RTD-Finder: A Tool for Compositional Verification of Real-Time Component-based Systems

Souha Ben-Rayana, Marius Bozga, Saddek Bensalem, Jacques Combaz

▶ **To cite this version:**

Souha Ben-Rayana, Marius Bozga, Saddek Bensalem, Jacques Combaz. RTD-Finder: A Tool for Compositional Verification of Real-Time Component-based Systems. 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2016), Apr 2016, Eindhoven, Netherlands. pp.394-406, 10.1007/978-3-662-49674-9_23 . hal-01889137

HAL Id: hal-01889137
https://hal.science/hal-01889137

Submitted on 5 Oct 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

RTD-Finder: A Tool for Compositional Verification of Real-Time Component-based Systems

Souha Ben-Rayana, Marius Bozga, Saddek Bensalem, and Jacques Combaz

Univ. Grenoble Alpes, VERIMAG, F-38000 Grenoble, France
CNRS, VERIMAG, F-38000 Grenoble, France **

Abstract. In this paper we present RTD-Finder, a tool which applies a fully compositional and automatic method for the verification of safety properties for real-time component-based systems modeled in the RT-BIP language. The core method is based on the compositional computation of a global invariant which over-approximates the set of reachable states of the system. The verification results show that when the invariant catches the safety property, the verification time for large systems is drastically reduced in comparison with exploration techniques. Nevertheless, the above method is based on an over-approximation of the reachable states set expressed by the invariant, hence false positives may occur in some cases. We completed our compositional verification method with a counterexample-based invariant refinement algorithm analyzing iteratively the generated counterexamples. The spurious counterexamples which are detected serve to strengthen incrementally the global invariant until a true counterexample is found or until it is proven that all the counterexamples are spurious.

1 Introduction

The synchronous model of time makes the compositional verification of real-time systems a challenging task. State-of-the-art tools [7,10,21,19] for the verification of such systems rely mostly on exploration techniques. Consequently, they suffer from the state-space explosion for systems with a large number of components. The aim of compositional verification is to avoid such limitations. The basic idea is to infer properties of a system from the properties of its components and the interactions relating them. In general, as explained in [18], the compositional verification rules concentrate on the following idea: if components B_1 and B_2 meet respectively properties ϕ_1 and ϕ_1, if some condition $C(B_1, B_2)$ characterizes their parallel composition, and if these properties imply conjointly a property Ψ, then the system resulting from their composition satisfies Ψ.

In [9], a compositional verification rule was proposed for untimed systems. It is meant to prove invariance properties Ψ for systems built on an n-ary composition operation via an interaction set γ as follows:

** Work partially supported by the European Integrated Project STREP 318772 D-MILS.

$$\frac{B_1 \models \Box\phi_1, \quad B_2 \models \Box\phi_2, \quad II(\gamma), \quad \phi_1 \wedge \phi_2 \wedge II(\gamma) \Rightarrow \Psi}{\|_\gamma B_1, B_2 \models \Box\Psi} \text{ (D-Finder VR)}$$

In the above rule, $II(\gamma)$ is an interaction invariant expressing constraints on global locations resulting from the interaction structure. If the computed invariant $(\phi_1 \wedge \phi_2 \wedge II(\gamma))$ implies the safety property Ψ, then the system satisfies it. The above rule was implemented in the D-Finder tool [8] and was successful on several benchmarks. Nonetheless, the D-Finder tool does not handle time syntax. Furthermore, this rule is rather weak for timed systems. A straightforward adaptation of the D-Finder method to timed systems mostly yields false positives as shown in [4]. The main reason behind its weakness is that it does not capture time synchronization between components. In [4], we extended the above method precisely with the goal of offering a more successful application to timed systems. At the heart of the extension is the use of auxiliary *history clocks* (HC) in order to capture relations between the clocks of the different components. These clocks are added during the verification process and do not influence the behavior of the system. More concretely, to each action a, we associate an action history clock h_a which is reset whenever a occurs. The intuition behind this is that, on the one hand, history clocks are related to local clocks of their components thanks to the local invariants of those components and on the other hand, relations between history clocks of different components are inferred from the structure of the interactions. For ease of reference, we use $\mathcal{E}^*(\gamma)$ to denote all the additional clock constraints. Taking them all together, we obtain relations between the clocks of the different components in our global invariant. This invariant is made stronger, in case of conflicting interactions (that is, interactions which share actions) by introducing history clocks for interactions. New constraints on the interaction history clocks are gathered in the so-called $\mathcal{S}(\gamma)$ invariant. All in all, the verification rule for a system with n components can be written as follows:

$$\frac{B_1^h \models \Box\phi_1, B_2^h \models \Box\phi_2, II(\gamma), \mathcal{E}^*(\gamma), \mathcal{S}(\gamma), \phi_1 \wedge \phi_2 \wedge II(\gamma) \wedge \mathcal{E}^*(\gamma) \wedge \mathcal{S}(\gamma) \Rightarrow \Psi}{\|_\gamma B_1, B_2 \models \Box\Psi} \text{ (VR)}$$

where B_i^h represents the component B_i extended with action history clocks. The tool RTD-Finder presented in this paper is an implementation of such a rule in the context of RT-BIP, a component-based framework for real-time systems where components synchronize through multi-party interactions.

2 Tool Structure and Main Functionalities

The structure of RTD-Finder is depicted in Figure 1. The tool takes as input a Real-time (RT) BIP [1] source file and a safety property Ψ to check for invariance. If the property is not provided by the user, the tool proceeds by default to the verification of deadlock-freedom. Following this, it computes the predicate characterizing the set of deadlock states, the so-called DLK module. The tool

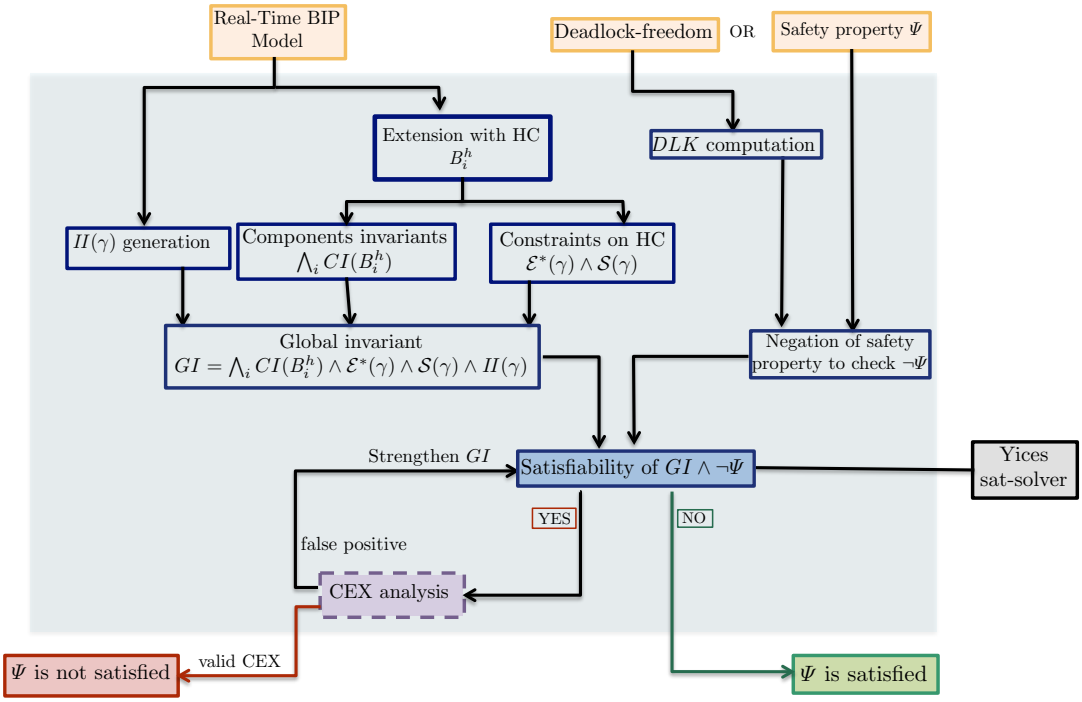

Fig. 1. RTD-Finder tool structure

extends each component B_i from the input file with history clocks (HC) into B_i^h. It then computes the invariants of B_i^h as the set of reachable symbolic states. Afterwards, it computes the interaction invariant and the inequalities on history clocks ($\mathcal{E}^*(\gamma)$ and $\mathcal{S}(\gamma)$). The combination of all the above invariants forms the global invariant GI. Together with the property Ψ, this invariant is input to Yices [13] [12], an SMT solver. If $GI \wedge \neg\Psi$ is unsatisfiable, the property is valid. Else, a counter-example is generated. A guided backward analysis module is developed to decide upon their validity (the dashed box in Figure 1).

2.1 The RT-BIP Framework

BIP (Behavior-Interaction-Priority) is a framework for modeling heterogeneous component-based systems. The BIP model is a superposition of three layers: the lowest layer models the behaviors of the components, the middle layer contains connectors describing interactions between the transitions of the different components and the top layer gathers priority rules to schedule among enabled interactions at a moment. Real-time (RT) BIP language extends BIP to support the continuous model of time where components are timed automata. Interested readers may refer to [1] for a detailed presentation.

2.2 Invariants Generation

RTD-Finder implements methods to compute components invariants, interaction invariants and the different constraints relating the history clocks:

Component Invariant Computation The component invariant is a local invariant proper to the component over-approximating its reachable states set. We compute it on the component extended with history clocks B_i^h. Intuitevely, the possible evaluations in a location l_i of component B_i^h is the disjunction of zones expressing constraints on the component clocks (including the history clocks). In our framework, the local invariant $CI(B_i^h)$ of B_i^h is computed as the set $Reach(B_i^h)$ of the reachable symbolic states which are computed by a depth-first-search algorithm. A symbolic state $s_i = (l_i, \zeta_i)$ of the component is defined by a location l_i and a zone ζ_i. In order to consider the operations and constraints on clocks during the computation of the reachability graph, we implemented various operations on zones and we included them in a DBM (Difference Bound Matrices [11,22]) library.

History Clocks Constraints Computation Constraints $\mathcal{E}^*(\gamma)$ and $\mathcal{S}(\gamma)$ relate local constraints obtained separately from the component invariants and by transitivity induce relations between inner clocks of the different components. Those constraints encode information like the fact that the history clock h_a of an action a is equal to the minimum among the history clocks of all the interactions to which it belongs. In fact, an action (resp. interaction) history clock is reset whenever the related action (resp. interaction) occurs. It results that the smaller the value of the history clock is, the more recent the related action (resp. interaction) is. The computation of these constraints is detailed in [4].

Interaction Invariant Generation Interaction invariant $II(\gamma)$ over-approximates the set of reachable global locations. It relates locations of different components and allows to disregard some unreachable configurations. As in [9], $II(\gamma)$ is computed by static analysis of the interaction structure. In order to implement it in the RT-BIP context, we make an abstraction from all timing aspects.

2.3 Checking Deadlock-freedom and Invariance Properties

Checking Invariance Properties After the computation of the global invariant GI, we export it to Yices sat-solver to check the satisfiability of the predicate $GI \wedge \neg\Psi$. The invariance safety properties follow this grammar:

$$\Psi ::= a \mid at(l_i) \mid \Psi_1 \wedge \Psi_2 \mid \neg\Psi$$

where a is an atomic clock constraint and $at(l_i)$ is a predicate expressing the presence of the component B_i at its location l_i.

If the predicate $(GI \wedge \neg\Psi)$ is not satisfiable, then the property Ψ is valid on the system and GI is strong enough to detect it. However, if a counter-example is generated, then RTD-Finder cannot conclude immediately on the validity of Ψ since the computed invariant is an over-approximation of the global reachable states set. A second stage of the tool aims at analyzing the generated counterexamples.

Checking Deadlock-freedom Deadlock predicate DLK expresses the set of global symbolic states from which all interactions are disabled. Checking deadlock freedom is equivalent to proving invariance of $\neg DLK$.

2.4 Counterexample-based Invariant Refinement

The method is proven to be sound. It is, however, incomplete: since it relies on an over-approximation of the reachable states set, a counter-example may satisfy the global invariant GI and be nonetheless unreachable. The violation of the desired safety property may be the outcome of some behavior in the over-approximation which does not belong to the original model. False positives appear particularly in heavily non-deterministic systems. To remedy this, we implemented a counter-example analysis module to decide the validity of the counter-examples returned by the sat-solver. Our approach is based on a backward state space search from the raised counter-example to the initial state. The algorithm removes iteratively false positives and verifies the existence of reachable bad states. It stops when a true counter-example is found or until no suspected counter-example remains, in which case we deduce that the property is valid.
The algorithm is shown in Figure 2. To describe it, we extend the notion of symbolic state from components to systems of parallel composition. The global location of a system is a n-tuple containing one location of each component and the zone of a global symbolic state is the conjunction of constraints relating the different components clocks.
The Yices SMT-solver generates well-defined locations of components and a precise valuation ν of clock variables of the counter-example (line 4). Therefore, a counter-example is perceived as $\theta = (l, \nu)$, where l is a global location and where the clock valuation ν is the conjunction of equalities of the form $x_{ij} = c_{ij}$. The variable x_{ij} is a clock of the component B_i^h and c_{ij} is the constant which it equals in the solution generated by the sat-solver.
As the clocks space is infinite ($\mathbb{R}$), we need to generalize the counter-example θ such that the algorithm terminates. Instead of considering only the counter example $\theta = (l, \nu)$, we analyze a set of counter-examples having the common global location l and gathered in a global symbolic state whose zone is generated as follows:

$$generalize(\theta, \Psi) = (l, \bigwedge_{z_k \in \mathcal{L}.\nu \models z_k} z_k \wedge \bigwedge_{z_k \in \mathcal{L}.\nu \not\models z_k} \neg z_k)$$

where $\mathcal{L}$ stands for the set of literals constraining the clocks in the property Ψ. The generalization reflects which literals of the safety property are satisfied by the counterexample or not. This generalization operation is implemented in the DBM library.
The backward computation starts from a generalized counterexample and computes iteratively its preimage, resulting at each step in a set of global symbolic states $\mathcal{P}$, until the initial state $\mathcal{I}$ is reached or until the preimage is empty.
To ensure termination, at each step, the visited symbolic states set $\mathcal{V}$ relative to the previous iterations is eliminated using the subtraction operator $\setminus$ in order to push the algorithm towards the initial state (line 11), else to conclude, if there is no intersection between $\mathcal{P}$ and $\mathcal{I}$ and if $\mathcal{P}$ is empty, that (l^θ, ζ^θ) does not contain any valid counterexample. The set $\mathcal{V}$ is cumulative: it contains the states that have been visited during the analysis of the previous counterexamples. They are all eliminated during the subtraction operation. If there exists a symbolic state $s_0 \in \mathcal{P} \cap \mathcal{I}$, then the length of the shortest path from s_0 to (l^θ, ζ^θ) is equal to the number of preimage computation operations required to reach s_0. For each analyzed counterexample, we note by the depth d the shortest path from (l^θ, ζ^θ) to the first backwards reachable state belonging to $\mathcal{I}$. If such a state does not exist, that is if the backward reachability algorithm reaches a set of symbolic states that has an empty preimage and has no intersection with $\mathcal{I}$, then the counterexample is spurious and the global invariant can be refined with its negation (line 16). We note that the operators $\setminus$ and $\cap$ on global symbolic states sets are slightly different from the usual set difference and conjunction operations on sets since symbolic states are defined by locations and zones. We consider the case where the zone of a symbolic state from a first set is strictly included in the zone of a symbolic state from another set and has its same location.

3 Experimentation

RTD-Finder is implemented in the Java programming language. It takes as input an RT-BIP file and a file where the property is expressed in Yices syntax. The tool saves all the computed invariants to an output Yices file and displays the verification result after the satisfiability checking of $GI \wedge \neg\Psi$ with Yices. If a counter example is found, then the Yices output is parsed as a symbolic state and is generalized with respect to the safety property. If a counterexample is spurious, its negation is conjoined with the global invariant in the Yices file and the satisfiability of $GI \wedge \neg\Psi$ is further checked.

We show in this section the experimental results for four benchmarks with different properties for each of them.

Train Gate Controller The first example is the classical train gate controller (TGC) system, where a controller, a gate and a number of trains interact together. We verified two properties:

```
1  GI := ⋀ CI(B_i^h) ∧ E*(γ) ∧ S(γ) ∧ II(γ) ;
2  V := ∅ ;
3  while GI ∧ ¬Ψ is satisfiable do
4  |  Let θ a solution of GI ∧ ¬Ψ ;
5  |  Let (l^θ, ζ^θ) := generalize(θ, Ψ) ;
6  |  Let P := {(l^θ, ζ^θ)} ;
7  |  while P ∩ I = ∅ and P ≠ ∅ do
8  |  |  V := V ∪ P ;
9  |  |  P := pre(P) \ V ;
10 |  end
11 |  if P ∩ I ≠ ∅ then
12 |  |  stop ;
13 |  |                          // The counterexample is valid
14 |  else
15 |  |  GI := GI ∧ ¬(at(l^θ) ∧ ζ^θ) ;
16 |  |                          // The counterexample is spurious
17 |  end
18 end
19 if GI ∧ ¬Ψ is not satisfiable then
20 |  Ψ is satisfied.
21 end
```

Fig. 2. Counterexample-based invariant refinement algorithm

1. (P_1) Utility property: The gate does not go down if all the trains are far from the crossing.
2. (P_2) Safety property: The gate is down when a train is in the crossing.

While the generated invariant was strong enough to verify the utility property, a spurious counterexample raises for the P_2 property. The counterexample-based refinement algorithm was necessary in this case.

Temperature Control System The second example is a timed adaptation of the temperature control (TC) system in [9]. It represents a simplified model of a nuclear plant. The system consists of a controller interacting with an arbitrary number of rods in order to maintain the temperature within some bounds. When the reactor spends 900 units of time in heating, a rod must be used to cool the reactor. We verified two properties:

1. (P_3) At least one rod is ready to take *cool* action together with the controller when necessary.
2. (P_4) No rod is in cool location if the controller and the other rods are in heat position.

While (P_3) property is implied by the computed invariant, the counterexample analyis module is needed for (P_4) property.

The TGC (resp. TC) example is run with great numbers of trains (resp. rods) in order to show the scalability of the method.

Gear Controller System The third benchmark is taken from [20] and models a gear controller system embedded inside vehicles. It is composed of an interface sending gear change requests to a gear controller component which interacts with an engine, a clutch and a gear-box component. In order to ensure the system correctness, some requirements have to be met. We verified the following properties after making abstraction from the data variables of the system:

1. (P_5) *Predictability*: When the engine is regulating the torque, the clutch should be closed.
2. (P_6) Error detection: The controller detects and indicates the precise errors when the clutch is not opened or closed at time and when the gear-box is unable to set or release a gear at time. (P_6) gathers 4 state properties.
3. (P_7) The gear controller system is deadlock-free.

The computed invariant was strong enough to verify all the above-mentioned correctness properties required for the gear control system.

Dual Chamber Implantable Pacemaker We considered the verification of a dual chamber implantable pacemaker modeled and verified in [16]. The system is designed to manage the cardiac rhythm. In the considered pacemaker mode, both the atrium and ventricle of the heart are paced. Based on the sensing of both chambers, the pacing can be restrained or activated. For a safe operation, it is essential that the ventricles of the heart should not be paced beyond a maximum rate equal to a $TURI$ constant. A ventricle pace (VP) can occur at least TURI time units after a ventricle event. This requirement expresses the *Upper rate limit* property. We summarize the verified properties in the following:

1. (P_8) There is a minimum time elapse $TURI$ between a ventricle (VS) sense and a ventricle pace (VP) event.
2. (P_9) There is a minimum time elapse $TURI$ between two ventricle pace (VP) events.
3. (P_{10}) The pacemaker system is deadlock-free.

As in [16], we verified both of (P_8) and (P_9) properties by translating them into a monitor component. Besides, our method offers another way to check the first property without resorting to the monitor as it can be expressed by means of the already introduced history clocks. In fact, the difference between the interactions history clocks relative to those two events is bigger than the desired time elapse. However, using history clocks to express the safety requirement is not possible for the second property since it compares two occurrences of the same action.
The global invariant is strong enough to catch the (P_8) property. Nevertheless,

the counterexample analysis is necessary to eliminate 28 raised spurious counterexamples appearing during the verification of (P_9) property. RTD-Finder verified also deadlock-freedom (P_{10}) after eliminating 11 spurious counterexamples.

Experimental Results

Table 1 gives an overview of the experimental results relative to the verification of the properties where no counterexample raises. In this table, n is the number of components in the considered example, q is the total number of control locations of its components and c (resp. h) is the number of system clocks (resp. actions history clocks) and $|\gamma|$ is the number of interactions. Finally, t shows the total verification time required for GI invariant computation and satisfiability checking of $GI \wedge \neg\Psi$ and t_{yices} specifies the satisfiablity checking time required by the sat-solver.

The tool and the detailed output results are available at `http://www-verimag.imag.fr/RTD-Finder`. We made a comparison of RTD-Finder with the monolithic verification tool UPPAAL based on the complete method of model-checking. UPPAAL incorporates reduction techniques that are very successful on some benchmarks, like the train-gate-controller system. Yet, in general, the state-space exploration has its costs. We made a comparison with RTD-Finder on TC system. For 10 rods, UPPAAL generated no results after five hours and 436519 explored states. Nevertheless, RTD-Finder checked the property for 300 rods in a few minutes, as shown in Table 1. All the experiments are run on Linux machine Intel Core 3.20 GHz $\times 4$ and 15.6 GiB Ram.

Model	Property	n	q	c	$\|\gamma\|$	h	t	t_{yices}
Train gate controller (50 trains)	P_1	52	158	52	102	106	$0.5s$	$0.3s$
Train gate controller (100 trains)	P_1	102	308	102	202	206	$5.3s$	$0.6s$
Train gate controller (200 trains)	P_1	202	608	202	402	406	$1m33s$	$5s$
Train gate controller (300 trains)	P_1	302	908	302	602	606	$9m8s$	$20s$
Train gate controller (500 trains)	P_1	502	1508	502	1002	1006	$1h13m20s$	$2m52s$
Temperature control (20 rods)	P_3	21	42	21	40	42	$0.07s$	$0.01s$
Temperature control (50 rods)	P_3	51	102	51	100	102	$0.35s$	$0.04s$
Temperature control (100 rods)	P_3	101	204	102	200	204	$3.7s$	$0.08s$
Temperature control (300 rods)	P_3	301	602	302	600	602	$5m47s$	$0.9s$
Gear controller	P_5, P_6	5	65	4	17	32	$15.1s$	$0.14s$
Gear controller	P_7	5	65	4	17	32	$17.6s$	$0.04s$
Pacemaker (with monitor)	P_8	7	19	11	6	21	$0.25s$	$0.004s$
Pacemaker (without monitor)	P_8	6	16	9	6	19	$0.24s$	$0.004s$

Table 1. Results from experiments where no counterexample raises

All of the properties shown in Table 1 are checked without resorting to the counterexample analysis module, that is the computed invariant catches them. At the opposite, the other properties are not implied by the invariant and for that the generated counterexamples have been analyzed. The results are shown in Table 2. For a spurious counterexample, d is the length of the path from the suspected state (l^θ, ζ^θ) to the symbolic states set $\mathcal{P}$ having an empty preimage. Intuitively, the depth d is the number of backward computation steps required to deduce the invalidity of the counterexample. The number d_{max} is the maximum depth d amongst the analyzed spurious counterexamples. By p, we note the total number of all the symbolic states computed and visited during the backward analysis and contained in the $\mathcal{P}$ sets, while by k_{cex} we refer to the number of analyzed counterexamples. The total verification time is t_{cex}.

The verification time is visibly less important when the invariant is strong

Model	Property	n	$\|\gamma\|$	d_{max}	p	k_{cex}	t_{cex}
Train gate controller (3 trains)	P_2	5	8	22	440	1	$0.6s$
Train gate controller (5trains)	P_2	7	12	22	2452	1	$3.2s$
Train gate controller (10 trains)	P_2	12	22	22	22982	1	$45s$
Train gate controller (20 trains)	P_2	22	42	22	199192	1	$19m45s$
Temperature control (3 rods)	P_4	4	6	7	48	1	$0.2s$
Temperature control (5 rods)	P_4	6	10	7	218	1	$0.6s$
Temperature control (20 rods)	P_4	21	40	7	8914	1	$3m46s$
Temperature control (50 rods)	P_4	51	100	7	128794	1	$14h14m$
Pacemaker (with monitor)	P_9	7	3	6	126	28	$0.6s$
Pacemaker	P_{10}	7	3	4	93	11	$0.5s$

Table 2. Results from experiments where counterexamples analysis is needed

enough to detect the desired property, which was the case for all the properties shown in Table 1. In some cases, even when counterexample analysis is needed, RTD-Finder remains competitive to model checking using forward reachability analysis. This is for instance the case of the temperature control system and (P4) property which is verified in 3 minutes for 20 rods using the counterexample analysis module compared to the inability to check the property in 5 hours for 10 rods with UPPAAL.

It is worth noting that some symmetry reduction techniques can be applied in order to ameliorate the performances of the counterexample-guided invariant refinement algorithm. Some of them were already applied under the model-checking tool UPPAAL [14]. Symmetry reduction would notably allow to refine the global invariant not only with the negation of a given counterexample proven to be spurious, but also with a set of counterexamples to which it is symmetric. It would also serve to reduce the complexity of the backward computation.

In Table 2, we notice that for the Train-gate-controller (resp. temperature control) system, the number of backward steps d_{max} necessary to deduce the invalid-

ity of the counterexample remains the same, independently from the number of replicated trains (resp. rods) in the system. This suggests some similar behaviors among systems differing only on the copies number of the replicated component and motivates the consideration of an extension for the verification of parameterized timed systems adapted to our compositional verification method.
We propose an extension of RTD-Finder precisely with the goal of offering uniform verification for a class of parameterized timed systems.

4 Ongoing Extensions

Our ongoing extensions are multifold and focus mainly on the expressive power of the system properties, and the class of systems itself.

Uniform Verification of Parameterized Timed systems In [5], we proposed an extension of (VR) to the verification to parameterized timed systems. Parameterized systems are those which rely on multiple copies of a given component. Illustrative examples are satellite systems, swarm robots, ad-hoc networks, device drivers, or multi-threaded programming. The main purpose is to establish system safety independently from the number of components. It turned out that typical small model results lend themselves well to the parameterized timed systems we considered. The main idea is that it is sufficient to apply (VR) for all the systems with less than n_0 identical components in order to conclude correctness for the systems with any number of copies. The bound n_0 is computed statically from the number of quantifiers in (VR). With n_0 at hand, the tool computes the global invariant for any $k \leq n_0$ and checks if the property is satisfied. The steps are precisely those depicted in Figure 1.
If we consider the TGC system and deadlock-freedom, then following the small model theorem, it suffices to check the property for all numbers of trains ranging from 1 to 5. The total RTD-Finder verification time for those *small models* is $1.4s$. We applied this method also on a timed version of a token ring system. We check that at any time, exactly one of the processes possesses the token (i.e is in a *busy* location). The total verification time of the small models, containing from 1 to 5 processes, is $0.4s$.
The present verification approach for parameterized timed systems states that if the global invariant implies the property for the small models, then it is verified for all numbers of the replicated component. We want to complete it in order to cover the cases where the refinement of the invariant with the negation of the spurious counter examples is needed. Since the backward reachability computation is in general practical for small models, this extension of our uniform verification method would drastically reduce the verification time even when raised false positives are eliminated.

Verification of Timed Systems with Parameters and Data One interesting extension of RTD-Finder is concerns parametric timed systems which

contain timing constraints defined by use of parameters. These parameters may range over infinite domains and are in general related by a timing constraints set. Since the lack of restriction makes emptiness undecidable, the verification of systems composed of parametric timed automata is even harder. It requires data structures that handle efficiently and compactly the configurations that are introduced by the plentifulness of parameters. Inspired by existing work [15,2,17], a feasible approach consists in extending DBMs to parametric DBMs [3] and existentially quantifying (VR) such that the prover returns concrete values of the parameters ensuring the desired safety property for the system. As most of the existing tools are based on exploring the whole state space, they handle a relatively small number (mostly below ten) of automata, the immediate advantage of our approach is in scalability. Another possible extension is the consideration of richer classes of models, handling data variables and urgency types on transitions [6]. This is not a trivial task since urgency does not lend itself to a compositional definition.

Properties Currently, RTD-Finder handles only state safety properties but the extension to check LTL properties by use of Timed Büchi automata is possible. As for the pacemaker example, history clocks may help to express some LTL properties without resorting to the monitors.
RTD-Finder offers high scalability especially when the property to check is not combinatorial. Checking the absence of deadlock is currently more problematic. If in the untimed case one can provide an exact formalization of deadlock by means of local characterizations, this is no longer the case for timed systems. More precisely, the condition which expresses that an interaction is eventually enabled in a timed setup cannot be decided by the consideration of the involved components only, but depends also from the timing constraints of the non involved components. We are working on some approximation techniques in order to avoid its full computation.

Acknowledgement The authors would like to thank Lacramioara Aştefănoaei for her contribution to the construction of the global invariant and to the verification of parameterized timed systems.

References

1. T. Abdellatif, J. Combaz, and J. Sifakis. Model-based implementation of real-time applications. In *Proceedings of the 10th International conference on Embedded software, EMSOFT*, pages 229–238, 2010.
2. É. André and R. Soulat. Synthesis of timing parameters satisfying safety properties. In *Proceedings of Reachability Problems - 5th International Workshop, RP*, pages 31–44, 2011.

3. A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *Computer Aided Verification, 12th International Conference, CAV*, pages 419–434, 2000.
4. L. Astefanoaei, S. B. Rayana, S. Bensalem, M. Bozga, and J. Combaz. Compositional invariant generation for timed systems. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS*, pages 263–278, 2014.
5. L. Astefanoaei, S. B. Rayana, S. Bensalem, M. Bozga, and J. Combaz. Compositional verification of parameterised timed systems. In *Proceedings of NASA Formal Methods - 7th International Symposium, NFM*, pages 66–81, 2015.
6. A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in BIP. In *Fourth IEEE International Conference on Software Engineering and Formal Methods SEFM*, pages 3–12, 2006.
7. G. Behrmann, A. David, K. G. Larsen, J. Håkansson, P. Pettersson, W. Yi, and M. Hendriks. UPPAAL 4.0. In *Third International Conference on the Quantitative Evaluation of Systems QEST*, pages 125–126, 2006.
8. S. Bensalem, M. Bozga, T. Nguyen, and J. Sifakis. D-finder: A tool for compositional deadlock detection and verification. In *Proceedings of Computer Aided Verification, 21st International Conference, CAV*, pages 614–619, 2009.
9. S. Bensalem, M. Bozga, J. Sifakis, and T. Nguyen. Compositional verification for component-based systems and application. In *Proceedings of Automated Technology for Verification and Analysis, 6th International Symposium, ATVA*, pages 64–79, 2008.
10. M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A model-checking tool for real-time systems. In *Proceedings of Computer Aided Verification, 10th International Conference, CAV*, pages 546–550, 1998.
11. D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proceedings of Automatic Verification Methods for Finite State Systems, International Workshop*, pages 197–212, 1989.
12. B. Dutertre. Yices 2.2. In *Proceedings of Computer Aided Verification - 26th International Conference, CAV*, pages 737–744, 2014.
13. B. Dutertre and L. de Moura. The Yices SMT solver. Technical report, SRI International, 2006.
14. M. Hendriks, G. Behrmann, K. G. Larsen, P. Niebert, and F. W. Vaandrager. Adding symmetry reduction to uppaal. In *Formal Modeling and Analysis of Timed Systems: First International Workshop, FORMATS*, pages 46–59, 2003.
15. T. Hune, J. Romijn, M. Stoelinga, and F. W. Vaandrager. Linear parametric model checking of timed automata. *J. Log. Algebr. Program.*, 52-53, 2002.
16. Z. Jiang, M. Pajic, S. Moarref, R. Alur, and R. Mangharam. Modeling and verification of a dual chamber implantable pacemaker. In *TACAS*, 2012.
17. A. Jovanovic, D. Lime, and O. H. Roux. Integer parameter synthesis for timed automata. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS*, pages 401–415, 2013.
18. O. Kupferman and M. Y. Vardi. Modular model checking. In *Compositionality: The Significant Difference, International Symposium, COMPOS*, pages 381–401, 1997.
19. D. Lime, O. H. Roux, C. Seidner, and L. Traonouez. Romeo: A parametric model-checker for petri nets with stopwatches. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference, TACAS*, pages 54–57, 2009.

20. M. Lindahl, P. Pettersson, and W. Yi. Formal design and analysis of a gear controller. In *Proceedings of Tools and Algorithms for Construction and Analysis of Systems, 4th International Conference, TACAS*, pages 281–297, 1998.
21. F. Wang. Redlib for the formal verification of embedded systems. In *Leveraging Applications of Formal Methods, Second International Symposium, ISoLA*, pages 341–346, 2006.
22. S. Yovine. Model checking timed automata. In *Lectures on Embedded Systems, European Educational Forum, School on Embedded Systems*, pages 114–152, 1996.