Compositional Verification for Component-based Systems and Application

Saddek Bensalem, Marius Bozga, Thanh-Hung Nguyen, Joseph Sifakis

Compositional Verification for Component-based Systems and Application

Saddek Bensalem Marius Bozga Thanh-Hung Nguyen Joseph Sifakis

Verimag Laboratory, Université Joseph Fourier Grenoble, CNRS.

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 which are over-approximations of components' reachability sets. Interaction invariants which are constraints on the states of components involved in interactions. Interaction invariants are obtained by computing traps of finite-state abstractions of the verified system. The method is applied for deadlock verification in the D-Finder tool. D-Finder is an interactive tool that takes as input BIP programs and applies proof strategies to eliminate potential deadlocks by computing increasingly stronger invariants. The experimental results on non-trivial examples 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 aply 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

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

That is, if two components with behaviors B_1, B_2 meet individually 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 individual 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.

One approach to compositional verification is by *assume-guarantee* where properties are decomposed 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 [2–9]). Many issues make the application of assume-guarantee rules diffcult. These are discussed in detail in a recent paper [10] 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.

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

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

The rule allows to prove invariance of Φ for systems obtained by using a n-ary composition operation parameterized by a set of interactions γ. It uses global invariants which are the conjunction of individual invariants of components Φ_i and an interaction invariant Ψ. The latter expresses constraints on the global state space induced by interactions between components. It can be computed automatically from abstractions of the system to be verified. These are the composition 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 [11] of the Petri net and are computed symbolically as solutions of a set of boolean equations.

Figure 1 illustrates the method for a system with two components, invariants Φ_1 and Φ_2 and interaction invariant Ψ. 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 [12] as they allow to eliminate product states which are not feasible by the semantics of parallel composition.

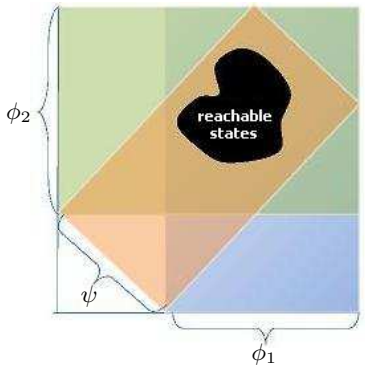

Fig. 1.

The paper provides a method for automated verification of component-based systems described in a subset of the BIP (Behavior-Interaction-Priority) language [13]. In BIP, a system is the composition of a set of atomic components which are automata extended with data and functions written in C. We restrict to programs where interactions are pure synchronizations. Nonetheless, the method can be easily extended for interactions involving data transfer. The main results are the following:

– We provide heuristics for computing component invariants and interaction invariants. Component invariants are over-approximations of the set of the reachable states generated by simple forward analysis. Interaction invariants are derived automatically from component invariants and their interactions. When proving invariance of a property fails, it is possible to find stronger invariants by computing stronger component invariants from which stronger interaction invariants are obtained.
– We present an implemention and application of the method in the D-Finder tool for deadlock verification. D-Finder takes as input BIP programs and progressively eliminates potential deadlocks by generating invariants. For this, it cooperates with two tools: Omega [14] for quantifier elimination and Yices [15] 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.

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.

2 Models, Invariants and their Properties

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

2.1 Basic model for BIP

We present a simplified model for component-based systems used in the *Behaviour-Interaction-Priority* (BIP) component framework developed at Verimag [13].

This framework has been implemented in a language and a toolset. The BIP language offers primitives and constructs for modelling and composing components. An atomic component consists of a set of ports used for the synchronization with other components, a set of transitions and a set of local variables. Transitions describe the behavior of the component. The BIP toolset includes an editor and a compiler for generating from BIP programs, C++ code executable on a dedicated platform.

We provide a formalization of atomic components in BIP and their composition by using interactions.

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*

  - $L = \{l_1, l_2, \ldots, l_k\}$ *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,*
- $X = \{x_1, \ldots, x_n\}$ *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.*

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* $(l, \mathbf{x}) \xrightarrow{p} (l', \mathbf{x}')$*.*

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 a parallel composition for components parameterized by a set of interactions. We consider only pure synchronizations, that is interactions do not involve 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* $\forall i = 1, \ldots, n \ |a \cap P_i| \leq 1$*.*

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* $(L, \gamma, \mathcal{T}, X, \{g_\tau\}_{\tau\in\mathcal{T}}, \{f_\tau\}_{\tau\in\mathcal{T}})$*, 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* $f_\tau = \bigwedge_{i\in I} f_{\tau_i} \wedge \bigwedge_{i\notin I}(X'_i = X_i)$*.*

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*.*

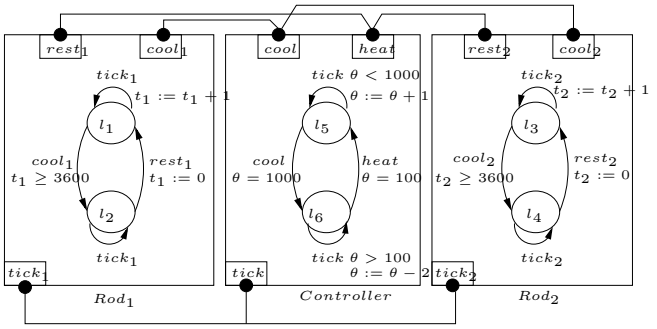

Fig. 2. Temperature Control System

Example 1 (Temperature Control System). [16] 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.

We provide 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 components are composed by using the following set of interactions, indicated by connectors in the figure: $\{tick, tick_1, tick_2\}$, $\{cool, cool_1\}$, $\{cool, cool_2\}$, $\{heat, rest_1\}, \{heat, rest_2\}$.

In our 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 $post(\Phi) = \bigvee_{l\in L} at\_l \wedge (\bigvee_{\tau=(l',p,l)} post_\tau(\varphi_{l'}))$. This allows computing $post(\Phi)$ 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_0 such that $\Phi_0 \Rightarrow \Phi$.*

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 use the following rule:

$$\frac{\{B_i < \Phi_i >\}_i^n,\ \Psi \in II(\gamma(B_1, \ldots, B_n), \{\Phi_i\}_i^n),\ (\bigwedge_i^n \Phi_i) \wedge \Psi \Rightarrow \Phi}{\gamma(B_1, \ldots, B_n) < \Phi >} \qquad (3)$$

where $B_i < \Phi_i >$ means that Φ_i is an invariant of component B_i and Ψ is an interaction invariant of $\gamma(B_1, \ldots, B_n)$ computed automatically from Φ_i and $\gamma(B_1, \ldots, B_n)$.

A key issue in the application of this rule is finding component invariants Φ_i. If the components B_i are finite state, then we can take $\Phi = Reach(B_i)$, the set of reachable state of B_i, or any upper approximation of $Reach(B_i)$. If the components are infinite state, $Reach(B_i)$ can be approximated as shown in [17, 18].

We provide below methods for computing component invariants used for checking deadlock-freedom (section 4). We also provide a general method for computing interaction invariants for $\gamma(B_1, \ldots, B_n)$ from a given set of component invariants Φ_i.

3.1 Computing Component Invariants

We present below a method for the lightweight computation of sequences of inductive invariants for atomic components. This method is used in the D-Finder toolset.

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

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

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 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 [19].

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 $post_\tau(\varphi)$

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

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

Example 2. For the Temperature Control System of figure 2, 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.

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*

$$l^\bullet = \left\{ \{\tau_i\}_{i\in I} \mid \forall i.\tau_i \in \mathcal{T}_i \ \wedge \ \exists i.^\bullet\tau_i = l \ \wedge \ \{port(\tau_i)\}_{i\in I} \in \gamma \right\}$$

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 location its backward interaction set

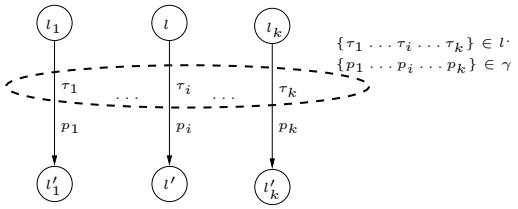

Fig. 3. Forward interaction sets.

$${}^\bullet L = \bigcup_{l \in L} {}^\bullet l \text{ where } {}^\bullet l = \left\{ \{\tau_i\}_{i \in I} \mid \forall i.\tau_i \in \mathcal{T}_i \ \wedge \ \exists i.\tau_i^\bullet = l \ \wedge \ \{port(\tau_i)\}_{i \in I} \in \gamma \right\}$$

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 can 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 $\mathcal{S}$.*

The following result given in [20] characterizes traps as solution of a system of implications.

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

$$\{l \in \bigcup_{i=1}^{n} L_i \mid \mathbf{v}(l) \ = \ true\}$$

is a trap, where $\mathbf{v}(l) \ \Rightarrow \bigwedge_{\{\tau_i\}_{i \in I} \in l^\bullet} \left(\bigvee_{l' \in \{\tau_i^\bullet\}_{i \in I}} \mathbf{v}(l') \right)$ *for* $l \in \bigcup_{i=1}^n L_i$

This characterization allows to compute by enumeration the minimal traps of $\gamma(B_1, ..., B_n)$. For this we use Yices [15] to successively obtain minimal solutions of the above system. As shown in [21, 22] computing the set of minimal traps is a NP-complete problem and in practice the trap extraction process may not be exhaustive.

Example 3. The set of 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 $L_5 = \{\phi_{12}, \phi_{21}, \phi_{22}, \phi_{51}\}$.

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 composionally 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 atomic predicates of the form $at\_l \wedge \varphi_{lm}$ are disjoint. 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 [23] and implemented in the InVeSt tool [24] 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 4. 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 2.

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

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

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 $B = \gamma(B_1, ..., B_n)$

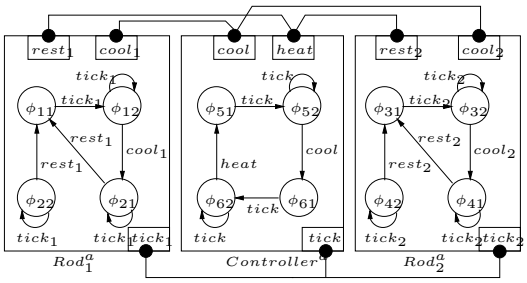

Fig. 4. Abstraction of the Temperature Control System.

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 Φ and its abstraction function α, 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 $\langle B, Init\rangle$.*

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 (3). 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. 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^α simulates $B^{\alpha'}$*

For two successive component invariants Φ_i and Φ_i' for 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'

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

Fig. 5. Iterative application of the rule in figure 3

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 $\gamma(B_1^{\alpha'_1}, ..., B_n^{\alpha'_n})$. We can show 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 $\gamma(B'_1, ..., B'_n)$ there exists a stronger interaction invariant of $\gamma(B_1, ..., B_n)$.

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 5. For the Temperature Control System (see figure 2), we have:
$DIS = (\neg(at\_l_5 \wedge \theta < 1000)) \bigwedge (\neg(at\_l_6 \wedge \theta = 100) \vee \neg at\_l_2)$
$\bigwedge (\neg(at\_l_6 \wedge \theta > 100)) \bigwedge (\neg(at\_l_5 \wedge \theta = 1000) \vee \neg(at\_l_3 \wedge t_2 \geq 3600))$
$\bigwedge (\neg(at\_l_5 \wedge \theta = 1000) \vee \neg(at\_l_1 \wedge t_1 \geq 3600)) \bigwedge (\neg(at\_l_6 \wedge \theta = 100) \vee \neg at\_l_4)$
□

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 $\Phi \wedge DIS = false$.

Figure 6 presents the verification heuristic for a system $\langle \gamma(B_1, \ldots, B_n), Init \rangle$ applied by the D-Finder toolset.

Example 6. $\Phi = \Phi_1 \wedge \Phi_2 \wedge \Phi_3$ is the conjunction of the deadlock-free invariants given in example 2. The predicate $\Phi \wedge DIS$, where DIS is given in example 5, is satisfiable and it is the disjunction of the following terms:

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 Φ 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 Φ' an invariant of $\mathcal{S}$
5. $\Phi := \Phi \wedge \Phi'$ go to 3
6. return the set of the solutions that satisfy $\Phi \wedge DIS$

Fig. 6. Heuristic for Deadlock Verification

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:

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

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. □

The D-Finder Toolset The D-Finder toolset allows deadlock verification by application of the method (figure 7). It takes as input a BIP model and computes component invariants CI 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 II. Then, it checks satisfiability of the conjunction $II \wedge CI \wedge DIS$. 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.

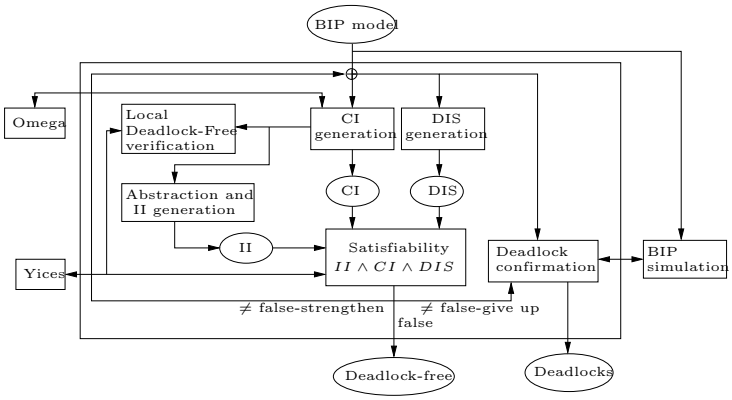

Fig. 7. D-Finder

Experimental results We provide experimental results for three examples. The first example is the Temperature Control System extensively presented in the paper. The second example is Utopar, an industrial 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, as third example, we consider Readers-Writer systems in order to evaluate how the method scales up for components without data.

The table below provides an overview of the experimental results obtained for the three 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 $DIS \wedge CI$ (resp. $DIS \wedge CI \wedge II$) and t is the total time for computing invariants and checking for satisfiability of $DIS \wedge CI \wedge II$. 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	1m05s
Utopar System (4 U-Cars, 9 Calling Units)	14	45	4	26	-	-	0	1m42s
Utopar System (8 U-Cars, 16 Calling Units)	25	91	8	50	-	-	0	22m02s
Readers-Writer (50 readers)	52	106	0	1	$\sim 10^{15}$	$\sim 10^{15}$	0	1m15s
Readers-Writer (100 readers)	102	206	0	1	$\sim 10^{30}$	$\sim 10^{30}$	0	15m28s
Readers-Writer (130 readers)	132	266	0	1	$\sim 10^{39}$	$\sim 10^{39}$	0	29m13s

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.

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. LNCS **1536** (1998) 381–401
2. Alur, R., Henzinger, T.: Reactive modules. In: Proceedings of the 11th Annual Symposium on LICS, IEEE Computer Society Press (1996) 207–208
3. Abadi, M., Lamport, L.: Conjoining specification. ACM Transactions on Programming Languages and Systems **17**(3) (1995) 507–534

4. 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
5. Chandy, K., J.Misra: Parallel program design: a foundation. Addison-Wesley Publishing Company (1988)
6. Grumberg, O., Long, D.E.: Model checking and modular verification. ACM Transactions on Programming Languages and Systems **16**(3) (1994) 843–871
7. McMillan, K.L.: A compositional rule for hardware design refinement. In: CAV '97, Springer-Verlag (1997) 24–35
8. Pnueli, A.: In transition from global to modular temporal reasoning about programs. (1985) 123–144
9. Stark, E.W.: A proof technique for rely/guarantee properties. In: FSTTCS: proceedings of the 5th conference. Volume 206., Springer-Verlag (1985) 369–391
10. 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)
11. Peterson, J.: Petri Net theory and the modelling of systems. Englewood-Cliffs: Prentice Hall (1981)
12. 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
13. Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time components in bip. In: SEFM. (2006) 3–12
14. Team, O.: The omega library. Version 1.1.0 (November 1996)
15. Dutertre, B., de Moura, L.: A fast linear-arithmetic solver for DPLL(T). In: CAV'06. Volume 4144 of LNCS. (2006) 81–94
16. 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
17. Lakhnech, Y., Bensalem, S., Berezin, S., Owre, S.: Incremental verification by abstraction. In: TACAS. (2001) 98–112
18. Bradley, A.R., Manna, Z.: Checking safety by inductive generalization of counterexamples to induction. In: FMCAD. (2007) 173–180
19. Bensalem, S., Lakhnech, Y.: Automatic generation of invariants. FMSD **15**(1) (July 1999) 75–92
20. Sifakis, J.: Structural properties of petri nets. In: MFCS'78. Volume 64 of LNCS. (1978) 474–483
21. 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)
22. 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
23. Bensalem, S., Lakhnech, Y., Owre, S.: Computing abstractions of infinite state systems automatically and compositionally. In: CAV'98. Volume 1427 of LNCS. 319–331
24. Bensalem, S., Lakhnech, Y., Owre, S.: Invest: A tool for the verification of invariants. In: CAV'98. Volume 1427 of LNCS. 505–510