge x_i = h_{rest_i}) \vee \\
& (l_{1i} \wedge x_i = h_{rest_i} \geq h_{cool_i} \wedge h_{rest_i} \geq 6 + h_{cool_i}) \\
CI(Controller^h) =& 15 \geq t \wedge \\
& \big((lc_0 \wedge t \geq 0) \vee \\
& (lc_1 \wedge t \geq 3 \wedge 15 = t + 2h_{cool}) \vee \\
& (lc_0 \wedge t \geq 3 \wedge 12 + 6h_{cool} = 45 + t \wedge 3 + 6h_{heat} = t) \vee \\
& (lc_1 \wedge t \geq 3 \wedge 15 = t + 2h_{cool} \wedge 3t + 6h_{heat} = 57)\big)
\end{aligned}$$

These local invariants together with the interaction invariant, the (in)equality and the separation constraints represented by Formulae (3), (4) are an instantiation of GI. We feed this instantiation together with *shutdown* to Z3 and ask for a solution. The result "no solution" allows us to conclude that the system is safe. To give an intuition why this is indeed the case, we take a closer look at the formulae at hand. The only problematic case is when both rods are at the initial locations (at the other location we already knew from Formula (1) that $x_i \geq 6$). The relevant equations are $x_i = h_{rest_i}$. Let us assume that the controller is at lc_0 (the other location is dismissed by the interaction invariant). To have a shutdown, t is 15. Consequently, h_{cool} is 8 and h_{heat} is 2. Assume that the minimum between h_{rest_i} is h_{rest_0}, that is, $x_0 = 2$. By the separation constraint, h_{rest_1} is at least 10 and implicitly x_1 can be used for cooling, consequently, the system is so far safe.

4 Bringing Parameters into Play

In this section we present some ongoing research about parameters and their roles within our framework. We distinguish two levels where parameters can enter the scene: (1) at component level, and this leads to systems of *parametric* LHAs, and (2) at system level, and this leads to *parameterised* hybrid systems.

4.1 Parametric LHAs

At component level, parameters may represent for instance physical constants dependent of the environment or values that the designer should set such as timing constraints which are to be known only at deployment. Working with parameters in early stages of development has the direct benefit that it makes it possible to explore different design choices and evaluate their robustness.

To handle parameters at the level of components, it suffices to split $\mathcal{X}$ into 2 disjoint sets $\mathcal{V}$ and $\mathcal{P}$, where $\mathcal{V}$ plays the role of $\mathcal{X}$ as in Definition 1 while $\mathcal{P}$ stores parameters. Parameters can be seen as a particular type of variables whose values do not change over time.

Returning to the temperature controller example, we recall that the constants used in the system are values for minimum and maximum bounds, recovering times for rods, temperature increase and decrease rates. If we see all these as parameters, the corresponding system is the one illustrated in Figure 4.

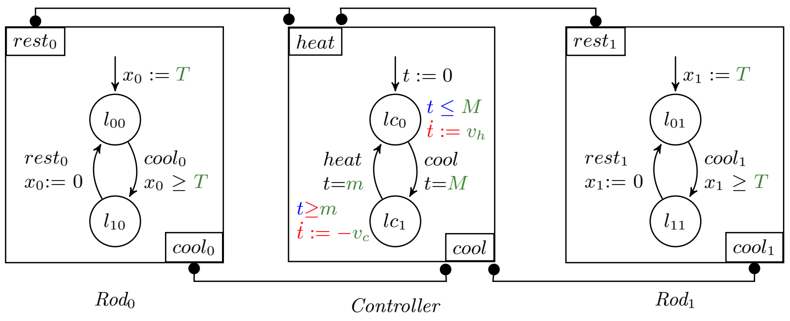

Fig. 4. Parametric Temperature Controller System

To apply (VR) on systems with components as parametric LHAs, all the methods for computing local invariants, as described in Section 3, except one, remain unchanged. The computation of interaction invariants and equality constraints only depends on the set of interactions. Neither does the computation of component invariants, however the output reflects parameters instead of constants. As an illustration, the component invariant for the controller computed with Hymitator is as follows:

$$\begin{aligned} CI(Controller^h) = & M \geq t \wedge M \geq 0 \wedge \\ & \big((lc_0 \wedge t \geq 0) \vee \\ & (lc_1 \wedge t \geq m \wedge M = t + 2h_{cool}) \vee \\ & (lc_0 \wedge t \geq m \wedge 4m + 6h_{cool} = 3M + t \wedge m + 6h_{heat} = t) \vee \\ & (lc_1 \wedge t \geq m \wedge M = t + 2h_{cool} \wedge m + 3t + 6h_{heat} = 4M)\big) \end{aligned}$$

The exception is the computation of separation constraints. The exact parametric value could be manually computed. In the case of the running example, one can show that k_{cool} and k_{heat} are both equal to $(M - m)(1/v_c + 1/v_h)$. Generalising such computations may seem hopeless. We note that not being able to compute separation constraints does not mean that (VR) is no longer applicable, but that it is less stronger. Also, in the case when the component where one needs to compute separation constraints does not have parameters, then (VR) preserves its strength. In our running example, this boils down to T being the only parameter. For the sake of reaching a conclusion, let us continue our thought experiment irrespective of being able to automatically compute separation constraints. We do so by asking the question: "knowing that by default variables are understood as universally quantified, does it make sense to input (VR) as is to Z3?" If we look at our running example, it is apparent that the system would not be safe for any value of T. In fact, by a closer analysis, for the system to be safe it is sufficient that the sum of the time lapse for the controller to raise the temperature and twice the time lapse for the two rods to refrigerate, i.e., $(M-m)(1/v_h+2/v_r)$, is greater than T. From this, we reach a more general remark that verification of parametric (hybrid) systems boils down to a synthesis problem. Basically, what we would like to ask (VR) is to synthesise concrete values such that the constraints we have just derived manually are satisfied.

As a side remark and as a justification of our thought experiment, we note that parameter synthesis for hybrid systems has already been addressed first in [25] and later refined in [20, 21, 11]. However, doing it compositionally is new.

To effectively tackle the new synthesis problem, a first approach is to quantify the parameters in (VR) existentially. However, as argued in [11], this would give us just one set of "good" parameters. In this sense, it would be of interest to follow the approach from [11] which basically computes all the good parameters by finding the bad ones: the good parameters are simply the domains from which the bad parameters are eliminated. Regarding this direction, our experiments with tactics for eliminating quantifiers and simplifying formulae in Z3 did not lead to concrete results. Experimenting along the first direction is more on the positive side. For instance, if we ask Z3 to solve $\exists T. GI[M \leftarrow 15, m \leftarrow 3] \wedge \Psi$ it yields a solution where T is 1. This is, indeed, a valid value but we would be interested in finding the maximal T for which a rod is allowed to recover without leading to a shutdown. We could find such a value in an iteratively manner, by means of a binary search, for instance, to advance at a faster pace. How useful would recent max-SMT techniques [7] be for finding optimal solutions and at what extent this approach can be automated and generalised to handle multiple parameters needs still to be investigated.

4.2 Parameterised Hybrid Systems

Taking parameters at system level is more inline with our previous work in [3]. There, we have made use of a small model result to verify parameterised timed systems, that is, timed systems with arbitrary many replicated components. The verification of parameterised systems is usually refered to as "uniform verification". As an illustration, we use our running example: the controller together with an arbitrary number of rods forms a parameterised hybrid system.

Concretely, in [3], we have shown how (VR) can be extended to tackle uniform verification of a restricted[8] class of $\forall^*\exists^*$ properties for parameterised timed

[8] The restriction consists in only allowing linear constraints on variables and comparisons between indices while disallowing comparisons between variables and indices.

systems. The underlying technicality was to "massage" the formula corresponding to (VR), that is, $GI \to \Psi$ into a restricted $\forall^*\exists^*$ property. Thanks to this, we were able to apply a small model result which allowed us to reduce uniform verification to the verification of a small number of replicas. The bound is mostly related to the number of universal quantifiers behind (VR). The methodology in [3] naturally extends to parameterised hybrid systems. The only difference is in computing local invariants, as shown in Section 3.

Remark 4. So far, the setup from [3] only allows replicas as *identical* copies. Consequently, as is, the framework cannot handle parametric components. This would be a wanted feature as it is not infrequent that guards in components are parametric in the number of components. It is even the case of our running example: the parameterised temperature controller system cannot be safe for an arbitrary number n of rods unless the guard in the replicated rod depends on n. However, technically, such a guard does not fit our $\forall^*\exists^*$ properties. It would be of interest to see if we could borrow ideas from [22] to relax our restrictions.

5 Conclusions

We presented a compositional approach to the verification of hybrid systems. On the positive side, thanks to compositionality itself and to the speed of SMT solvers such as Z3, the approach scales quite well. On the negative side, while working with abstractions we run into false positives. In principle, false positives could be eliminated by means of a CEGAR approach but further experiments are needed to evaluate the real strength of the method.

Besides clarifying the points raised in Section 4, we would be interested in a few more experiments. One is about using tools such as Hycomp [12] or flow* [8]. Hycomp has the advantage that it uses IC3 [40] for a property driven computation of the set of reachable states. In principle, this would allow us to have smaller component invariants. flow* handles non-linear hybrid automata, thus it would make it possible to effectively experiment with (VR) and non-linear hybrid systems. Ideally, we would like to have a (VR)-based platform where different tools for computing sets of reachable states of hybrid automata could be plugged in. Some effort in this direction is already visible [4] at the level of input formats. It would be helpful to have a similar result at the level of output.

Another direction we intend to look into is moving towards hybrid I/O automata [34] as a more suitable model for PLCs.

Regarding modelling aspects, we also note that many of the hybrid systems we came across in the literature have global variables and in general they are described in a "monolithic" manner. This latter observation brings us to the issue of decomposing hybrid systems into components. As we have mentioned, in the temperature controller example, we did the decomposition by hand. To automate such a decomposition, a possible approach we could look into is the one from [38] which is based on strongly connected components. In a different direction, it would be of interest at what extent would results from algebraic geometry be useful.

Acknowledgements We would like to thank Chih-Hong Cheng for his friendly feedback and for sharing some ideas about possible ways to tackle decomposition.

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theor. Comput. Sci.*, 138, 1995.
2. L. Astefanoaei, S. B. Rayana, S. Bensalem, M. Bozga, and J. Combaz. Compositional invariant generation for timed systems. In *TACAS*, 2014.
3. L. Astefanoaei, S. B. Rayana, S. Bensalem, M. Bozga, and J. Combaz. Compositional verification of parameterised timed systems. In *NFM*, 2015.
4. S. Bak, S. Bogomolov, and T. T. Johnson. HYST: a source transformation and translation tool for hybrid automaton models. In *HSCC*, 2015.
5. S. Bensalem, M. Bozga, J. Sifakis, and T.-H. Nguyen. Compositional verification for component-based systems and application. In *ATVA*, 2008.
6. S. Bogomolov, A. Donzé, G. Frehse, R. Grosu, T. T. Johnson, H. Ladan, A. Podelski, and M. Wehrle. Abstraction-based guided search for hybrid systems. In *SPIN*, 2013.
7. M. Brockschmidt, D. Larraz, A. Oliveras, E. R. Carbonell, and A. Rubio. Compositional safety verification with max-smt. In *FMCAD*, 2015.
8. X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *CAV*, 2013.
9. X. Chen, S. Schupp, I. B. Makhlouf, E. Ábrahám, G. Frehse, and S. Kowalewski. A benchmark suite for hybrid systems reachability analysis. In *NFM*, 2015.
10. A. Cimatti. Application of SMT solvers to hybrid system verification. In *FMCAD*, 2012.
11. A. Cimatti, A. Griggio, S. Mover, and S. Tonetta. Parameter synthesis with IC3. In *FMCAD*, 2013.
12. A. Cimatti, A. Griggio, S. Mover, and S. Tonetta. Hycomp: An smt-based model checker for hybrid systems. In *TACAS*, 2015.
13. C. Courcoubetis and M. Yannakakis. Minimum and maximum delay problems in real-time systems. *Formal Methods in System Design*, 1992.
14. W. Damm, E. Möhlmann, and A. Rakow. Component based design of hybrid systems: A case study on concurrency and coupling. In *HSCC*, 2014.
15. A. David, K. G. Larsen, A. Legay, and D. B. Poulsen. Statistical model checking of dynamic networks of stochastic hybrid automata. *ECEASST*, 66, 2013.
16. A. Donzé. Breach, A toolbox for verification and parameter synthesis of hybrid systems. In *CAV*, 2010.
17. A. Eggers, N. Ramdani, N. S. Nedialkov, and M. Fränzle. Improving SAT modulo ODE for hybrid systems analysis by combining different enclosure methods. In *SEFM*, 2011.
18. G. Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations*. PhD thesis, Radboud Universiteit Nijmegen, 2005.
19. G. Frehse, C. L. Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. Spaceex: Scalable verification of hybrid systems. In *CAV*, 2011.
20. G. Frehse, S. K. Jha, and B. H. Krogh. A counterexample-guided approach to parameter synthesis for linear hybrid automata. In *HSCC*, 2008.

21. L. Fribourg and U. Kühne. Parametric verification and test coverage for hybrid automata using the inverse method. *Int. J. Found. Comput. Sci.*, 24, 2013.
22. P. Habermehl, R. Iosif, and T. Vojnar. What else is decidable about integer arrays? In *FOSSACS*, 2008.
23. T. A. Henzinger. The theory of hybrid automata. In *LICS*, 1996.
24. T. A. Henzinger, M. Minea, and V. S. Prabhu. Assume-guarantee reasoning for hierarchical hybrid systems. In *HSCC*, 2001.
25. T. A. Henzinger and H. Wong-Toi. Using HyTech to synthesize control parameters for a steam boiler. In *FMIA*, 1995.
26. H. Hermanns, J. Krčál, and J. Křetínský. Compositional verification and optimization of interactive markov chains. In *CONCUR*, 2013.
27. P.-H. Ho. *Automatic Analysis of Hybrid Systems.* PhD thesis, Cornell University, 1995.
28. M. S. Jaffe, N. G. Leveson, M. P. E. Heimdahl, and B. E. Melhart. Software requirements analysis for real-time process-control systems. *IEEE Trans. Softw. Eng.*, 17, 1991.
29. J. Jeannin and A. Platzer. dtl2: Differential temporal dynamic logic with nested temporalities for hybrid systems. In *IJCAR*, 2014.
30. T. T. Johnson and S. Mitra. A small model theorem for rectangular hybrid automata networks. In *FMOODS*, 2012.
31. T. T. Johnson and S. Mitra. Anonymized reachability of hybrid automata networks. In *FORMATS*, 2014.
32. S. Kong, S. Gao, W. Chen, and E. M. Clarke. dreach: δ-reachability analysis for hybrid systems. In *TACAS*, 2015.
33. A. Legay, S. Bensalem, B. Boyer, and M. Bozga. Incremental generation of linear invariants for component-based systems. In *ACSD*, 2013.
34. N. A. Lynch, R. Segala, and F. W. Vaandrager. Hybrid I/O automata. *Inf. Comput.*, 185, 2003.
35. L. Moura and N. Bjørner. Efficient e-matching for smt solvers. In *Proceedings of CADE*, 2007.
36. S. Mover. *Verification of Hybrid Systems using Satisfiability Modulo Theories*. PhD thesis, FBK-IRST/DIT, 2014.
37. S. Mover, A. Cimatti, A. Tiwari, and S. Tonetta. Time-aware relational abstractions for hybrid systems. In *EMSOFT*, 2013.
38. J. Oehlerking. *Decomposition of Stability Proofs for Hybrid Systems.* PhD thesis, Carl von Ossietzky Universität, Oldenburg, 2011.
39. J. Quesel and A. Platzer. Playing hybrid games with keymaera. In *IJCAR*, 2012.
40. F. Somenzi and A. R. Bradley. IC3: where monolithic and incremental meet. In *FMCAD*, 2011.
41. R. Testylier and T. Dang. NLTOOLBOX: A library for reachability computation of nonlinear dynamical systems. In *ATVA*, 2013.
42. L. Zhang, Z. She, S. Ratschan, H. Hermanns, and E. M. Hahn. Safety verification for probabilistic hybrid systems. *Eur. J. Control*, 18, 2012.