Compositional Verification for Component-based Systems and Application[*]

Saddek Bensalem Marius Bozga Thanh-Hung Nguyen Joseph Sifakis

Verimag Laboratory, Université Joseph Fourier Grenoble, CNRS.

`{bensalem, bozga, thnguyen, sifakis}@imag.fr`

March 11, 2010

Abstract

We present a compositional method for the verification of component-based systems described in a subset of the BIP language encompassing multi-party interaction without data transfer. The method is based on the use of two kinds of invariants. Component invariants are over-approximations of components' reachability sets. Interaction invariants are global constraints on the states of components involved in interactions. The method has been implemented in the D-Finder tool and has been applied for checking deadlock-freedom. The experimental results on non-trivial examples show that our method allow either to prove deadlock-freedom or to identify very few deadlock configurations that can be analyzed by using state space exploration.

1 Introduction

Compositional verification techniques are used to cope with state explosion in concurrent systems. The idea is to apply divide-and-conquer approaches to infer global properties of complex systems from properties of their components. Separate verification of components limits state explosion. Nonetheless, components mutually interact in a system and their behavior and properties are inter-related. This is a major difficulty in designing compositional techniques. As explained in [1], compositional rules are in general of the form

[*]This work has been partially supported by EU 7th Framework Programme IST STREP 215543 COMponent-Based Embedded Systems design Techniques COMBEST.

$$\frac{B_1 < \Phi_1 >,\ B_2 < \Phi_2 >,\ C(\Phi_1, \Phi_2, \Phi)}{B_1 \| B_2 < \Phi >} \tag{1}$$

That is, if two components with behaviors B_1, B_2 satisfy local properties Φ_1, Φ_2 respectively, and $C(\Phi_1, \Phi_2, \Phi)$ is some condition taking into account the semantics of parallel composition operation and relating the local properties with the global property, then the system $B_1 \| B_2$ resulting from the composition of B_1 and B_2 will satisfy a global property Φ.

We present here a different approach for compositional verification of invariants based on the following rule:

$$\frac{\{B_i < \Phi_i >\}_i,\ \Psi \in II(\|_\gamma \{B_i\}_i, \{\Phi_i\}_i),\ (\bigwedge_i \Phi_i) \wedge \Psi \Rightarrow \Phi}{\|_\gamma \{B_i\}_i < \Phi >} \tag{2}$$

This rule allows to prove invariance of Φ for systems obtained by using a n-ary parallel composition operation parameterized by a set of interactions γ. It uses global invariants which are the conjunction of local invariants of components Φ_i and an interaction invariant Ψ. The latter expresses constraints on the global state space induced by interactions between components. Interaction invariants are computed automatically from abstractions of the system to be verified.

The paper provides a method derived from the rule (2) for automated verification of component-based systems described in a subset of the BIP (Behavior-Interaction-Priority) language [2]. In BIP, atomic components are automata extended with data and functions written in C. Moreover, parallel composition is parameterized by n-ary interactions extended with data transfer and by priority rules. Here, we restrict ourselves to BIP systems where interactions are pure synchronizations, that is, without data transfer between atomic components. Also, we do not consider priority rules. The main results are the following:

- We provide heuristics for computing sequences of increasingly stronger component invariants. Component invariants are over-approximations of the set of the reachable states and are generated by simple forward static analysis. When proving invariance of a property fails, it is possible to find stronger global invariants by computing stronger component invariants and consequently, stronger interaction invariants, as shown below.

- We provide heuristics for computing interaction invariants. Interaction invariants are computed automatically from abstractions of the system. These abstractions are the composition, according to interactions, of finite state abstractions B_i^α of the components B_i with respect to their invariants Φ_i. They can be represented as a Petri net whose transitions correspond to interactions between components. Interaction invariants correspond to traps [3] of the Petri net and are computed symbolically as solutions of a set of boolean equations.

- We present an implementation and application of the method in the D-Finder tool for verification of deadlock-freedom. D-Finder takes as input BIP programs and progressively eliminates potential deadlocks by generating increasingly stronger invariants. For this, it cooperates tightly with three tools: Omega [4] for quantifier elimination occuring in computation of finite state-abstractions, CUDDs [5] for symbolic computation of interaction invariants and Yices [6] for checking satisfiability of predicates. It is also connected to the state space exploration tool of the BIP platform, for finer analysis when the heuristic fails to prove deadlock-freedom. We provide non trivial examples showing the capabilities of D-Finder as well as the efficiency of the method.

Related work

There are mainly two approaches for verifying compositionally invariance properties of infinite-state systems, respectively, assume-guarantee and deductive approaches.

The assume-guarantee approach relies on the asymmetric decomposition of properties into two parts. One is an assumption about the global behavior of the environment of the component; the other is a property guaranteed by the component when the assumption about its environment holds. This approach has been extensively studied (see for example [7, 8, 9, 10, 11, 12, 13, 14]), however, many issues make the application of assume-guarantee rules difficult. These are discussed in detail in a recent paper [15] which provides an evaluation of automated assume-guarantee techniques. The main difficulties are finding decompositions into sub-systems and choosing adequate assumptions for a particular decomposition. However, a recent work presented in [31] has shown that automated assume-guarantee reasoning can be applied to $n > 2$ components by recursively applying the learning algorithm. This approach is significantly more successful than considering only 2-way decompositions. Moreover, alphabet refinement, which extends the assumption learning process to infer assumption alphabets, can be applied to make assume-guarantee reasoning more efficient.

Our method differs from assume-guarantee methods in that it avoids combinatorial explosion of the decomposition and is directly applicable to systems with multiparty (not only binary) interactions. Furthermore, it needs only guarantees for components. It replaces the search for adequate assumptions for each component by the use of interaction invariants. These can be computed automatically from given component invariants (guarantees). Interaction invariants correspond to a "*cooperation test*" in the terminology of [16] as they allow to eliminate product states which are not feasible by the semantics of parallel composition.

Deductive methods for proving invariance properties of transition systems are based on a proof rule which can be formulated as follows. To prove that some given predicate Φ is an invariant of a given program S i.e., that every reachable state of S satisfies Φ, it is necessary and sufficient to find an auxiliary predicate Φ^{aux} with the following properties: 1) Φ^{aux} is stronger than Φ, 2) Φ^{aux} is preserved by every transition of S i.e., for every states s and s', if s satisfies Φ^{aux} and s' is reachable from s by a transition, then s' also satisfies Φ^{aux}, and 3) Φ^{aux} is satisfied by every initial state of S.

As shown e.g., in [17], this rule is sound and (relatively) complete for proving invariance properties of transition systems. It is very important to understand that the completeness result/proof of this rule does not give a clue of how to find the auxiliary predicate. Indeed, choosing the set of reachable states $\mathit{Reach}(S)$ as auxiliary predicate reduces the original problem to the checking of the first premise. Moreover, if S is a finite-state system the predicates can be expressed in propositional logic and checking the premises can be done algorithmically. However, in general, one needs an assertion language which is at least as expressive as integer arithmetic to express predicates, which makes checking the premises of the rule undecidable. Even worse, there are systems for which $\mathit{Reach}(S)$ is expressible using closed formula over integers yet computing such a representation cannot be done effectively.

The deductive rule provides only a partial answer to the verification of invariance properties. It leaves open (i) how to find the auxiliary predicate Φ^{aux} and (ii) how to prove that Φ^{aux} is preserved by every transition of S and satisfied by the initial states. Problem (ii) is related to the problem of proving tautologies of the underlying assertion language. We argue that our rule (2) can be easily seen as an instance of the proof rule of the deductive approach. But in our case we describe techniques for dealing with problem (i) as well i.e., how to automatically generate auxiliary predicates.

Organization of the paper

The paper is organized as follows. Section 2 introduces the basic definitions about BIP and invariants. The method for computing component invariants and the corresponding interaction invariants is presented in Section 3. Section 4 presents the application of the method for checking deadlock-freedom including a description of D-Finder and experimental results. Section 5 presents concluding remarks and future work.

Throughout the paper, we use the Temperature Control System from [18] as a running example to illustrate all the new introduced concepts and the verification method.

Example 1 (Temperature Control System) *[18] This system controls the coolant temperature in a reactor tank by moving two independent control rods. The goal is to maintain the coolant between the temperatures θ_m and θ_M. When the temperature reaches its maximum value θ_M, the tank must be refrigerated with one of the rods. The temperature rises at a rate v_r and decreases at rate v_d. A rod can be moved again only if T time units have elapsed since the end of its previous movement. If the temperature of the coolant cannot decrease because there is no available rod, a complete shutdown is required.*

2 Models, Invariants and their Properties

In this section, we present the basic model for the BIP framework as well as the notion of invariant.

2.1 Basic model for BIP

The BIP component framework [2] is based on a 3-tier architecture, the layers being *behavior*, *interaction* and *priority*, where:

1. *Behavior* describes the dynamic behavior of atomic components. It consists of a set of extended transition systems. Each transition has a port, a guard and a function. Guards are conditions depending on local state. Ports characterize the component's ability to interact with a given environment.

2. *Interactions* describe architectural constraints on behavior. They define joint state changes of composed components used to coordinate their execution.

3. *Priorities* provide a mechanism for restricting the global behavior of the layers underneath by filtering amongst possible interactions. They help reducing non-determinism in the execution

of the interactions between the components. They are useful for enforcing state invariant properties and/or scheduling policies.

BIP provides mechanisms for composition of *behavior* using *interaction* and *priority* glues. This framework has been implemented in a language and a toolset. The BIP language leverages on C style variables and data type declarations, expressions and statements, and provides additional syntactic constructs for defining component behavior, specifying the interactions and the priorities. The BIP toolset includes an editor and a compiler for generating from BIP programs C++ code executable on a dedicated middleware.

We provide hereafter a formalization of atomic components in BIP and their composition by using interactions. Priorities are not considered.

Definition 1 (Atomic Component) *An atomic component is a transition system extended with data* $B = (L, P, \mathcal{T}, X, \{g_\tau\}_{\tau\in\mathcal{T}}, \{f_\tau\}_{\tau\in\mathcal{T}})$*, where:*

- $(L, P, \mathcal{T})$ *is a transition system, that is*
  - $L = \{l_1, l_2, \dots, l_k\}$ *is a set of control locations,*
  - P *is a set of ports,*
  - $\mathcal{T} \subseteq L \times P \times L$ *is a set of transitions,*
- $X = \{x_1, \dots, x_n\}$ *is a set of variables and for each* $\tau \in \mathcal{T}$ *respectively,* g_τ *is a guard, a predicate on* X*, and* $f_\tau(X, X')$ *is an update relation, a predicate on* X *(current) and* X' *(next) state variables.*

Example 2 *We provide in Figure 1 a discretized model of the Temperature Control System in BIP, decomposed into three atomic components: a Controller and two components Rod1, Rod2 modeling the rods. We take* $\theta_m = 100°$, $\theta_M = 1000°$, $T = 3600$ *seconds. Furthermore, we assume that* $v_r = 1°/s$ *and* $v_d = 2°/s$*. The Controller has two control locations* $\{l_5, l_6\}$*, a variable* θ*, three ports* $\{tick, cool, heat\}$ *and four transitions: 2 loop transitions labeled by tick which increase or decrease the temperature as time progresses and 2 transitions triggering moves of the rods. The components Rod1 and Rod2 are identical, up to the renaming of states and ports. Each one has two control locations and four transitions: two loop transitions labeled by tick and two transitions synchronized with transitions of the Controller.*

The following definition provides the operational semantics of atomic behavior in terms of plain labeled transition systems (LTS). States of LTS correspond to configurations, that are pairs of control locations and valuations on variables. Transitions of LTS are obtained from transitions of the atomic behavior according to the guards and update relations specified on them.

Definition 2 (Semantics of extended transition system) *The semantics of* B=(L,P, $\mathcal{T}$, X, $\{g_\tau\}_{\tau\in\mathcal{T}}$, $\{f_\tau\}_{\tau\in\mathcal{T}}$), *is a transition system* $(Q, P, \mathcal{T}_0)$ *such that*

- $Q = L \times \mathbf{X}$ *where* $\mathbf{X}$ *denotes the set of valuations of variables* X.
- $\mathcal{T}_0$ *is the set including transitions* $((l, \mathbf{x}), p, (l', \mathbf{x}'))$ *such that* $g_\tau(\mathbf{x}) \wedge f_\tau(\mathbf{x}, \mathbf{x}')$ *for some* $\tau = (l, p, l') \in \mathcal{T}$. *As usual, if* $((l, \mathbf{x}), p, (l', \mathbf{x}')) \in \mathcal{T}_0$ *we write* $(l, \mathbf{x}) \xrightarrow{p} (l', \mathbf{x}')$.

Let us introduce few useful notations. Given a transition $\tau = (l, p, l') \in \mathcal{T}$, l and l' are respectively, the *source* and the *target* location denoted respectively by ${}^\bullet\tau$ and $\tau^\bullet$. For a location l, we use the predicate *at_l* which is *true* iff the system is at location l. A state predicate Φ is a boolean expression involving location predicates and predicates on X. Any state predicate can be put in the form $\bigvee_{l\in L} at\_l \wedge \varphi_l$. Notice that predicates on locations are disjoint and their disjunction is true.

We define below parallel composition for components parameterized by a set of interactions. For the sake of clarity, we consider only pure synchronizations, that is flat interactions without data transfer between components.

Definition 3 (Interactions) *Given a set of components* $B_1, B_2, \ldots, B_n$, *where* $B_i = (L_i$,P_i, $\mathcal{T}_i$, X_i, $\{g_\tau\}_{\tau\in\mathcal{T}_i}$, $\{f_\tau\}_{\tau\in\mathcal{T}_i})$, *an interaction* a *is a set of ports, subset of* $\bigcup_{i=1}^n P_i$, *such that* $\forall i = 1, \ldots, n$ $|a \cap P_i| \leq 1$.

The BIP framework allows to define rich interaction models by using hierarchical interactions extended with data transfer as presented in [19, 20]. In this work, we restrict to pure synchronizations. The absence of hierarchy is not a real limitation, as long as hierarchical interaction models can be statically transformed into equivalent flat interaction models with a potentially increased number of interactions [21]. Nevertheless, the absence of data transfer between components is a severe limitation in practice, but fortunately, it allows easier decomposition and compositional reasoning on the system, as shown later in section 3.

Definition 4 (Parallel Composition) *Given* n *components* B_i=(L_i,P_i,$\mathcal{T}_i$, X_i, $\{g_\tau\}_{\tau\in\mathcal{T}_i}$,$\{f_\tau\}_{\tau\in\mathcal{T}_i}$) *and a set of interactions* γ, *we define* $B = \gamma(B_1, \ldots, B_n)$ *as the component* (L,γ, $\mathcal{T}$, X, $\{g_\tau\}_{\tau\in\mathcal{T}}$, $\{f_\tau\}_{\tau\in\mathcal{T}}$), *where:*

- $(L, \gamma, \mathcal{T})$ *is the transition system such that*
  - $L = L_1 \times L_2 \times \ldots \times L_n$ *is the set of control locations,*
  - $\mathcal{T} \subseteq L \times \gamma \times L$ *contains transitions* $\tau = ((l_1, \ldots, l_n), a, (l'_1, \ldots, l'_n))$ *obtained by synchronization of sets of transitions* $\{\tau_i = (l_i, p_i, l'_i) \in \mathcal{T}_i\}_{i \in I}$ *such that* $\{p_i\}_{i \in I} = a \in \gamma$ *and* $l'_j = l_j$ *if* $j \notin I$*, for arbitrary* $I \subseteq \{1, ..., n\}$
- $X = \bigcup_{i=1}^{n} X_i$ *and for a transition* τ *resulting from the synchronization of a set of transitions* $\{\tau_i\}_{i \in I}$*, the associated guard and function are respectively* $g_\tau = \bigwedge_{i \in I} g_{\tau_i}$ *and* $f_\tau = \bigwedge_{i \in I} f_{\tau_i} \wedge \bigwedge_{i \notin I}(X'_i = X_i)$.

Finally, we consider systems defined as parallel composition of components together with an initial condition.

Definition 5 (System) *A system* $\mathcal{S}$ *is a pair* $\langle B, Init \rangle$ *where* B *is a component and* $Init$ *is a state predicate characterizing the initial states of* B.

Example 3 *The three atomic components Controller, Rod1 and Rod2 are composed by using the following set of interactions, indicated by connectors in Figure 1 :* $\{tick, tick_1, tick_2\}$, $\{cool, cool_1\}$, $\{cool, cool_2\}$, $\{heat, rest_1\}$, $\{heat, rest_2\}$. *In this model, complete shutdown corresponds to a deadlock. Throughout the paper we verify deadlock-freedom of this example by taking* $Init = at\_l_5 \wedge (\theta = 100) \wedge at\_l_1 \wedge (t_1 = 3600) \wedge at\_l_3 \wedge (t_2 = 3600)$.

2.2 Invariants and Their Properties

For a component $B = (L, P, \mathcal{T}, X, \{g_\tau\}_{\tau \in \mathcal{T}}, \{f_\tau\}_{\tau \in \mathcal{T}})$, we recall here the definition of the *post* predicate transformer allowing to compute successors of global states represented symbolically by state predicates. Given a state predicate $\Phi = \bigvee_{l \in L} at\_l \wedge \varphi_l$, we define $post(\Phi) = \bigvee_{l \in L}(\bigvee_{\tau=(l,p,l')} at\_l' \wedge post_\tau(\varphi_l))$ where $post_\tau(\varphi)(X) = \exists X'.g_\tau(X') \wedge f_\tau(X', X) \wedge \varphi(X')$. Equivalently, we have that $post(\Phi) = \bigvee_{l \in L} at\_l \wedge (\bigvee_{\tau=(l',p,l)} post_\tau(\varphi_{l'}))$. That is $post(\Phi)$ can be computed by forward propagation of the assertions associated with control locations in Φ.

We define in a similar way, the pre_τ predicate transformer for a transition τ, $pre_\tau(\varphi)(X) = \exists X'.g_\tau(X) \wedge f_\tau(X, X') \wedge \varphi(X')$.

Definition 6 (Invariants) *Given a system* $\langle B, Init \rangle$ *a state predicate* Φ *is*

- *an inductive invariant iff* $(Init \vee post(\Phi)) \Rightarrow \Phi$.

- *an invariant iff there exists an inductive invariant Φ^{aux} such that $\Phi^{aux} \Rightarrow \Phi$.*

Example 4 *For the Temperature Control System of Figure 1, the predicate $\Phi_1 = (at\_l_1 \wedge t_1 \geq 0) \vee (at\_l_2 \wedge t_1 \geq 3600)$ is an inductive invariant of the Rod1 component given the initial condition $Init_1 = at\_l_1 \wedge t_1 = 3600$. Also, the predicate $\Phi_3 = (at\_l_5 \wedge 100 \leq \theta \leq 1000) \vee (at\_l_6 \wedge 100 \leq \theta \leq 1000)$ is an non-inductive invariant of the Controller component, given the initial condition $Init_3 = at\_l_5 \wedge \theta = 100$. An auxiliary inductive invariant that implies Φ_3 is $\Phi_3^{aux} = (at\_l_5 \wedge 100 \leq \theta \leq 1000) \vee (at\_l_6 \wedge 100 \leq \theta \leq 1000 \wedge (\theta \text{ is even}))$.*

Notice that invariants are over-approximations of the set of the reachable states from $Init$. We extensively use the following well-known results about invariants.

Proposition 1 *Let Φ_1, Φ_2 be two invariants of a component B. Then $\Phi_1 \wedge \Phi_2$, $\Phi_1 \vee \Phi_2$ are invariants of B.*

3 The method

We consider a system $\gamma(B_1, \ldots, B_n)$ obtained by composing a set of atomic components $B_1, ..., B_n$ by using a set of interactions γ. To prove a global invariant Φ for $\gamma(B_1, \ldots, B_n)$, we recall the rule presented in the introduction:

$$\frac{\{B_i < \Phi_i >\}_i^n,\ \Psi \in II(\gamma(B_1, \ldots, B_n), \{\Phi_i\}_i^n),\ (\bigwedge_i^n \Phi_i) \wedge \Psi \Rightarrow \Phi}{\gamma(B_1, \ldots, B_n) < \Phi >}$$

where $B_i < \Phi_i >$ means that Φ_i is an invariant of component B_i and Ψ belongs to the set II of interaction invariants of $\gamma(B_1, \ldots, B_n)$ computed automatically from Φ_i and $\gamma(B_1, \ldots, B_n)$. The rule is illustrated on Figure 2 for a system with two components, invariants Φ_1 and Φ_2 and interaction invariant Ψ.

We provide below methods for computing component invariants by static analysis. We also provide a general method for computing interaction invariants for $\gamma(B_1, \ldots, B_n)$, given a set of component invariants Φ_i.

3.1 Computing Component Invariants

For the application of rule (2) we need to compute component invariants Φ_i. We can take $\Phi_i = Reach(B_i)$, the set of reachable state of B_i, or any upper approximation Φ_i such that $Reach(B_i) \Rightarrow$

Φ_i.

We present below a lightweight method for the computation of sequences of increasingly stronger inductive invariants for atomic components. The method is implemented in the D-Finder tool.

Proposition 2 *Given a system $\mathcal{S} = \langle B, Init \rangle$, the following iteration defines a sequence of increasingly stronger inductive invariants:*

$$\Phi_0 = true \quad \Phi_{i+1} = Init \vee post(\Phi_i)$$

Proof. By induction. Φ_0 is an inductive invariant. If Φ_i is an inductive invariant then $Init \vee post(\Phi_i) \Rightarrow \Phi_i$. As *post* is monotonic and distributes over disjunction, $post(\Phi_{i+1}) = post(Init \vee post(\Phi_i)) \Rightarrow post(\Phi_i) \Rightarrow \Phi_{i+1}$. Moreover, $Init \Rightarrow \Phi_{i+1}$. So Φ_{i+1} is an inductive invariant. $\Box$

We use different strategies for producing such invariants. We usually iterate until we find deadlock-free invariants. Their use guarantees that global deadlocks are exclusively due to synchronization.

A key issue is efficient computation of such component invariants as the precise computation of *post* requires quantifier elimination. An alternative to quantifier elimination is to compute over-approximations of *post* based on syntactic analysis of the predicates. In this case, the obtained invariants may not be inductive.

We provide a brief description of a syntactic technique used for approximating $post_\tau$ for a fixed transition τ. A more detailed presentation, as well as other techniques for generating component invariants are given in [22].

Consider a transition $\tau = (l, p, l')$ of $B = (L, P, \mathcal{T}, X, \{g_\tau\}_{\tau\in\mathcal{T}}, \{f_\tau\}_{\tau\in\mathcal{T}})$. Assume that its guard is of the form $g_\tau(Y)$ and the associated update function f_τ is of the form $Z_1' = e_\tau(U) \wedge Z_2' = Z_2$ where $Y, Z_1, Z_2, U \subseteq X$ and $\{Z_1, Z_2\}$ is a partition of X.

For an arbitrary predicate φ find a decomposition $\varphi = \varphi_1(Y_1) \wedge \varphi_2(Y_2)$ such that $Y_2 \cap Z_1 = \emptyset$ i.e., which has a conjunct not affected by the update function f_τ. We apply the following rule to compute over-approximations $post^a_\tau(\varphi)$ of $post_\tau(\varphi)$:

$$post^a_\tau(\varphi) = \varphi_2(Y_2) \wedge \left\{ \begin{array}{ll} g_\tau(Y) & \text{if } Z_1 \cap Y = \emptyset \\ true & \text{otherwise} \end{array} \right\} \wedge \left\{ \begin{array}{ll} Z_1 = e_\tau(U) & \text{if } Z_1 \cap U = \emptyset \\ true & \text{otherwise} \end{array} \right\}$$

Proposition 3 *If τ and φ are respectively a transition and a state predicate as above, then $post_\tau(\varphi) \Rightarrow post^a_\tau(\varphi)$.*

Proof. We can over-approximate successively $post_\tau(\varphi)$ as follows:

$$\begin{array}{rcl}
post_\tau(\varphi)(X') & = & \exists X.\,(\varphi(X) \wedge g_\tau(X) \wedge f_\tau(X,X')) \\
& = & \exists Z_1, Z_2.\,(\varphi_1(Y_1) \wedge \varphi_2(Y_2) \wedge g_\tau(Y) \wedge Z_1' = e_\tau(U) \wedge Z_2' = Z_2) \\
& \Rightarrow & \exists Z_2.\,(\varphi_2(Y_2) \wedge Z_2' = Z_2) \bigwedge \exists Z_1, Z_2.\,(g_\tau(Y) \wedge Z_1' = e_\tau(U) \wedge Z_2' = Z_2) \\
& = & \varphi_2(Y_2') \bigwedge \exists Z_1, Z_2.\,(g_\tau(Y) \wedge Z_1' = e_\tau(U) \wedge Z_2' = Z_2) \\
& \Rightarrow & \varphi_2(Y_2') \bigwedge \left\{ \begin{array}{ll} g_\tau(Y') & \text{if } Z_1 \cap Y = \emptyset \\ true & \text{otherwise} \end{array} \right\} \bigwedge \left\{ \begin{array}{ll} Z_1' = e_\tau(U') & \text{if } Z_1 \cap U = \emptyset \\ true & \text{otherwise} \end{array} \right\} \\
& = & post^a_\tau(\varphi)(X')
\end{array}$$

□

Example 5 *For the Temperature Control System of Figure 1, the predicates $\Phi_1 = (at\_l_1 \wedge t_1 \geq 0) \vee (at\_l_2 \wedge t_1 \geq 3600)$, $\Phi_2 = (at\_l_3 \wedge t_2 \geq 0) \vee (at\_l_4 \wedge t_2 \geq 3600)$ and $\Phi_3 = (at\_l_5 \wedge 100 \leq \theta \leq 1000) \vee (at\_l_6 \wedge 100 \leq \theta \leq 1000)$ are respectively invariants of the atomic components Rod1, Rod2 and Controller.*

3.2 Computing Interaction Invariants

For the sake of clarity, we first show how to compute interaction invariants for a system $\gamma(B_1, \ldots, B_n)$ without variables, that is, where the atomic components B_i are finite transition systems. Then, we show how to deal with infinite state systems.

3.2.1 For finite state systems

Definition 7 (Forward Interaction Sets) *Given a system $\gamma(B_1, \ldots, B_n)$ where $B_i = (L_i, P_i, \mathcal{T}_i)$ are transition systems, we define for a set of locations $L \subseteq \bigcup_{i=1}^n L_i$ its forward interaction set $L^\bullet = \bigcup_{l \in L} l^\bullet$ where $l^\bullet = \big\{\{\tau_i\}_{i \in I} \mid \forall i.\tau_i \in \mathcal{T}_i \;\wedge\; \exists i.{}^\bullet\tau_i = l \;\wedge\; \{port(\tau_i)\}_{i \in I} \in \gamma\big\}$.*

That is, $l^\bullet$ consists of sets of component transitions involved in some interaction of γ in which a transition τ_i issued from l can participate (see Figure 3). We define in a similar manner, for a set of locations its backward interaction set ${}^\bullet L = \bigcup_{l \in L} {}^\bullet l$ where ${}^\bullet l = \big\{\{\tau_i\}_{i \in I} \mid \forall i.\tau_i \in \mathcal{T}_i \;\wedge\; \exists i.\tau_i^\bullet =$

$l \wedge \{port(\tau_i)\}_{i \in I} \in \gamma\big\}$. The elements of ${}^\bullet l$ and $l^\bullet$ can be also viewed as transitions of a Petri net, which correspond to interactions of γ. As for Petri nets, we define the notion of trap.

Definition 8 (Traps) *Given a parallel composition $\gamma(B_1, \ldots, B_n)$ where $B_i = (L_i, P_i, \mathcal{T}_i)$, a trap is a set L of locations $L \subseteq \bigcup_{i=1}^n L_i$ such that $L^\bullet \subseteq {}^\bullet L$.*

The following proposition expresses a characteristic property of traps: if the initial state of $\gamma(B_1, \ldots, B_n)$ has some control location belonging to a trap then all its successor states have some control location belonging to the trap.

Proposition 4 *Given a system $\mathcal{S} = \langle \gamma(B_1, \ldots, B_n), Init \rangle$, if the set of locations $L \subseteq \bigcup_{i=1}^n L_i$ is a trap containing an initial state of some component then $\bigvee_{l \in L} at\_l$ is an invariant of $\mathcal{S}$.*

Proof. The behavior B obtained by a composition of atomic components without data is equally represented by a 1-safe Petri net where places correspond to control states of $\bigcup_{i=1}^n L_i$ and transitions correspond to interactions of γ. Moreover, the traps previously introduced correspond precisely to traps in this Petri net. Now, concerning traps in Petri nets, the following invariance property holds: *if a trap is initially marked, it remains marked through all computation of the net* (see [3] for details). This property is simply lifted to BIP in order to obtain synchronization invariants.□

The following result given in [23] characterizes traps as solution of a system of implications.

Proposition 5 *Let $\gamma(B_1, ..., B_n)$ be a system and let $\mathbf{v} : \bigcup_{i=1}^n L_i \to \mathbb{B}$ be a boolean valuation. If $\mathbf{v}$ satisfies the following set of the implications:*

$$\mathbf{v}(l) \;\Rightarrow\; \bigwedge_{\{\tau_i\}_{i \in I} \in l^\bullet} \left(\bigvee_{l' \in \{\tau_i^\bullet\}_{i \in I}} \mathbf{v}(l') \right) \quad \textit{for } l \in \bigcup_{i=1}^n L_i$$

then the set $\{l \in \bigcup_{i=1}^n L_i \mid \mathbf{v}(l) = true\}$ is a trap.

This characterization allows us to compute the set of traps using different approaches. We experiment two of them. The first approach uses the SAT-solver Yices [6] to progressively obtain minimal solutions of the above system. Nevertheless, as shown in [24, 25], computing the set of minimal traps is a NP-complete problem and in practice, an iterative trap extraction process may be long and non-exhaustive. The second approach uses symbolic BDD-based representation [5] to obtain the set of all traps at once, as solution of the boolean system of implications above. This method has the advantage of computing all the traps, however, it suffers from the potential explosion of its symbolic BDD representation, which is very difficult to predict and control in practice.

Example 6 *For the abstraction of the Temperature Control System given in Figure 4 the following set of implications characterizes the set of traps:*

$$
\begin{aligned}
\phi_{11} \Rightarrow\ & (\phi_{12} \vee \phi_{42} \vee \phi_{52}) \wedge \\
& (\phi_{12} \vee \phi_{42} \vee \phi_{62}) \wedge \\
& (\phi_{12} \vee \phi_{32} \vee \phi_{52}) \wedge \\
& (\phi_{12} \vee \phi_{32} \vee \phi_{62}) \wedge \\
& (\phi_{12} \vee \phi_{41} \vee \phi_{52}) \wedge \\
& (\phi_{12} \vee \phi_{41} \vee \phi_{62}) \\
\phi_{12} \Rightarrow\ & (\phi_{61} \vee \phi_{21}) \\
\phi_{21} \Rightarrow\ & (\phi_{51} \vee \phi_{11}) \\
\phi_{22} \Rightarrow\ & (\phi_{51} \vee \phi_{11}) \\
\phi_{31} \Rightarrow\ & (\phi_{22} \vee \phi_{32} \vee \phi_{52}) \wedge \\
& (\phi_{22} \vee \phi_{32} \vee \phi_{62}) \wedge \\
& (\phi_{12} \vee \phi_{32} \vee \phi_{52}) \wedge \\
& (\phi_{12} \vee \phi_{32} \vee \phi_{62}) \wedge \\
& (\phi_{21} \vee \phi_{32} \vee \phi_{52}) \wedge \\
& (\phi_{21} \vee \phi_{32} \vee \phi_{62}) \\
\phi_{32} \Rightarrow\ & (\phi_{61} \vee \phi_{41}) \\
\phi_{41} \Rightarrow\ & (\phi_{51} \vee \phi_{31}) \\
\phi_{42} \Rightarrow\ & (\phi_{51} \vee \phi_{31}) \\
\phi_{51} \Rightarrow\ & (\phi_{22} \vee \phi_{42} \vee \phi_{52}) \wedge \\
& (\phi_{22} \vee \phi_{32} \vee \phi_{52}) \wedge \\
& (\phi_{12} \vee \phi_{42} \vee \phi_{52}) \wedge \\
& (\phi_{12} \vee \phi_{32} \vee \phi_{52}) \wedge \\
& (\phi_{22} \vee \phi_{41} \vee \phi_{52}) \wedge \\
& (\phi_{21} \vee \phi_{42} \vee \phi_{52}) \wedge \\
& (\phi_{21} \vee \phi_{41} \vee \phi_{52}) \wedge \\
& (\phi_{21} \vee \phi_{32} \vee \phi_{52}) \wedge \\
& (\phi_{12} \vee \phi_{41} \vee \phi_{52}) \\
\phi_{52} \Rightarrow\ & (\phi_{61} \vee \phi_{21}) \wedge \\
& (\phi_{61} \vee \phi_{41}) \\
\phi_{61} \Rightarrow\ & (\phi_{22} \vee \phi_{42} \vee \phi_{62}) \wedge \\
& (\phi_{22} \vee \phi_{32} \vee \phi_{62}) \wedge \\
& (\phi_{12} \vee \phi_{42} \vee \phi_{62}) \wedge \\
& (\phi_{12} \vee \phi_{32} \vee \phi_{62}) \wedge \\
& (\phi_{22} \vee \phi_{41} \vee \phi_{62}) \wedge \\
& (\phi_{21} \vee \phi_{42} \vee \phi_{62}) \wedge \\
& (\phi_{21} \vee \phi_{41} \vee \phi_{62}) \wedge \\
& (\phi_{21} \vee \phi_{32} \vee \phi_{62}) \wedge \\
& (\phi_{12} \vee \phi_{41} \vee \phi_{62}) \\
\phi_{62} \Rightarrow\ & (\phi_{51} \vee \phi_{11}) \wedge \\
& (\phi_{51} \vee \phi_{31})
\end{aligned}
$$

The set of minimal traps for the example given in Figure 4 are:
$L_1 = \{\phi_{21}, \phi_{41}, \phi_{51}, \phi_{52}\}$, $L_2 = \{\phi_{11}, \phi_{12}, \phi_{21}, \phi_{31}, \phi_{32}, \phi_{41}\}$, $L_3 = \{\phi_{32}, \phi_{41}, \phi_{42}, \phi_{51}\}$, $L_4 = \{\phi_{11}, \phi_{12}, \phi_{31}, \phi_{32}, \phi_{61}, \phi_{62}\}$ *and* $L_5 = \{\phi_{12}, \phi_{21}, \phi_{22}, \phi_{51}\}$.

Traps define a particularly interesting category of structural invariants for Petri nets. They have been extensively studied in the literature. Used jointly with locks (or siphons) they allow to establish (statically) properties such as liveness and deadlock-freedom for some classes of Petri nets [26, 27].

3.2.2 For infinite state systems

We have shown how to compute interaction invariants from traps relating control locations of finite state components. To compute interaction invariants for infinite state systems, we first compute compositionally a finite state abstraction of the composite system. Interaction invariants are concretizations of the traps of the abstract system.

Consider a system $\mathcal{S} = \langle \gamma(B_1, \ldots, B_n), Init \rangle$ and a set of component invariants $\Phi_1 \ldots \Phi_n$ associated with the atomic components. We show below, for each component B_i and its associated invariant Φ_i, how to define a finite state abstraction α_i and to compute an abstract transition system $B_i^{\alpha_i}$.

Definition 9 (Abstraction Function) *Let Φ be an invariant of a system $\langle B, Init \rangle$ written in disjunctive form $\Phi = \bigvee_{l \in L} at\_l \wedge (\bigvee_{m \in M_l} \varphi_{lm})$ such that the atomic predicates $at\_l \wedge \varphi_{lm}$ are disjoint. Given Φ, an abstraction function α is an injective function associating with each atomic predicate $at\_l \wedge \varphi_{lm}$ a symbol $\phi = \alpha(at\_l \wedge \varphi_{lm})$ called abstract state. We denote by Φ^α the set of the abstract states.*

Definition 10 (Abstract System) *Given a system $\mathcal{S} = \langle B, Init \rangle$, an invariant Φ and an associated abstraction function α, we define the abstract system $\mathcal{S}^\alpha = \langle B^\alpha, Init^\alpha \rangle$ where*

- *$B^\alpha = (\Phi^\alpha, P, \rightsquigarrow)$ is a transition system with $\rightsquigarrow$ such that for any pair of abstract states $\phi = \alpha(at\_l \wedge \varphi)$ and $\phi' = \alpha(at\_l' \wedge \varphi')$ we have $\phi \overset{p}{\rightsquigarrow} \phi'$ iff $\exists \tau = (l, p, l') \in \mathcal{T}$ and $\varphi \wedge pre_\tau(\varphi') \neq false$,*
- *$Init^\alpha = \bigvee_{\phi \in \Phi_0^\alpha} at\_\phi$ where $\Phi_0^\alpha = \{\phi \in \Phi^\alpha \mid \alpha^{-1}(\phi) \wedge Init \neq false\}$ is the set of the initial abstract states.*

We apply the method presented in [28] and implemented in the InVeSt tool [29] in order to compute an abstract transition system B^α for a component B. The method proceeds by elimination, starting from the universal relation on abstract states. We eliminate pairs of abstract states in a conservative way. To check whether $\phi \overset{p}{\rightsquigarrow} \phi'$, where $\phi = \alpha(at\_l \wedge \varphi)$ and $\phi' = \alpha(at\_l' \wedge \varphi')$, can be eliminated, we check that for all concrete transitions $\tau = (l, p, l')$ we have $\varphi \wedge pre_\tau(\varphi') = false$.

Example 7 *The table below provides the abstract states constructed from the components invariants Φ_1, Φ_2, Φ_3 of respectively Rod1, Rod2, Controller given in example 5.*

$\phi_{11} = at\_l_1 \wedge t_1 = 0$	$\phi_{51} = at\_l_5 \wedge \theta = 100$	$\phi_{31} = at\_l_3 \wedge t_2 = 0$
$\phi_{12} = at\_l_1 \wedge t_1 \geq 1$	$\phi_{52} = at\_l_5 \wedge 101 \leq \theta \leq 1000$	$\phi_{32} = at\_l_3 \wedge t_2 \geq 1$
$\phi_{21} = at\_l_2 \wedge t_1 \geq 3600$	$\phi_{61} = at\_l_6 \wedge \theta = 1000$	$\phi_{41} = at\_l_4 \wedge t_2 \geq 3600$
$\phi_{22} = at\_l_2 \wedge t_1 < 3600$	$\phi_{62} = at\_l_6 \wedge 100 \leq \theta \leq 998$	$\phi_{42} = at\_l_4 \wedge t_2 < 3600$

Figure 4 presents the computed abstraction of the Temperature Control System with respect to the considered invariants.$\Box$

By combining well-known results about abstractions, we compute interaction invariants of $\langle \gamma(B_1, ..., B_n)$, $Init \rangle$ from interaction invariants of $\langle \gamma(B_1^{\alpha}, \ldots, B_n^{\alpha}), Init^{\alpha} \rangle$.

The following proposition says that $\gamma(B_1^{\alpha_1}, \ldots, B_n^{\alpha_n})$ is an abstraction of $B = \gamma(B_1, ..., B_n)$

Proposition 6 *If $B_i^{\alpha_i}$ is an abstraction of B_i with respect to an invariant Φ_i and its abstraction function α_i for $i = 1, ..., n$, then $B^{\alpha} = \gamma(B_1^{\alpha_1}, \ldots, B_n^{\alpha_n})$ is an abstraction of $B = \gamma(B_1, ..., B_n)$ with respect to $\bigwedge_{i=1}^{n} \Phi_i$ and an abstraction function α obtained as the composition of the α_i.*

The following proposition says that invariants of the abstract system are also invariants of the concrete system.

Proposition 7 *If B^{α} is an abstraction of B with respect to an invariant Φ and α its abstraction function, then B^{α} simulates B. Moreover, if Φ^{α} is an invariant of $\langle B^{\alpha}, Init^{\alpha} \rangle$ then $\alpha^{-1}(\Phi^{\alpha})$ is an invariant of $\langle B, Init \rangle$.*

Proof. We show that the relation $(l, \mathbf{x}) R \phi$ is a simulation if $\alpha^{-1}(\phi) = at\_l \wedge \varphi$ and $\varphi(\mathbf{x})$ for the valuation $\mathbf{x}$. If $(l, \mathbf{x}) \xrightarrow{p} (l', \mathbf{x}')$ is a transition of B and $(l, \mathbf{x}) R \phi$ for some abstract state ϕ, then we show that there exists $\phi' = \alpha(at\_l' \wedge \varphi')$ such that $\phi \overset{p}{\rightsquigarrow} \phi'$. As Φ is an invariant of B, if $(l', \mathbf{x}')$ is reachable then $\exists \varphi' \ at\_l' \wedge \varphi' \Rightarrow \Phi$ such that $\varphi'(\mathbf{x}')$ and $\phi' = \alpha(at\_l' \wedge \varphi')$. Moreover, as $\varphi(\mathbf{x}) \wedge \varphi'(\mathbf{x}')$, we have $\varphi(\mathbf{x}) \wedge pre_{\tau}(\varphi)(\mathbf{x}) \neq false$ for $\tau = (l, p, l')$ and therefore $\phi \overset{p}{\rightsquigarrow} \phi'$. $\square$

Thus, it is possible to compute from traps which are interaction invariants of the abstract system, interaction invariants for the concrete system $B = \gamma(B_1, ..., B_n)$.

3.3 Wrap up

We give a sketch of a semi-algorithm allowing to prove invariance of Φ by iterative application of the rule (2). The semi-algorithm takes a system $\langle \gamma(B_1, \ldots, B_n), Init \rangle$ and a predicate Φ. It iteratively computes invariants of the form $\mathcal{X} = \Psi \wedge (\bigwedge_{i=1}^{n} \Phi_i)$ where Ψ is an interaction invariant and Φ_i an invariant of component B_i. If $\mathcal{X}$ is not strong enough for proving that Φ is an invariant ($\mathcal{X} \wedge \neg\Phi = false$) then either a new iteration with stronger Φ_i is started or we stop. In this case, we cannot conclude about invariance of Φ.

Input:	$\mathcal{S} = \langle \gamma(B_1, \ldots, B_n), Init \rangle$, Φ
Initially:	$\Phi_i = true$ for each $i = 1, \ldots, n$
Output:	True or inconclusive.

1. For each B_i, compute a component invariant Φ'_i; $\Phi_i := \Phi_i \wedge \Phi'_i$
2. For each B_i and Φ_i compute the corresponding abstraction $B_i^{\alpha_i}$.
3. For $\gamma(B_1^{\alpha_1}, ..., B_n^{\alpha_n})$, compute traps $L_1, L_2, \ldots, L_m$

   containing some abstract initial state.
4. For each trap L_k, compute the interaction invariant $\Psi_k = \bigvee_{\phi \in L_k} \alpha^{-1}(\phi)$;

   $\Psi := \bigwedge_{k=1}^{m} \Psi_k$.
5. If $\neg\Phi \wedge \Psi \wedge (\bigwedge_{i=1}^{n} \Phi_i) = \text{false}$ then Φ is an invariant else goto 1 or stop.

We can show by application of the following proposition that the iteration process gives progressively stronger invariants, in particular that for stronger component invariants we get stronger interaction invariants.

Proposition 8 *Let $\langle B, Init\rangle$ be a system and Φ, Φ' two non empty invariants such that $\Phi \Rightarrow \Phi'$. If α and α' are the abstraction functions corresponding to Φ and Φ' respectively, then $B^{\alpha'}$ simulates B^{α}.*

For two successive component invariants Φ_i and Φ'_i of B_i, we have $\Phi_i \Rightarrow \Phi'_i$. From proposition 8 we deduce that $B_i^{\alpha'_i}$ simulates $B_i^{\alpha_i}$ where α_i and α'_i are the abstraction functions corresponding to Φ_i and Φ'_i. As the simulation relation is preserved by parallel composition, we have $\gamma(B_1^{\alpha'_1}, ..., B_n^{\alpha'_n})$ simulates $\gamma(B_1^{\alpha_1}, ..., B_n^{\alpha_n})$. It can be shown that for each trap L' of $\gamma(B_1^{\alpha'_1}, ..., B_n^{\alpha'_n})$ there exists a trap L of $\gamma(B_1^{\alpha_1}, ..., B_n^{\alpha_n})$ such that $L \subseteq L'$. From this we infer that for each interaction invariant of $\gamma(B_1^{\alpha'_1}, ..., B_n^{\alpha'_n})$ there exists a stronger interaction invariant of $\gamma(B_1^{\alpha_1}, ..., B_n^{\alpha_n})$.

4 Application for Checking Deadlock-Freedom

We present an application of the method for checking deadlock-freedom.

Definition 11 (Deadlock States) *We define the predicate DIS characterizing the set of the states of $\gamma(B_1, \ldots, B_n)$ from which all interactions are disabled:*

$$DIS = \bigwedge_{a \in \gamma} \neg en(a) \quad where \quad en(a) = \bigvee_{port(\mathcal{T}') = a} \bigwedge_{\tau \in \mathcal{T}'} en(\tau)$$

$port(\mathcal{T}')$ for a set of transitions $\mathcal{T}' \subseteq \mathcal{T}$ is the set of ports labeling these transitions. That is, $en(a)$ characterizes all the states from which interaction a can be executed.

Example 8 *For the Temperature Control System (see Figure 1), we have:*

$$\begin{aligned} DIS \;=\;& (\neg(at\_l_5 \wedge \theta < 1000)) \bigwedge (\neg(at\_l_6 \wedge \theta = 100) \vee \neg at\_l_2) \\ \bigwedge\;& (\neg(at\_l_6 \wedge \theta > 100)) \bigwedge (\neg(at\_l_5 \wedge \theta = 1000) \vee \neg(at\_l_3 \wedge t_2 \geq 3600)) \\ \bigwedge\;& (\neg(at\_l_5 \wedge \theta = 1000) \vee \neg(at\_l_1 \wedge t_1 \geq 3600)) \bigwedge (\neg(at\_l_6 \wedge \theta = 100) \vee \neg at\_l_4) \end{aligned}$$

The system $\langle \gamma(B_1, \ldots, B_n), Init \rangle$ is deadlock-free if the predicate $\neg DIS$ is an invariant of the system. To check that $\neg DIS$ is an invariant, we need a stronger invariant Φ such that $\Phi \Rightarrow \neg DIS$ or equivalently $\Phi \wedge DIS = false$. We present below the verification heuristic for a system $\langle \gamma(B_1, \ldots, B_n), Init \rangle$ applied by the D-Finder toolset.

```
Input:    $\mathcal{S} = \langle \gamma(B_1, \ldots, B_n), Init \rangle$
Output:   $\mathcal{S}$ is deadlock-free or has a set of potential deadlocks.

1. Find $\Phi$ an invariant of $\mathcal{S}$
2. Compute $DIS$ for $\gamma(B_1, \ldots, B_n)$.
3. If $\Phi \wedge DIS = false$ then return "$\mathcal{S}$ is deadlock-free" else go to 4 or 6
4. Find $\Phi'$ an invariant of $\mathcal{S}$
5. $\Phi := \Phi \wedge \Phi'$ go to 3
6. return the set of the solutions that satisfy $\Phi \wedge DIS$
```

Example 9 $\Phi = \Phi_1 \wedge \Phi_2 \wedge \Phi_3$ *is the conjunction of the deadlock-free invariants given in example 5. The predicate $\Phi \wedge DIS$, where DIS is given in example 8, is satisfiable and it is the disjunction of the following terms:*

1. $(at\_l_1 \wedge 0 \leq t_1 < 3600) \wedge (at\_l_3 \wedge 0 \leq t_2 < 3600) \wedge (at\_l_6 \wedge \theta = 100)$
2. $(at\_l_1 \wedge 0 \leq t_1 < 3600) \wedge (at\_l_4 \wedge t_2 \geq 3600) \wedge (at\_l_5 \wedge \theta = 1000)$
3. $(at\_l_1 \wedge 0 \leq t_1 < 3600) \wedge (at\_l_3 \wedge 0 \leq t_2 < 3600) \wedge (at\_l_5 \wedge \theta = 1000)$
4. $(at\_l_2 \wedge t_1 \geq 3600) \wedge (at\_l_3 \wedge 0 \leq t_2 < 3600) \wedge (at\_l_5 \wedge \theta = 1000)$
5. $(at\_l_2 \wedge t_1 \geq 3600) \wedge (at\_l_4 \wedge t_2 \geq 3600) \wedge (at\_l_5 \wedge \theta = 1000)$

Each one of the above terms represents a family of possible deadlocks. To decrease the number of potential deadlocks, we find a new invariant Φ' stronger than Φ, such that $\Phi' = \Phi \wedge \Phi_{int}$, where Φ_{int} is an invariant on the states of Rod1, Rod2 and Controller induced by the interactions:

$$\begin{aligned}
&(\ (at\_l_2 \wedge t_1 \geq 3600) \vee (at\_l_4 \wedge t_2 \geq 3600) \vee (at\_l_5 \wedge 100 \leq \theta \leq 1000)\ )\\
\bigwedge\ &(\ (at\_l_1 \wedge t_1 \geq 0) \vee (at\_l_2 \wedge t_1 \geq 3600) \vee (at\_l_3 \wedge t_2 \geq 0) \vee\ (at\_l_4 \wedge t_2 \geq 3600)\ )\\
\bigwedge\ &(\ (at\_l_3 \wedge t_2 \geq 1) \vee (at\_l_4) \vee (at\_l_5 \wedge \theta = 100)\ )\\
\bigwedge\ &(\ (at\_l_1 \wedge t_1 \geq 0) \vee (at\_l_3 \wedge t_2 \geq 0) \vee (at\_l_6 \wedge \theta = 1000) \vee (at\_l_6 \vee 100 \leq \theta \leq 998)\ )\\
\bigwedge\ &(\ (at\_l_1 \wedge t_1 \geq 1) \vee (at\_l_2) \vee (at\_l_5 \wedge \theta = 100)\ )
\end{aligned}$$

The predicate $\Phi' \wedge DIS$ is reduced to:

6. $(at\_l_1 \wedge 1 \leq t_1 < 3600) \wedge (at\_l_3 \wedge 1 \leq t_2 < 3600) \wedge (at\_l_5 \wedge \theta = 1000)$

7. $(at\_l_1 \wedge 1 \leq t_1 < 3600) \wedge (at\_l_4 \wedge t_2 \geq 3600) \wedge (at\_l_5 \wedge \theta = 1000)$

8. $(at\_l_2 \wedge t_1 \geq 3600) \wedge (at\_l_3 \wedge 1 \leq t_2 < 3600) \wedge (at\_l_5 \wedge \theta = 1000)$

Finally, it can be checked by using finite state reachability analysis on an abstraction of the system without variables, that only the first term represents feasible deadlocks, the two other being spurious. This term characterizes deadlock configurations leading to complete shutdown.

4.1 The D-Finder Toolset

The D-Finder toolset allows deadlock verification by application of the method (Figure 5). It takes as input a BIP model and computes component invariants Φ_i by using Proposition 2. This step may require quantifier elimination by using Omega. Then, it checks for deadlock-freedom of component invariants by using Yices. From the generated component invariants, it computes an abstraction of the BIP model and the corresponding interaction invariants Ψ. Then, it checks satisfiability of the conjunction $\Psi \wedge \bigwedge_{i=1} \Phi_i \wedge DIS$. If the conjunction is unsatisfiable, then there is no deadlock else either it generates stronger component and interaction invariants or it tries to confirm the detected deadlocks by using reachability analysis techniques.

4.2 Experimental results

We provide experimental results for five examples. The first example is the Temperature Control System extensively presented in the paper. The second example is Utopar, a case study of the European Integrated project SPEEDS (http://www.speeds.eu.com/) about an automated transportation system. A succinct description of Utopar can be found at http://www.combest.eu/home/?link=Application2. The system is the composition of three types of components: autonomous vehicles, called U-cars, a centralized Automatic Control System and Calling Units. The latter two types have (almost exclusively) discrete behavior. U-cars are equipped with a local controller, responsible for handling the U-cars sensors and performing various routing and driving computations depending on users' requests. We analyzed a simplified version of Utopar by abstracting from data exchanged between components as well as from continuous dynamics of the cars. In this version, each U-Car is modeled by a component having 7 control locations and 6 integer variables. The Automatic Control System has 3 control locations and 2 integer variables. The Calling Units have 2 control locations and no variables.

Finally, the last three examples are classic finite-state benchmarks for compositional verification methods: dining philosophers, readers-writers and gas station [15]. We consider them in order to evaluate how the method scales up for components without data.

Table 1 provides an overview of the experimental results obtained for the examples. For the columns: n is the number of BIP components in the example, q is the total number of control locations, x_b (resp. x_i) is the total number of boolean (resp. integer) variables, D provides, when possible, the estimated number of deadlock configurations in DIS, D_c (resp. D_{ci}) is the number of deadlock configurations remaining in $DIS \wedge \bigwedge_i \Phi_i$ (resp. $DIS \wedge \bigwedge_i \Phi_i \wedge \Psi$) and t is the total time for computing invariants and checking for satisfiability of $DIS \wedge \bigwedge_i \Phi_i \wedge \Psi$. Detailed results are available at http://www-verimag.imag.fr/~ thnguyen/tool.

example	n	q	x_b	x_i	D	D_c	D_{ci}	t
Temperature Control System (2 rods)	3	6	0	3	8	5	3	3s
Temperature Control System (4 rods)	5	10	0	5	32	17	15	6s
Utopar System (40 U-Cars, 256 Calling Units)	297	795	40	242	-	-	0	3m46s
Utopar System (60 U-Cars, 625 Calling Units)	686	1673	60	362	-	-	0	25m29s
Readers-Writer (7000 readers)	7002	14006	0	1	-	-	0	17m27s
Readers-Writer (10000 readers)	10002	20006	0	1	-	-	0	36m10s
Gas station (100 pumps - 1000 customers)	1101	4302	0	0	-	-	0	9m14s
Philosophers (2000 Philos)	4000	10000	0	0	-	-	3	32m14s
Philosophers (3001 Philos)	6001	15005	0	0	-	-	1	54m34s

Table 1: Experimental results

We also compared D-Finder to some well-known monolithic verification tools such as NuSMV and Spin. We ran the benchmarks on a Linux machine Intel Pentium 4 3.0 GHz and 1G Ram.

In [15], Cobleigh et al. show for a set of finite-state benchmarks that only for 30% of the considered benchmarks assume-guarantee tools outperform model-checking tools. On the contrary, for all the case studies that we have verified by using D-Finder and monolithic model checkers, D-Finder outperforms these tools, in particular for large systems. Of course this comparison is not completely balanced because D-Finder uses heuristics and is tuned for checking deadlock-freedom.

The first comparison between NuSMV, Spin and D-Finder is on Dining Philosophers example. We increase the number of philosophers and compare the verification time between these three tools (Figure 6). Spin runs out of memory at the size 17 (philosophers); NuSMV runs out of memory at the size 150 while D-Finder can go much further until the size 3000.

The second comparison between NuSMV and D-Finder is on Gas Station example. We consider a system with 3 pumps and increase the number of customers. The comparison of verification time is in Figure 7. NuSMV runs out of memory at the size 180 (customers) while D-Finder can go much further until the size 3000.

5 Conclusion

The paper presents a compositional method for invariant verification of component-based systems. In contrast to assume-guarantee methods based on assumptions, we use interaction invariants to characterize contexts of individual components. These can be computed automatically from component invariants which play the role of guarantees for individual components.

There are two key issues in the application of the method. The first is the choice of component invariants depending on the property to be proved. The second is the computation of the corresponding interaction invariants. Here there is a risk of explosion, if exhaustiveness of solutions is necessary in the analysis process. However, this issue can be solved by using symbolic computation using BDDs.

The implementation and application of the method for proving deadlock-freedom of component-based systems is promising. We use a class of component invariants which capture well-enough guarantees for component deadlock-freedom. Their computation does not involve fixpoints and avoids state space exploration. D-Finder applies an iterative process for computing progressively stronger invariants. Best precision is achieved when component reachability sets are used as component invariants. This is feasible for finite state components. There are no restrictions on the type of data as long as we stay within theories for which there exist efficient decision procedures.

The obtained experimental results for non trivial case studies are really convincing. The method can be adapted to interactions with data transfer. Data transfer with finite domains, can be encoded by creating individual interactions for each configuration of transferred data. Otherwise, the notion of component invariant and subsequently the notion of interaction invariant can be extended to take into account transferred data. Finally, an interesting work direction is extending D-Finder to prove properties other than deadlock-fredom.

References

[1] Kupferman, O., Vardi, M.Y.: Modular model checking. In: Compositionality: The Significant Difference. International Symposium, COMPOS'97, Bad Malente, Germany. Volume 1536 of LNCS., Springer-Verlag (1998) 381–401

[2] Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time components in bip. In: Proceedings of SEFM'06, Pune, India, IEEE Computer Society Press (2006) 3–12

[3] Peterson, J.: Petri Net theory and the modelling of systems. Englewood-Cliffs: Prentice Hall (1981)

[4] Team, O.: The omega library. Version 1.1.0 (November 1996)

[5] Somenzi, F.: CUDD Decision Diagram Package. Colorado University

[6] Dutertre, B., de Moura, L.: A fast linear-arithmetic solver for DPLL(T). In: Proceedings of CAV'06, Seattle, WA, USA. Volume 4144 of LNCS. (2006) 81–94

[7] Alur, R., Henzinger, T.: Reactive modules. In: Proceedings of the 11th Annual Symposium on LICS, IEEE Computer Society (1996) 207–208

[8] Abadi, M., Lamport, L.: Conjoining specification. ACM Transactions on Programming Languages and Systems **17**(3) (1995) 507–534

[9] Clarke, E., Long, D., McMillan, K.: Compositional model checking. In: Proceedings of the 4th Annual Symposium on LICS, IEEE Computer Society Press (1989) 353–362

[10] Chandy, K., J.Misra: Parallel program design: a foundation. Addison-Wesley Publishing Company (1988)

[11] Grumberg, O., Long, D.E.: Model checking and modular verification. ACM Transactions on Programming Languages and Systems **16**(3) (1994) 843–871

[12] McMillan, K.L.: A compositional rule for hardware design refinement. In: Proceedings of CAV'97, Haifa, Israel. Volume 1254 of LNCS., Springer-Verlag (1997) 24–35

[13] Pnueli, A.: In transition from global to modular temporal reasoning about programs. In: Logics and models of concurrent systems. Springer-Verlag, Inc., New York, USA (1985) 123–144

[14] Stark, E.W.: A proof technique for rely/guarantee properties. In: Proceedings of FSTTCS'85. Volume 206 of LNCS., Springer-Verlag (1985) 369–391

[15] Cobleigh, J.M., Avrunin, G.S., Clarke, L.A.: Breaking up is hard to do: An evaluation of automated assume-guarantee reasoning. ACM Transactions on Software Engineering and Methodology **17**(2) (2008)

[16] Apt, K.R., Francez, N., de Roever, W.P.: A proof system for communicating sequential processes. ACM Trans. Program. Lang. Syst. **2**(3) (1980) 359–385

[17] Manna, Z., Pnueli, A.: Temporal Verification of Reactive Systems: Safety. Springer-Verlag (1995)

[18] Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. TCS **138**(1) (1995) 3–34

[19] Bliudze, S., Sifakis, J.: The Algebra of Connectors — Structuring Interaction in BIP. In: Proceeding of the EMSOFT'07, Salzburg, Austria, ACM SigBED (October 2007) 11–20

[20] Basu, A.: Component-based Modeling of Heterogeneous Real-Time Systems in BIP. PhD thesis, Université Joseph Fourier (December 2008)

[21] Bozga, M., Jaber, M., Sifakis, J.: Source-to-Source Architecture Transformation for Performance Optimization in BIP. In: Proceedings of SIES'09 - IEEE Symposium on Industrial Embedded Systems - Lausanne, Switzerland. (July 2009)

[22] Bensalem, S., Lakhnech, Y.: Automatic generation of invariants. FMSD **15**(1) (July 1999) 75–92

[23] Sifakis, J.: Structural properties of petri nets. In: Proceedings of MFCS'78, Zakopane, Poland. Volume 64 of LNCS. (1978) 474–483

[24] Yamauchi, M., Watanabe, T.: Time complexity analysis of the minimal siphon extraction problem of petri nets. IEICE Transactions on Communications/Electronics/Information and Systems (1999)

[25] Tanimoto, S., Yamauchi, M., Watanabe, T.: Finding minimal siphons in general petri nets. IEICE Trans. on Fundamentals in Electronics, Communications and Computer Science **E79-A**(11) (1996) 1817–1824

[26] Comoner, F.: Deadlocks in petri nets. Technical Report CA-7206-2311, Massachusetts Computer Associates, Wakefield, Mass. (June 1972)

[27] Barkaoui, K., Minoux, M.: A polynomial-time graph algorithm to decide liveness of some basic classes of bounded petri nets. In Jensen, K., ed.: Application and Theory of Petri Nets 1992, 13th International Conference, Sheffield, UK, June 22-26, 1992, Proceedings. Volume 616 of LNCS., Springer (1992) 62–75

[28] Bensalem, S., Lakhnech, Y., Owre, S.: Computing abstractions of infinite state systems automatically and compositionally. In: Proceedings of CAV'98, Vancouver, BC, Canada. Volume 1427 of LNCS. 319–331

[29] Bensalem, S., Lakhnech, Y., Owre, S.: Invest: A tool for the verification of invariants. In: Proceedings of CAV'98, Vancouver, BC, Canada. Volume 1427 of LNCS. 505–510

[30] Angluin, D.: Learning regular sets from queries and counterexamples. In: Inf. Comput., Volume 75, Issue 2, Academic Press, Inc. (1987) 87–106

[31] Păsăreanu, Corina S. and Giannakopoulou, Dimitra and Bobaru, Mihaela Gheorghiu and Cobleigh, Jamieson M. and Barringer, Howard: Learning to divide and conquer: applying the L* algorithm to automate assume-guarantee reasoning. In: Form. Methods Syst. Des. Volume 32, Issue 3, Kluwer Academic Publishers (2008) 175–205

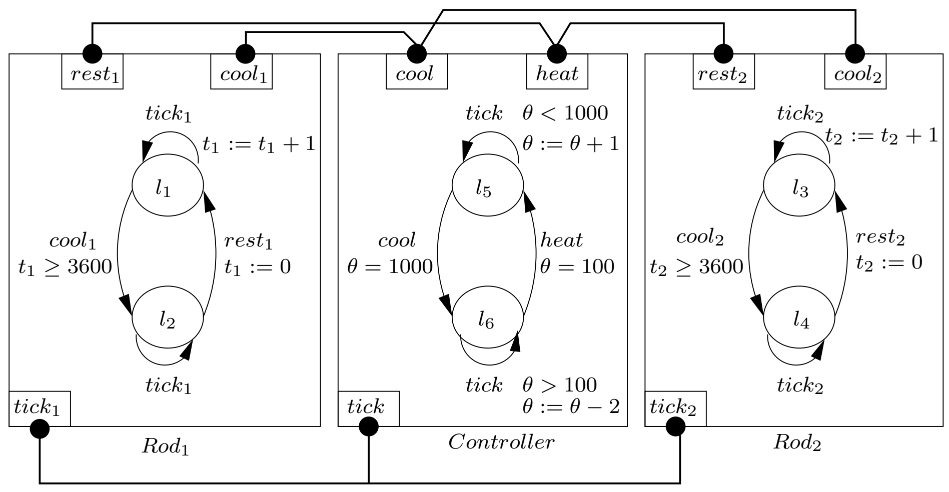


Figure 1: Temperature Control System Example.

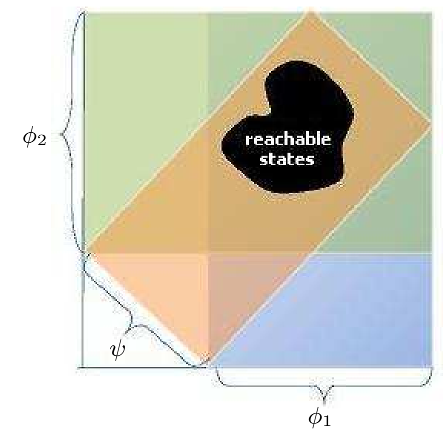
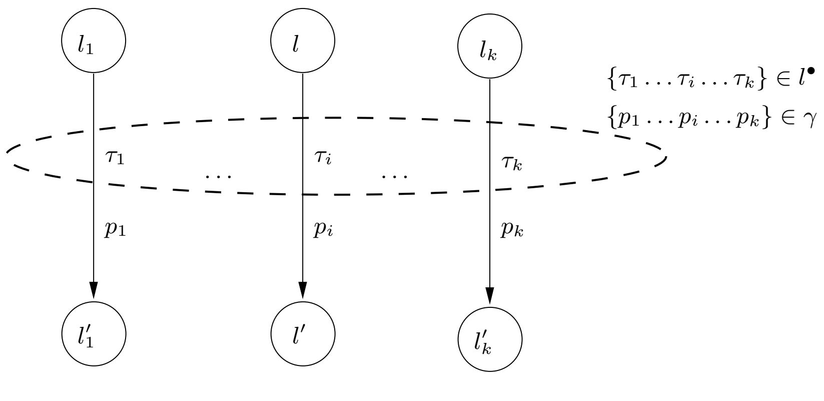


Figure 2: Illustration of the method.



l_1
l
l_k
$\{\tau_1 \dots \tau_i \dots \tau_k\} \in l^\bullet$
$\{p_1 \dots p_i \dots p_k\} \in \gamma$
τ_1
$\dots$
τ_i
$\dots$
τ_k
p_1
p_i
p_k
l'_1
l'
l'_k

Figure 3: Forward interaction sets.

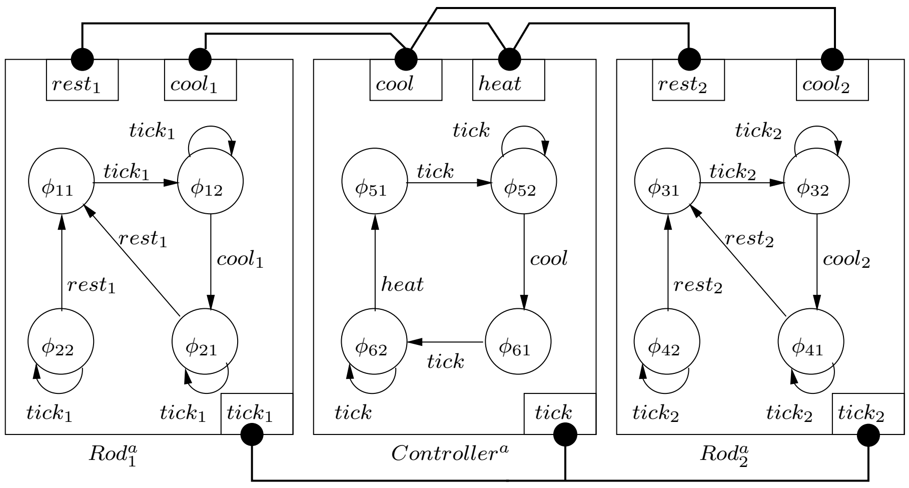


Figure 4: Abstraction of the Temperature Control System.

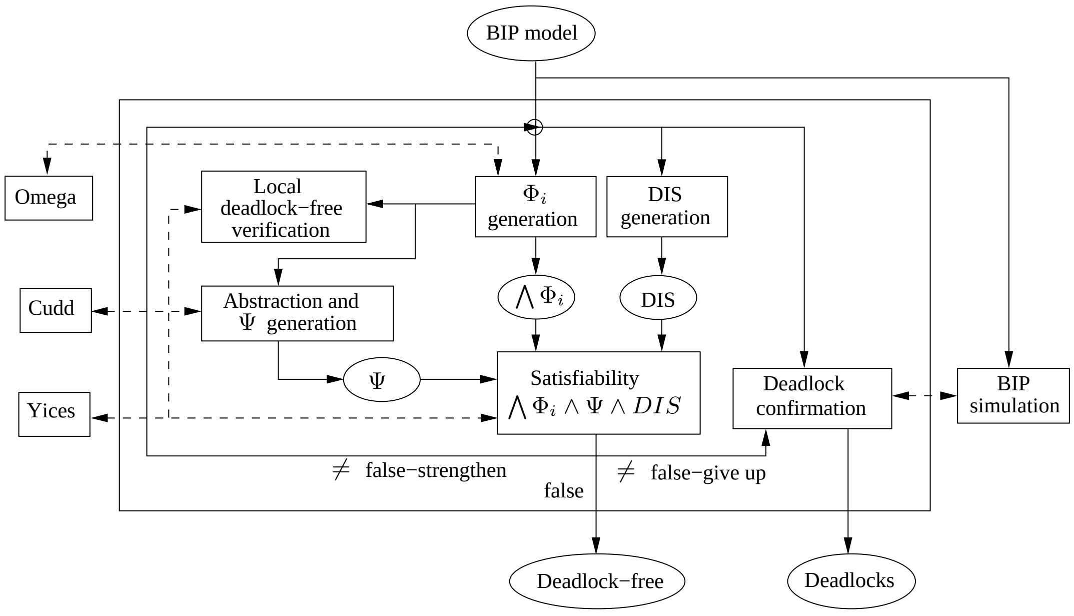


Figure 5: D-Finder Tool Architecture.

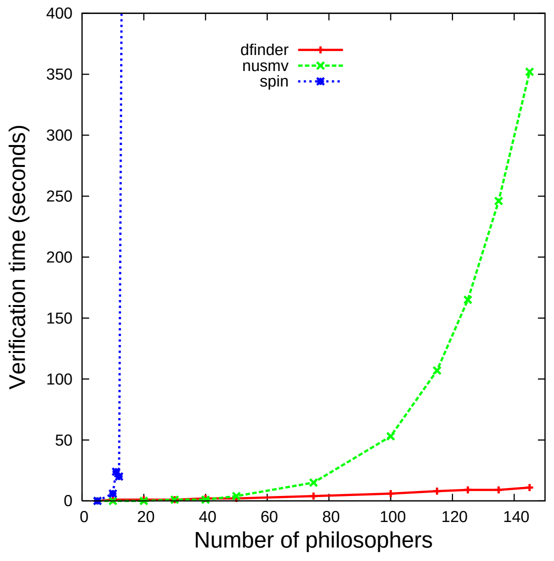


Figure 6: Comparison with NuSMV and Spin on Dining Philosophers

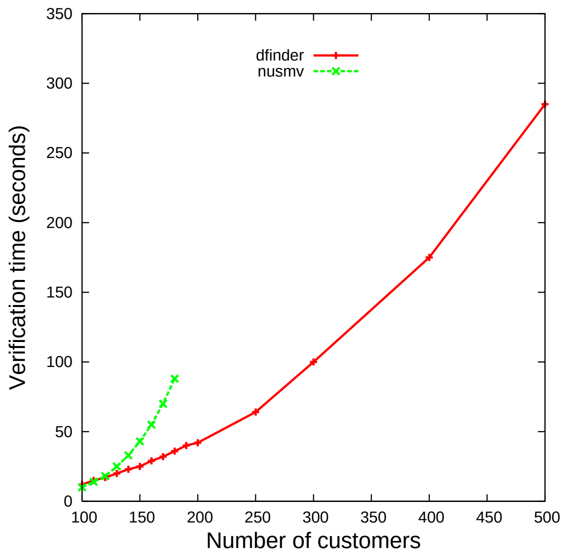


Figure 7: Comparison with NuSMV on Gas Station