Compositional Verification of Timed Systems

Saddek Bensalem[*]

Abstract

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

We addressed the problem of state-explosion inherent to model-checking of timed systems built from large number of components. Our solution consists in adapting the compositional verification approach of [BBSN08] to timed systems. The main challenge was to be able to capture the relations between the local timing of the components induced by their interactions. Without them the proposed compositional analysis proved to be too weak for verifying even simple systems. The proposed relations take the shape of equalities between the clocks of components used for expressing their timing constraints. We proved the soundness of the proposed approach, and successfully applied it to academic examples and non trivial case studies.

Compositional methods in verification have been developed to cope with state explosion. Generally based on divide-and-conquer principles, compositional methods attempt to break monolithic verification problems into smaller sub-problems by exploiting either the structure of the system or of the property to verify, or both. Compositional reasoning can be used in different flavors, e.g. deductive verification, assume-guarantee reasoning [MC81, Jon83, Pnu84], contract-based verification [DLL$^+$12, LLH$^+$12], compositional generation, etc. Nonetheless, compositional verification has not been very successfully applied to timed systems. The most used tools for the verification of such systems are based on symbolic state space exploration, using efficient data structures and exploration techniques. Few attempts have been made however for exploiting compositionality principles but they remain marginal in the research literature. Nowadays, it is generally admitted that the difficulty for using compositional reasoning is inherently due to the synchronous model of time. Time progress hides continuous synchronization of all the components of the system. Getting rid of such synchronization is necessary for analyzing independently different parts of the system (or of the property) but also extremely problematic when attempting to re-compose the partial verification results.

We proposed a different approach for exploiting compositionality for analysis of timed systems. We developed a novel compositional method for the generation of invariants for timed systems. In contrast to exact reachability analysis, invariants are symbolic approximations of the set of reachable states of the system. We show that quite precise invariants can be computed compositionally, from the separate analysis of the components in the system and from their composition glue. This method is sound for verification of safety properties, that is, if a given property can be deduced from the invariants computed for the system, then the system is guaranteed to satisfy that property. However, the method is not complete, that is, it may be not able to prove certain properties even if they are satisfied by the system.

Our Compositional Verification Method. The compositional method we propose here is based on the verification rule (VR) from [BBSN08]. Assume that a system consists of n components B_i interacting by means of an interaction set γ, and that the system property of interest is Φ. If components B_i, respectively interactions γ, can be locally characterized by means of invariants $CI(B_i)$, respectively $II(\gamma)$, and if Φ can

[*]This is joint work with L. Aştefănoaei, S. Ben Rayana, M. Bozga, J. Combaz.

be proved to be a logical consequence of the conjunction of the local invariants, then Φ is a global invariant. This is what the rule below synthesizes.

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

In the rule (VR), the symbol $\vdash$ is used to underline that the logical implication can be effectively proved (for instance with an SMT solver) and the notation $B \models \Box\Phi$ is to be read as "Φ holds in every reachable state of B".

The key idea behind the compositional generation method is to use additional *history clocks* in order to track the timing of interactions between different components. History clocks allow to decouple the analysis for components and for their composition. On component level, history clocks are used to capture and expose the local timing constraints relevant to their interactions. At composition level, extra constraints on history clocks are enforced due to simultaneity of interactions and to the synchrony of time progress.

Timed Systems. In our framework, the components are timed automata [AD94] and systems are compositions of timed automata with respect to n-ary interactions. Timed automata represent the behavior of components. They have control locations and transitions between these locations. Transitions may have timing constraints, which are defined on clocks. Clocks can be reset and/or tested along with transition execution. Formally, a timed automaton is tuple $(L, l_0, A, T, X, \mathsf{tpc})$ where L is a finite set of control locations, l_0 is an initial control location, A a finite set of actions, X is a finite set of clocks, $T \subseteq L \times (A \times \mathcal{C} \times 2^X) \times L$ is finite set of transitions labeled with actions, guards, and a subset of clocks to be reset, and $\mathsf{tpc} : L \rightarrow \mathcal{C}$ assigns a time progress condition[1] to each location. $\mathcal{C}$ is the set of timing constraints which are predicates on the clocks X defined by the following grammar:

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

with $x, y \in X$, $\# \in \{<, \leq, =, \geq, >\}$ and $ct \in \mathbb{Z}$. Time progress conditions are restricted to conjunctions of constraints as $x \leq ct$. For simplicity, we assume that at each location l the guards of the outgoing transitions imply the time progress condition $\mathsf{tpc}(l)$ of l.

A timed automaton is a syntactic structure whose semantics is based on continuous and synchronous time progress. That is, a state is given by a control location paired with real-valued assignments of the clocks. From a given state, a timed automaton can let time progresses when permitted by the time progress condition of the corresponding location, or execute a (discrete) transition if its guard evaluates to true. The effect of time progress of $\delta > 0$ is to increase synchronously all the clocks by the the real value δ. Executions of transitions are instantaneous, that is, they keep values of clocks unchanged except the ones that are reset (i.e. assigned to 0). Because of their continuous semantics, timed automata have in general infinite state spaces. However, they admit finite symbolic representations of their state spaces called zone graph [ACD$^+$92, Alu99, HNSY94, YPD94], in which equivalent assignments of clocks are grouped in a single (symbolic) state call *zone* having the shape of timing constraints defined previously. That is, the reachable states of a timed automata corresponds to a finite number of configurations (l_j, ζ_j), $1 \leq j \leq m$, where for all j, l_j is a control location and ζ_j is a timing constraint.

Examples of timed automata are provided by Figure 1. For instance, components $Worker_i$, $i \in \{1, 2\}$, are implemented by similar timed automata, consisting of two control locations l_1^i and l_2^i and two transitions: a transition from l_1^i to l_2^i labelled by action b_i and having timing constraint $y \geq 8$, and a transition from l_2^i to l_1^i having action d_i and resetting clock y. By convention non displayed guards of transitions and time progress conditions of locations are *true*.

In our framework, components interact by means of strong synchronization between their actions. The synchronizations are specified in the so called *interactions* as sets of actions. An interaction can involve at

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

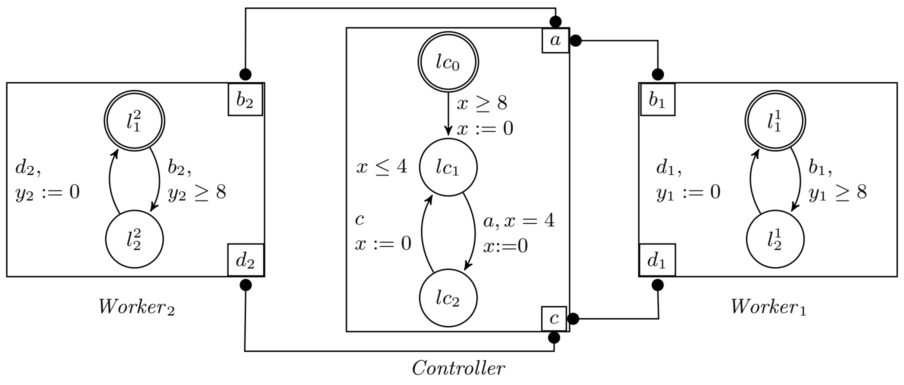

Figure 1: A composition of three timed automata.

most one action of each component. Given n components (i.e. timed automata) $B_i = (L^i, l_0^i, A^i, T^i, X^i, \mathsf{tpc}^i)$, $1 \leq i \leq n$, and a set of interactions γ, we denote by $\gamma(B_1, \ldots, B_n)$ the composition of components B_i with respect to interactions γ. States of the composition $\gamma(B_1, \ldots, B_n)$ are combinations of the states of the components B_i. In $\gamma(B_1, \ldots, B_n)$, a component B_i can execute an action a_i only as part of an interaction $\alpha \in \gamma$, $a_i \in \alpha$, that is, along with the execution of all the actions participating to α, which corresponds to the usual notion of multi-party interaction. Notice that for a component B_i of a composition $\gamma(B_1, \ldots, B_n)$, the application of interactions γ can only restrict its reachable states. That is, the reachable states of B_i when executed in the composition $\gamma(B_1, \ldots, B_n)$ are included in the reachable states of B_i executed alone (i.e. as a single timed automata). This property is essential for building our compositional verification method, presented below.

Components and Interaction Invariants. To give a logical characterization of a system $S = \gamma(B_1, \ldots, B_n)$ we use invariants. An invariant Φ is a state property which holds in every reachable state of S, in symbols, $S \models \Box\Phi$.

Component invariants $\mathit{CI}(B_i)$ characterize the reachable states of components B_i when considered alone. Such invariants can easily be computed from the zones of the corresponding timed automata. More precisely, given the reachable (symbolic) states (l_j, ζ_j), $1 \leq j \leq m$, of component B_i, the invariant for B_i is defined by:

$$\bigvee_{1 \leq j \leq m} l_j \wedge \zeta_j,$$

where we abuse of notation and use l_j for the predicate that holds whenever B_i is at location l_j. Notice that zones ζ_j are timing constraints, that is, predicates on clocks. Notice also that invariants $\mathit{CI}(B_i)$ still hold for the composed system $S = \gamma(B_1, \ldots, B_n)$, but are only over approximations of the states reached by each component B_i in S. For example, the component invariants for *Controller*, Worker_1 and Worker_2 of Figure 1 are as follows:

$$\begin{aligned} \mathit{CI}(\mathit{Controller}) &= (lc_0 \wedge x \geq 0) \vee (lc_1 \wedge x \leq 4) \vee (lc_2 \wedge x \geq 0) \\ \mathit{CI}(\mathit{Worker}_i) &= (l_1^i \wedge y_i \geq 0) \vee (l_2^i \wedge y_i \geq 8). \end{aligned}$$

Interaction invariants $\mathit{II}(\gamma)$ are induced by the synchronizations and have the form of global conditions involving control locations of components. In previous work, we have considered boolean conditions [BBSN08] as well as linear constraints [SBL12] for $\mathit{II}(\gamma)$. For instance, such invariants exclude configurations such that $lc_1 \wedge l_2^i$, that is, they establish $\neg\big(lc_1 \wedge (l_2^1 \vee l_2^1)\big)$. Interaction invariants are not the main purpose of this work, interested readers should refer to [BBSN08] and [SBL12] for detailed presentations.

A safety property of interest for example of Figure 1 is absence of deadlocks. A necessary condition for deadlock freedom is that a can synchronize with b_1 or b_2 when the controller is at lc_1 and the workers are at l_1^i, that is, $\Phi = l_{c_1} \wedge l_1^1 \wedge l_1^2 \implies y_1 - x \geq 4 \vee y_2 - x \geq 4$. Even if Φ holds in S, it cannot be proved by applying (VR) using only component invariants $CI(B_i)$ and interaction invariant $II(\gamma)$. A counter example is given by $lc_1 \wedge l_1^1 \wedge l_1^2$ and $x = y_1 = y_2 = 0$, which satisfies the invariant $CI(Controller) \wedge CI(Worker_1) \wedge CI(Worker_2) \wedge II(\gamma)^2$ but violate property Φ, that is, $CI(Controller) \wedge CI(Worker_1) \wedge CI(Worker_2) \wedge II(\gamma) \not\Longrightarrow \Phi$. One problem is that the proposed invariants cannot relate values of clocks of different components according to their synchronizations (e.g. synchronous reset of clocks).

Adding History Clocks. To strengthen computed invariants, we proposed to equip each component B_i (and later, interactions) with *history clocks*: one clock h_{a_i} per action of a_i of B_i. A history clock h_{a_i} is reset on all transitions executing a_i. Notice that since there is no timing constraint involving history clocks, the behavior of the components remain unchanged after the addition of the history clocks, which shown in [ARB+13]. They are only introduced for establishing properties. Each time an interaction $\alpha \in \gamma$ is executed, all the history clocks corresponding to the actions participating in α are reset synchronously, and then become identical at the next state (until another interaction is executed). Moreover, history clocks of actions of the last executed interaction α are necessarily lower than the ones of actions not participating in α, since they are the last being reset. This is captured by the following invariant:

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

where $\gamma \ominus \alpha = \{\beta \setminus \alpha \mid \beta \in \gamma \wedge \beta \not\subseteq \alpha\}$. It can be shown that $\mathcal{E}(\gamma)$ is an invariant of the system [ARB+13]. For example of Figure 1, invariant $\mathcal{E}(\gamma)$ is given by:

$$\mathcal{E}(\gamma) \quad = \quad (h_a = h_{b_1} \leq h_{b_2} \vee h_a = h_{b_2} \leq h_{b_1}) \wedge (h_c = h_{d_1} \leq h_{d_2} \vee h_c = h_{d_2} \leq h_{d_1}).$$

Component invariants for example of Figure 1 including the history clocks are as follows:

$$\begin{aligned} CI(Controller^h) &= lc_0 \vee (lc_1 \wedge x \leq 4 \wedge (h_a = h_c \geq 8 + x \ \vee x = h_c \leq h_a)) \vee \\ &\qquad (lc_2 \wedge x = h_a \wedge (h_c \geq h_a + 12 \vee h_c = h_a + 4)) \\ CI(Worker_i^h) &= (y = h_{d_i} \wedge l_1^i \wedge h_{d_i} \leq h_{b_i}) \vee (l_2^i \wedge h_{d_i} \geq 8 + h_{b_i}). \end{aligned}$$

Such invariants proved to be sufficient for stating deadlock-freedom for a similar example involving only one worker, but are too weak for establishing deadlock-freedom with two workers. When interactions are conflicting on shared action a_i, the proposed invariants for history clock h_{a_i} always consider that any of these interactions can execute. For instance, in example of Figure 1 our invariants cannot capture the fact that if action a of *Controller* synchronizes with b_1 of *Worker*$_1$, then the following execution of action c of *Controller* can only synchronize with d_1 of *Worker*$_1$ (it cannot synchronize with d_2 of *Worker*$_2$).

Handling Conflicting Interactions. We developed a general way for computing stronger invariants relating execution of the interactions. The principle is to add again history clocks h_α for each the interaction α of γ, and to reset h_α each time α is executed by the means of an additional component and adequate synchronizations. A full description of this approach can be found in [ARB+13]. For an action a_i of component B_i, we define the separation constraint $\mathcal{S}(\gamma, a_i)$ as:

$$\mathcal{S}(\gamma, a_i) = \bigwedge_{\substack{\alpha, \beta \in \gamma \ \mid \ a_i \in \alpha, \beta \\ \alpha \neq \beta}} \mid h_\alpha - h_\beta \mid \geq \delta_{a_i},$$

[2]Notice that interaction invariants cannot exclude $lc_1 \wedge l_1^1 \wedge l_1^2$ since it is a reachable configuration.

where δ_{a_i} is a lower bound of the time elapsed between two consecutive executions of a_i in B_i, which can be statically computed from the timed automata of B_i. It can be shown [ARB$^+$13] that separation constraints $\mathcal{S}(\gamma, a_i)$ are invariants of the system, that is, the following is an invariant of the system:

$$\mathcal{S}(\gamma) = \bigwedge_{1 \le i \le n} \bigwedge_{a_i \in A_i} \mathcal{S}(\gamma, a_i).$$

Invariant $\mathcal{E}(\gamma)$ can be rewritten using additional history clocks as follows:

$$\mathcal{E}(\gamma) = \bigwedge_{1 \le i \le n} \bigwedge_{a_i \in A_i} h_{a_i} = \min_{\alpha \ni a_i} h_\alpha.$$

This corresponds to the intuition that the history clock of an action a_i equals the history clock of the last executed interaction α involving a_i, which is the one having h_α minimal.

Experimental Results. We have developed a prototype in Scala implementing the approach. It takes as input components B_i, interactions γ, and a global safety property Φ, and checks whether the system satisfy Φ. To this end, it first computes the invariants proposed above, using PPL[3]. Then it generates Z3[4] Python code to check the satisfiability of the following formula:

$$\bigwedge_{1 \le i \le n} CI(B_i) \wedge II(\gamma) \wedge \mathcal{E}(\gamma) \wedge \mathcal{S}(\gamma) \wedge \neg\Phi. \tag{1}$$

Notice that when γ has no conflicting interactions we can simply use the initial form for $\mathcal{E}(\gamma)$ and discard $\mathcal{S}(\gamma)$. If (1) is not satisfiable then the system is guaranteed to satisfy Φ (i.e. our approach is sound). Otherwise, Z3 returns an assignment of the variables satisfying (1) and corresponding to a global state of the system that violates property Φ. Since we use over-approximations (i.e. invariants) instead of the exact behavior of the system, this state may be not reachable and Φ may actually hold in the system.

We experimented the approach on several classical examples, namely the *Train-Gate-Controller* (*TGC*), the *Fischer* mutual exclusion protocol, and the *Temperature-Control-System* (*TCS*). We compared our prototype implementation with Uppaal[5]. Uppaal is a widely used model-checker for timed systems implementing symbolic reachability of parallel composition of timed automata using zones. We measured execution times for verifying properties of interest for these examples, i.e. mutual exclusion for *TGC* and *Fischer* and deadlock-freedom for *TCS* (see Table 1). Experimental results shown that Uppaal is subject to state-explosion when increasing the number of components, which happened with *TCS* for 16 components or more, and with *Fischer* for 14 components or more. In contrast, our prototype managed to verify *TCS* even for 124 components in less than 20 seconds. We believe that such compositional approach is very interesting for systems composed of large number of identical components (e.g. swarms of robots) since in this case we reuse already computed invariants following incremental approaches of [BGL$^+$11].

Conclusion. We have presented a compositional verification method for systems subject to timing constraints. It relies on invariants computed separately from system components and their interactions. This method is sound for verification of safety properties, that is, it can be used to prove that the system cannot reach an undesirable configuration. We believe that it is suited to check correctness of coordinations within distributed systems, usually implemented by communication protocols relying on time. Its applicability has already been considered in an European project for an ensemble of robots and devices coordinating in real-time to build consistent knowledge and to achieve safe behavior.

[3]`bugseng.com/products/ppl`
[4]`research.microsoft.com/en-us/um/redmond/projects/z3`
[5]`www.uppaal.org`

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

Table 1: Experimental results for model-checking tool Uppaal and our prototype tool.

References

[ACD+92] Rajeev Alur, Costas Courcoubetis, David L. Dill, Nicolas Halbwachs, and Howard Wong-Toi. An implementation of three algorithms for timing verification based on automata emptiness. In *RTSS*, pages 157–166, 1992.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[Alu99] Rajeev Alur. Timed automata. In *Proceedings of the 11th International Conference on Computer Aided Verification (CAV)*, LNCS, pages 8–22. Springer, 1999.

[ARB+13] Lăcrămioara Aştefănoaei, Souha Ben Rayana, Saddek Bensalem, Marius Bozga, and Jacques Combaz. Compositional invariant generation for timed systems. Technical Report TR-2013-5, Verimag Research Report, 2013.

[BBSN08] Saddek Bensalem, Marius Bozga, Joseph Sifakis, and Thanh-Hung Nguyen. Compositional verification for component-based systems and application. In *Proceedings of the 6th International Symposium on Automated Technology for Verification and Analysis*, ATVA '08, pages 64–79, Berlin, Heidelberg, 2008. Springer-Verlag.

[BGL+11] Saddek Bensalem, Andreas Griesmayer, Axel Legay, Thanh-Hung Nguyen, Joseph Sifakis, and Rongjie Yan. D-finder 2: Towards efficient correctness of incremental design. In Mihaela Gheorghiu Bobaru, Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi, editors, *NASA Formal Methods*, volume 6617 of *Lecture Notes in Computer Science*, pages 453–458. Springer, 2011.

[DLL+12] Alexandre David, Kim Guldstrand Larsen, Axel Legay, Mikael H. Møller, Ulrik Nyman, Anders P. Ravn, Arne Skou, and Andrzej Wasowski. Compositional verification of real-time systems using Ecdar. *STTT*, 2012.

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Inf. Comput.*, 111(2):193–244, June 1994.

[Jon83] Cliff B. Jones. Specification and design of (parallel) programs. pages 321–332, 1983.

[LLH+12] Shang-Wei Lin, Yang Liu, Pao-Ann Hsiung, Jun Sun, and Jin Song Dong. Automatic generation of provably correct embedded systems. In *ICFEM*, 2012.

[MC81] Jayadev Misra and Kanianthra Mani Chandy. Proofs of networks of processes. page 4:417426, 1981.

[Pnu84] Amir Pnueli. In transition from global to modular temporal reasoning about programs. page 123144, 1984.

[SBL12] Marius Bozga Saddek Bensalem, Benoit Boyer and Axel Legay. Incremental generation of linear invariants for component-based systems. Technical Report TR-2012-15, Verimag Research Report, 2012.

[YPD94] Wang Yi, Paul Pettersson, and Mats Daniels. Automatic verification of real-time communicating systems by constraint-solving. In *FORTE*, pages 243–258, 1994.