Incremental Component-based Construction and Verification using Invariants

Saddek Bensalem[1] Marius Bozga[1] Axel Legay[2] Thanh-Hung Nguyen[1] Joseph Sifakis[1] Rongjie Yan[1]
[1] Verimag Laboratory, Université Joseph Fourier Grenoble, CNRS
[2]INRIA/IRISA, Rennes

***Abstract*—We propose invariant-based techniques for the efficient verification of safety and deadlock properties of concurrent systems. We assume that components and component interactions are described within the BIP framework, a tool for component-based design. We build on a compositional methodology in which the invariant is obtained by combining the invariants of the individual components with an interaction invariant that takes concurrency and interaction between components into account. In this paper, we propose new efficient techniques for computing interaction invariants. This is achieved in several steps. First, we propose a formalization of incremental component-based design. Then we suggest sufficient conditions that ensure the preservation of invariants through the introduction of new interactions. For cases in which these conditions are not satisfied, we propose methods for generation of new invariants in an incremental manner. The reuse of existing invariants reduces considerably the verification effort. Our techniques have been implemented in the D-Finder toolset. Among the experiments conducted, we have been capable of verifying properties and deadlock-freedom of DALA, an autonomous robot whose behaviors in the functional level are described with 500000 lines of C Code. This experiment, which is conducted with industrial partners, is far beyond the scope of existing academic tools such as NuSMV or SPIN.**

I. Introduction

Model Checking [10, 14] of concurrent systems is a challenging problem. Indeed, concurrency often requires computing the product of the individual systems by using both interleaving and synchronization. In general, the size of this structure is prohibitive and cannot be handled without manual interventions. In a series of recent works, it has been advocated that *compositional verification techniques* could be used to cope with state explosion in concurrent systems. Component-based design techniques confer numerous advantages, in particular, through reuse of existing components. A key issue is the existence of composition frameworks ensuring the correctness of composite components. We need frameworks allowing us not only reuse of components but also reuse of their properties for establishing global properties of composite components from properties of their constituent components. This should help cope with the complexity of global monolithic verification techniques.

Compositionality allows us to infer global properties of complex systems from properties of their components. The idea of compositional verification techniques is to apply divide-and-conquer approaches to infer global properties of complex systems from properties of their components. They are used to cope with state explosion in concurrent systems. Nonetheless, we also should consider the behavior and properties resulted from mutually interacting components. A compositional verification method based on invariant computation is presented in [3, 2]. This approach is based on the following rule:

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

The rule allows to prove invariance of property Φ for systems obtained by using an n-ary composition operation $||$ parameterized by a set of interactions γ. It uses global invariants that are the conjunction of individual invariants Φ_i of individual components B_i and an *interaction invariant* Ψ. The latter expresses constraints on the global state space induced by interactions between components. In [3], we have shown that Ψ 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. The approach has been implemented in the D-Finder toolset [2] and applied to check deadlock-freedom on several case studies described in the BIP (Behavior, Interaction, Priority) [1] language. The results of these experiments show that D-Finder is exponentially faster than well-established tools such as NuSMV [9].

Incremental system design methodologies often work by adding new interactions to existing sets of components. Each time an interaction is added, one may be interested to verify whether the resulting system satisfies some given property. Indeed, it is important to report an error as soon as it appears. However, each verification step may be time consuming, which means that intermediary verification steps are generally avoided. The situation could be improved if the result of the verification process could be reused when new interactions are added. Existing techniques, including the one in [3], do not focus on such aspects. In a very recent work [6], we have proposed a new fixed point based technique that takes incremental design into account. This technique is generally faster than the one in [3] for systems with an acyclic topology. For systems with a cyclic topology, the situation may however be reversed. There are also many case studies that are beyond the scope of these techniques.

In this paper, we continue the quest for efficient incremental

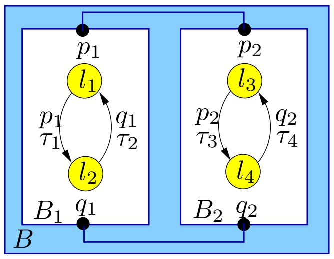

Fig. 1. A simple example

techniques for computing invariants of concurrent systems. We present a detailed methodology for incremental construction and verification of component-based systems. This is achieved in several steps. First, we propose a formalization of incremental component-based design. Then we suggest sufficient conditions that ensure the preservation of invariants through the introduction of new interactions. For cases in which these conditions are not satisfied, we propose methods for generation of new invariants in an incremental manner. The reuse of existing invariants reduces considerably the verification effort. Contrary to the technique in [6], our technique, which relies on a relation between behaviors of components and interactions, turns out to be efficient for both cyclic and acyclic topologies.

Our techniques have been implemented as extensions of the D-Finder toolset [2] and applied on several case studies. Our experiments show that our new methodology is generally much faster than the ones proposed in [3, 6]. In particular, we have been capable of verifying deadlock-freedom and safety properties of DALA, an autonomous robot whose behaviors in the functional level are described with 500000 lines of C Code. This experiment, which is conducted with industrial partners, is far beyond the scope of [3, 6] and of existing academic tools such as NuSMV or SPIN.

Structure of the paper. In section II, we recap the concepts that will be used through the paper as well as the incremental methodology introduced in [6]. Section III discusses sufficient conditions for invariant preservation while Section IV presents our incremental construction for invariants. Section V discusses the experiments. Finally, Section VI concludes the paper. Due to space limitation, some proofs and model descriptions are available from http://www-verimag.imag.fr/~yan/.

II. Preliminaries

In this section, we present concepts and definitions that will be used through the rest of the paper. We start with the concepts of *components*, *parallel composition of components*, *systems*, and *invariants*. In the second part of the section, we will recap a very recent methodology [6] we proposed for *incremental design* of composite systems.

A. Components, Interactions, and Invariants

In the paper, we will be working with a simplified model for component-based design. Roughly speaking, an atomic component is nothing more than a transition system whose transitions' labels are called *ports*. These ports are used to synchronize with other components. Formally, we have the following definition.

Definition 1 (Atomic Component)**.** *An atomic component is a transition system $B = (L, P, \mathcal{T})$, where:*

- $L = \{l_1, l_2, \ldots, l_k\}$ *is a set of locations,*
- P *is a set of ports, and*
- $\mathcal{T} \subseteq L \times P \times L$ *is a set of transitions.*

Given $\tau = (l, p, l') \in \mathcal{T}$, l and l' are the *source* and *destination* locations, respectively. In the rest of the paper, we use ${}^\bullet\tau$ and $\tau^\bullet$ to compute the source and destination of τ, respectively.

Example 1. *Figure 1 presents two atomic components. The ports of component B_1 are p_1 and q_1. B_1 has two locations: l_1 and l_2 and two transitions: $\tau_1 = (l_1, p_1, l_2)$ and $\tau_2 = (l_2, q_1, l_1)$.*

We are now ready to define parallel composition between atomic components. In the incremental design setting, the parallel composition operation allows to build bigger components starting from *atomic components*. Any composition operation requires to define a communication mode between components. In our context, components communicate via *interactions*, i.e., by synchronization on ports. Formally, we have the following definition.

Definition 2 (Interactions)**.** *Given a set of n components $B_1, B_2, \ldots, B_n$ with $B_i = (L_i, P_i, \mathcal{T}_i)$, an interaction a is a set of ports, i.e., a subset of $\bigcup_{i=1}^n P_i$, such that $\forall i = 1, \ldots, n.$ $|a \cap P_i| \leq 1$.*

By definition, each interaction has at most one port per component. In the figures, we will represent interactions by link between ports. As an example, the set $\{p_1, p_2\}$ is an interaction between Components B_1 and B_2 of Figure 1. This interaction describes a synchronization between Components B_1 and B_2 by Ports p_1 and p_2. Another interaction is given by the set $\{q_1, q_2\}$. The idea being that a parallel composition is entirely defined by a set of interactions, which we call a *connector*. As an example the connector for B_1 and B_2 is the set $\{\{p_1, p_2\}, \{q_1, q_2\}\}$. In the rest of the paper, we simplify the notations and write $p_1 p_2 \ \ldots \ p_k$ instead of $\{p_1, \ldots, p_k\}$. We also write $a_1 + \ldots + a_m$ for the connector $\{a_1, \ldots, a_m\}$. As an example, notation for the connector $\{\{p_1, p_2\}, \{q_1, q_2\}\}$ is $p_1 \ p_2 + q_1 \ q_2$.

We now propose our definition for parallel composition. In what follows, we use I for a set of integers.

Definition 3 (Parallel Composition)**.** *Given n atomic components $B_i = (L_i, P_i, \mathcal{T}_i)$ and a connector γ, we define the parallel composition $B = \gamma(B_1, \ldots, B_n)$ as the transition system $(\mathcal{L}, \gamma, \mathcal{T})$, where:*

- $\mathcal{L} = L_1 \times L_2 \times \ldots \times L_n$ *is the set of* global *locations,*
- γ *is a set of interactions, and*
- $\mathcal{T} \subseteq \mathcal{L} \times \gamma \times \mathcal{L}$ *contains all 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 $j \notin I$.*

The idea is that components communicate by synchronization with respect to interactions. Given an interaction a, only those components that are involved in a can make a step. This is ensured by following a transition labelled by the corresponding port involved in a. If a component does not participate to the interaction, then it has to remain in the same state. In the rest of the paper, a component that is obtained by composing several components will be called a *composite component*. Consider the example given in Figure 1, we have a composite component $\gamma(B_1, B_2)$, where $\gamma = p_1\ p_2 + q_1\ q_2$. Observe that the component $\gamma_\perp(B_1, \ldots, B_n)$, which is obtained by applying the connector $\gamma_\perp = \sum_{i=1}^{n}(\sum_{p_j \in P_i} p_j)$, is the transition system obtained by interleaving the transitions of atomic components. Observe also that the parallel composition $\gamma(B_1, \ldots, B_n)$ of $B_1, \ldots, B_n$ can be seen as a *1-safe Petri net* (the number of tokens in all places is at most one) whose set of places is given by $L = \bigcup_{i=1}^{n} L_i$ and whose transitions relation is given by $\mathcal{T}$. In the rest of the paper, L will be called the *set of locations of* B, while $\mathcal{L}$ is the set of *global states*. We now define the concept of invariants, which can be used to verify properties of (parallel composition of) components. We first propose the definition of *system* that is a component with an initial set of states.

Definition 4 (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.*

In a similar way, we distinguish invariants of a component from those of a system such that the invariants of a system $\mathcal{S} = \langle B, Init\rangle$ can be obtained from those of B according to the constraint $Init$. Therefore we define invariants for a component and for a system separately.

Definition 5 (Invariants). *Given a component $B = (L, P, \mathcal{T})$, a predicate $\mathcal{I}$ on L is an invariant of B, denoted by $inv(B, \mathcal{I})$, if for any location $l \in L$ and any port $p \in P$, $\mathcal{I}(l)$ and $l \xrightarrow{p} l' \in \mathcal{T}$ imply $\mathcal{I}(l')$, where $\mathcal{I}(l)$ means that l satisfies $\mathcal{I}$. For a system $\mathcal{S} = \langle B, Init\rangle$, $\mathcal{I}$ is an invariant of $\mathcal{S}$, denoted by $inv(\mathcal{S}, \mathcal{I})$, if it is an invariant of B and if $Init \Rightarrow \mathcal{I}$.*

Clearly, if $\mathcal{I}_1$, $\mathcal{I}_2$ are invariants of B (respectively $\mathcal{S}$) then $\mathcal{I}_1 \wedge \mathcal{I}_2$ and $\mathcal{I}_1 \vee \mathcal{I}_2$ are also invariants of B (respectively $\mathcal{S}$).

Let $\gamma(B_1, \ldots, B_n)$ be the composition of n components with $B_i = (L_i,\ P_i,\ \mathcal{T}_i)$ for $i \in 1 \ldots n$. In the paper, an invariant on B_i is called a *component invariant* and an invariant on $\gamma(B_1, \ldots, B_n)$ is called an *interaction invariant*. To simplify the notations, we will assume that interaction invariants are predicates on $\bigcup_{i=1}^{n} L_i$.

B. Incremental Design

In component-based design, the construction of a composite system is both step-wise and hierarchical. This means that a system is obtained from a set of atomic components by successive additions of new interactions also called *increments*. In a very recent work [6], we have proposed a methodology to add new interactions to a composite component without breaking the synchronization. The techniques we will propose to compute and reuse invariants intensively build on this methodology, which is described hereafter.

First, when building a composite system in a bottom-up manner, it is essential that some already enforced synchronizations are not relaxed when increments are added. To guarantee this property, we propose the notion of *forbidden interactions*.

Definition 6 (Closure and Forbidden Interactions). *Let γ be a connector.*

- *The closure γ^c of γ, is the set of the non empty interactions contained in some interaction of γ. That is $\gamma^c = \{a \neq \emptyset \mid \exists b \in \gamma.\ a \subseteq b\}$.*
- *The forbidden interactions γ^f of γ is the set of the interactions strictly contained in all the interactions of γ. That is $\gamma^f = \gamma^c - \gamma$.*

It is easy to see that for two connectors γ_1 and γ_2, we have $(\gamma_1 + \gamma_2)^c = \gamma_1^c + \gamma_2^c$ and $(\gamma_1 + \gamma_2)^f = (\gamma_1 + \gamma_2)^c - \gamma_1 - \gamma_2$.

In our theory, a connector describes a set of interactions and, by default, also those interactions in where only one component can make progress. This assumption allows us to define new increments in terms of existing interactions.

Definition 7 (Increments). *Consider a connector γ over B and let $\delta \subseteq 2^\gamma$ be a set of interactions. We say δ is an increment over γ if for any interaction $a \in \delta$ we have interactions $b_1, \ldots, b_n \in \gamma$ such that $\bigcup_{i=1}^{n} b_i = a$.*

In practice, one has to make sure that existing interactions defined by γ will not break the synchronizations that are enforced by the increment δ. For doing so, we remove from the original connector γ all the interactions that are forbidden by δ. This is done with the operation of *Layering*, which describes how an increment can be added to an existing set of interactions without breaking synchronization enforced by the increment. Formally, we have the following definition.

Definition 8 (Layering). *Given a connector γ and an increment δ over γ, the new set of interactions obtained by combining δ and γ, also called layering, is given by the following set $\delta\gamma = (\gamma - \delta^f) + \delta$ the incremental construction by layering, that is, the incremental modification of γ by δ.*

The above definition describes one-layer incremental construction. By successive applications of the rule, we can construct a system with multiple layers. Besides the fusion of interactions, incremental construction can also be obtained by first combining the increments and then apply the result to the existing system. This process is called *Superposition*. Formally, we have the following definition.

Definition 9 (Superposition). *Given two increments δ_1, δ_2 over a connector γ, the operation of superposition between δ_1 and δ_2 is defined by $\delta_1 + \delta_2$.*

Superposition can be seen as a composition between increments. If we combine the superposition of increments with the layering proposed in Definition 8, then we obtain an incremental construction from a set of increments. Formally, we have the following proposition.

Proposition 1. *Let γ be a connector over B, the incremental* *construction by the superposition of n increments $\{\delta_i\}_{1\le i\le n}$ is given by*

$$(\sum_{i=1}^{n}\delta_i)\gamma = (\gamma - (\sum_{i=1}^{n}\delta_i)^f) + \sum_{i=1}^{n}\delta_i \qquad (1)$$

The above proposition provides a way to transform incremental construction by a set of increments into the separate constituents, where $\gamma - (\Sigma_{i=1}^{n}\delta_i)^f$ is the set of interactions that are allowed during the incremental construction process.

III. Invariant Preservation in Incremental Design

In Section II-B, we have presented a methodology for the incremental design of composite systems. In this section, we study the concept of *invariant preservation*. More precisely, we propose sufficient conditions to guarantee that already satisfied invariants are not violated when new interactions are added to the design.

We start by introducing the *looser synchronization preorder* on connectors, which we will use to characterize invariant preservation. As we have seen, interactions characterize the behavior of a composite component. We observe that if two interactions do not contain the same port, the execution of one interaction will not block the execution of the other interaction. Formally, we have the following definition.

Definition 10 (Conflict-free Interactions)**.** *Given a connector γ, let $a_1,\ a_2 \in \gamma$, if $a_1 \cap a_2 = \emptyset$, we say that there is no conflict between a_1 and a_2. If there is no conflict between any interactions of γ, we say that γ is conflict-free.*

We now propose a preorder relation that allows to guarantee the absence of conflicts when new interactions are added. Formally, we have the following definition.

Definition 11 (Looser Synchronization Preorder)**.** *We define the looser synchronization preorder $\preccurlyeq \subseteq 2^{2^P} \times 2^{2^P}$. For two connectors γ_1, γ_2, $\gamma_1 \preccurlyeq \gamma_2$ if for any interaction $a \in \gamma_2$, there exist interactions $b_1, \ldots, b_n \in \gamma_1$, such that $a = \bigcup_{i=1}^{n} b_i$ and there is no conflict between any b_i and b_j, where $1 \le i, j \le n$ and $i \ne j$. We simply say that γ_1 is looser than γ_2.*

The above definition requires that the stronger synchronization should be obtained by the fusion of conflict-free interactions. The reason is that the execution of interactions may be disturbed by two conflict interactions, i.e., the execution of one interaction could block the transitions issued from the other interaction. However, if we fuse them together, it means that the transitions of both interactions can be executed, which violates the constraints of the previous behavior. It is easy to see that if $\gamma_1,\ \gamma_2,\ \gamma_3,\ \gamma_4$ are connectors such that $\gamma_1 \preccurlyeq \gamma_2$, and $\gamma_3 \preccurlyeq \gamma_4$, then we have $\gamma_1 + \gamma_3 \preccurlyeq \gamma_2 + \gamma_4$.

We now propose the following proposition which establishes a link between the looser synchronization preorder and invariant preservation.

Proposition 2. *Let $\gamma_1,\ \gamma_2$ be two connectors over B. If $\gamma_1 \preccurlyeq \gamma_2$, we have $inv(\gamma_1(B), \mathcal{I}) \Rightarrow inv(\gamma_2(B), \mathcal{I})$.*

The above proposition, which will be used in the incremental design, simply says that if an invariant is satisfied, then it will remain when combinations of conflict-free interactions are added (following our incremental methodology) to the connector. This is not surprising as the tighter connector can only restrict the behaviors of the composite system.

We now switch to the more interesting problem of providing sufficient conditions to guarantee that invariants are preserved by the incremental construction.

Proposition 3. *Let γ be a connector over B and δ be an increment of γ such that $\gamma \preccurlyeq \delta$, then we have $\gamma \preccurlyeq \delta\gamma$.*

The above proposition, together with Proposition 2, says that the addition of an increment preserves the invariant if the initial connector is looser than the increment.

We continue our study and discuss the invariant preservation between the components obtained from superposition of increments and separately applying increments over the same set of components. We use the following definition.

Definition 12 (Interference-free Connectors)**.** *Given two connectors γ_1, γ_2, for any $a_1 \in \gamma_1$, $a_2 \in \gamma_2$, if either a_1 and a_2 are conflict-free or $a_1 = a_2$, we say that γ_1 and γ_2 are interference-free.*

This definition considers a relation between two connectors. We observe that two interference-free connectors will not break or block the synchronizations specified by each other. Though we require that the interactions between γ_1 and γ_2 are conflict-free, γ_1 or γ_2 respectively can contain conflict interactions. For example, consider two connectors $\gamma_1 = p_1\ p_2 + p_2\ p_3$, $\gamma_2 = p_4\ p_5$. γ_1 is not conflict-free, but γ_1 and γ_2 are interference-free.

We now present the main result of the section.

Proposition 4. *Consider two increments $\delta_1,\ \delta_2$ over γ such that $\gamma \preccurlyeq \delta_1$ and $\gamma \preccurlyeq \delta_2$, if δ_1 and δ_2 are interference-free, and $inv(\delta_1\gamma(B), \mathcal{I}_1)$, $inv(\delta_2\gamma(B), \mathcal{I}_2)$, we have $inv((\delta_1 + \delta_2)\gamma(B), \mathcal{I}_1 \wedge \mathcal{I}_2)$.*

The above proposition considers a set of increments $\{\delta_i\}_{1\le i\le n}$ over γ that are interference-free. The proposition says that if for any δ_i the separate application of increments over component $\delta_i\gamma(B)$ preserves the original invariants of $\gamma(B)$, then the system obtained from considering the superposition of increments over γ preserves the conjunction of the invariants of individual increments.

We now briefly study the relation between the looser synchronization preorder and *property preservation*. Figure 2 shows the three ingredients of the BIP language, that are (1) priorities, which we will not use here, (2) interactions, and (3) behaviors of components. We shall see that the looser synchronization preorder preserves invariants (Proposition 4). This means that the preorder preserves the so-called reachability properties. On the other hand, the preorder does not preserve deadlocks. Indeed, adding new interactions may lead to the addition of new deadlock conditions. Given two connectors γ_1 and γ_2 over component B such that γ_2 is tighter than γ_1,

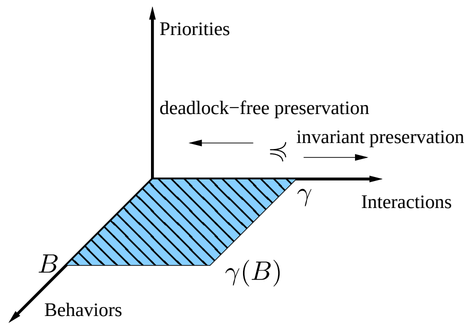
Priorities
deadlock−free preservation
$\preccurlyeq$ invariant preservation
Interactions
γ
B
$\gamma(B)$
Behaviors

Fig. 2. Invariant preservation for looser synchronization relation

i.e., $\gamma_1 \preccurlyeq \gamma_2$, we can conclude that if $\gamma_2(B)$ is deadlock-free, then $\gamma_1(B)$ is deadlock-free. However, we can still reuse the invariant of $\gamma_1(B)$ as an over-approximation of the one of $\gamma_2(B)$.

Discussion. Though we can reuse invariants to save computation time, the invariants of the system with a looser connector may be too weak with respect to a new system obtained with a tighter connector. Consider the example given in Figure 1 and let $\gamma = p_1+p_2+q_1+q_2$, $\delta_1 = p_1\ p_2$, and $\delta_2 = q_1\ q_2$. By using the technique presented in the next section, we shall see that the invariant for $\delta_1\gamma(B)$ and $\delta_2\gamma(B)$ is $(l_1 \vee l_2) \wedge (l_3 \vee l_4)$. By applying Proposition 4, we obtain that this invariant is preserved for $(\delta_1+\delta_2)\gamma(B)$. This invariant is weaker than the invariant $(l_1 \vee l_2) \wedge (l_3 \vee l_4) \wedge (l_1 \vee l_4) \wedge (l_2 \vee l_3)$ that is directly computed on $(\delta_1+\delta_2)\gamma(B)$. To overcome the above problem, we will now propose an approach that can be used to compute invariants in an incremental manner.

IV. Efficient Incremental computation of Invariants

In Section II-B, we have proposed a methodology to build a composite system by successive addition of increments. We now propose a methodology that allows to reuse existing interaction invariants when new interactions are added to the system. The section is divided in two subsections. In the first subsection, we recap the concept of *Boolean Behavioral Constraints* [3, 6], which can be used to characterize interaction invariants. In the second subsection, we propose our incremental methodology.

A. Boolean Behavioral Constraints (BBCs)

In [3], we have presented a verification method for component-based systems. The method uses a heuristic to symbolically compute invariants of a composite component. These invariants capture the interactions between components, which are the cause of global deadlocks. For this, it is sufficient to find an invariant that does not contain deadlock states. In this section, we improve the presentation of the result of [3] and prepare them for the incremental version that we will present in the next subsection.

Interactions describe the communication between different components, and transitions are the internal behavior of components. Here we unify these two types of behavioral description by introducing *Boolean Behavioral Constraints* (BBCs). We take $a_\tau = \{\{\tau_i\}_{i\in I} \mid (\forall i.\tau_i \in \mathcal{T}_i) \wedge (\{port(\tau_i)\}_{i\in I} = a)\}$. That is, a_τ consists of sets of component transitions involved in interaction a. As an example, consider the components given in Figure 1. Given $\gamma = p_1\ p_2 + q_1\ q_2$, we have $(p_1\ p_2)_\tau = \{\{\tau_1, \tau_3\}\}$, and $(q_1\ q_2)_\tau = \{\{\tau_2, \tau_4\}\}$.

Locations of components will be viewed as Boolean variables. We use $Bool[L]$ to denote the free Boolean algebra generated by the set of locations L. We also extend the notation ${}^\bullet\tau,\ \tau^\bullet$ to interactions, that is ${}^\bullet a = \{{}^\bullet\tau \mid \tau \in \mathcal{T}_i \wedge port(\tau) \in a\}$, and $a^\bullet = \{\tau^\bullet \mid \tau \in \mathcal{T}_i \wedge port(\tau) \in a\}$.

Definition 13 (Boolean Behavioral Constraints (BBCs)). *Let γ be a connector over a tuple of components $B = (B_1, \cdots, B_n)$ with $B_i = (L_i, P_i, \mathcal{T}_i)$ and $L = \bigcup_{i=1}^n L_i$. The Boolean behavioral constraints for component $\gamma(B)$ are given by the function $|\cdot| : \gamma(B) \to Bool[L]$ such that*

$$|\gamma(B)| = \bigwedge_{a\in\gamma} |a(B)|,$$
$$|a(B)| = \bigwedge_{\{\tau_i\}_{i\in I}\in a_\tau} \Big(\bigwedge_{l\in\{{}^\bullet\tau_i\}} (l \Rightarrow \bigvee_{l'\in\{\tau_i^\bullet\}} l')\Big)$$

If $\gamma = \emptyset$, then $|\gamma(B)| = true$, which means that no interactions between the components of B will be considered.

Roughly speaking, one implication $l \Rightarrow \bigvee_{l'\in\{\tau_i^\bullet\}} l'$ in $|\gamma(B)|$ describes a constraint on l that is restricted by an interaction of γ issued from l.

In what follows, we use $\bar{l}$ for the complement of l, i.e., $\neg l$.

Example 2. *Consider the components in Figure 1. Consider also the following connector $\gamma = p_1 + p_2 + q_1 + q_2$. Two increments over γ are $\delta_1 = p_1\ p_2$ and $\delta_2 = q_1\ q_2$. According to Definition 8, we have $\delta_1\gamma = p_1\ p_2 + q_1 + q_2$ when we only consider increment δ_1 over γ. For $\delta_1\gamma(B)$, the BBC $|p_1\ p_2(B)|$, $|q_1(B)|$ and $|q_2(B)|$ are respectively given by:*

$$|p_1p_2(B)| = (l_1 \Rightarrow l_2 \vee l_4) \wedge (l_3 \Rightarrow l_2 \vee l_4),$$
$$|q_1(B)| = (l_2 \Rightarrow l_1), \quad |q_2(B)| = (l_4 \Rightarrow l_3)$$

The BBC for $\delta_1\gamma(B)$ is $|\delta_1\gamma(B)| = |p_1p_2(B)| \wedge |q_1(B)| \wedge |q_2(B)| = (l_1 \Rightarrow l_2 \wedge l_4) \wedge (l_3 \Rightarrow l_2 \wedge l_4) \wedge (l_2 \Rightarrow l_1) \wedge (l_4 \Rightarrow l_3) = (\bar{l_1} \wedge \bar{l_2} \wedge \bar{l_3} \wedge \bar{l_4}) \vee (\bar{l_4} \wedge l_1 \wedge l_2) \vee (\bar{l_2} \wedge l_3 \wedge l_4) \vee (l_1 \wedge l_2 \wedge l_3) \vee (l_1 \wedge l_3 \wedge l_4)$.

When we consider two increments together, we have $(\delta_1 + \delta_2)\gamma(B) = p_1\ p_2 + q_1\ q_2$ by Definition 8 and 9. Because the BBC for interaction $q_1\ q_2$ over B is $(l_2 \Rightarrow l_1 \vee l_3) \wedge (l_4 \Rightarrow l_1 \vee l_3)$, we obtain that the BBC for $(\delta_1 + \delta_2)\gamma(B)$ is $|(\delta_1 + \delta_2)\gamma(B)| = |p_1p_2(B)| \wedge |q_1q_2(B)| = (l_1 \Rightarrow l_2 \vee l_4) \wedge (l_2 \Rightarrow l_1 \vee l_3) \wedge (l_3 \Rightarrow l_2 \vee l_4) \wedge (l_4 \Rightarrow l_1 \vee l_3) = (\bar{l_1} \wedge \bar{l_2} \wedge \bar{l_3} \wedge \bar{l_4}) \vee (l_1 \wedge l_2) \vee (l_2 \wedge l_3) \vee (l_1 \wedge l_4) \vee (l_3 \wedge l_4)$.

Example 2 shows that any BBC $|\gamma(B)|$ can be rewritten into a disjunctive normal form (DNF), where every conjunctive form is called a *monomial*. Any satisfiable monomial of $|\gamma(B)|$ is a solution of $|\gamma(B)|$. In fact, the enumeration of the clause of any monomial corresponds to an interaction invariant.

Theorem 1. *Let γ be a connector over a set of components $B = (B_1, \cdots, B_n)$ with $B_i = (L_i, P_i, \mathcal{T}_i)$ and $L = \bigcup_{i=1}^n L_i$, and $v : L \to \{true, false\}$ be a Boolean valuation different from false. If v is a solution of $|\gamma(B)|$, i.e., $|\gamma(B)|(v) = true$, then $\bigvee_{v(l)=true} l$ is an invariant of $\gamma(B)$.*

The above theorem gives a methodology to compute interaction invariants of $\gamma(B)$ directly from the solutions of $|\gamma(B)|$. In the rest of the paper, we will often use the term *BBC-invariant* to refer to the invariant that corresponds to a single solution of the BBC.

Since locations are viewed as Boolean variables, a location in a BBC is either a variable or the negation of a variable. As an example, l is a positive variable and $\neg l$ is a negative variable. However, as observed in Theorem 1, invariants are derived from positive variables of the solution of $|\gamma(B)|$. This suggests that all the negations should be removed. In general, due to incremental design and implementation (see Proposition 6 and Section V), these valuations can be removed gradually. We now propose a general mapping on removing variables with negations that do not belong to a given set of variables.

Definition 14 (Positive Mapping)**.** *Given two sets of variables L and L' such that $L' \subseteq L$, we define a mapping $p(L')$ over a disjunctive normal form formula that removes all the variables not in L' and with negations from the formula, such that*

$$(\bigwedge_{l_i\in L} l_i \wedge \bigwedge_{l_j\in L'} \bar{l_j} \wedge \bigwedge_{l_k\in L-L'} \bar{l_k})^{p(L')} = \bigwedge_{l_i\in L} l_i \wedge \bigwedge_{l_j\in L'} \bar{l_j}$$
$$(f_1 \vee f_2)^{p(L')} = f_1^{p(L')} \vee f_2^{p(L')}$$

where f_1 and f_2 are in disjunctive normal form.

If L' is empty, then the positive mapping will remove all the negations from a DNF formula f, which we will denote by f^p. Notice that $(\bigwedge_{i\in I} \bar{l_i})^p = false$.

We are now ready to propose an interaction invariant that takes all the solutions of the BBCs into account. We first introduce the notation $\widetilde{f}$ that stands for the dual of f, by replacing the AND operators with ORs (and vice versa) and the constant 0 with 1 (and vice versa). As we have seen, BBCs can be rewritten as a disjunction of monomials. By dualizing a monomial, one can obtain an interaction invariant. If one wants the strongest invariant that takes all the solution into account, one simply has to dualize the BBC. This is stated with the following theorem.

Theorem 2. *For any connector γ applied to a tuple of components $B = (B_1, \cdots, B_n)$, the interaction invariant of $\gamma(B)$ can be obtained as the dual of $|\gamma(B)|^p$, denoted by $\widetilde{|\gamma(B)|^p}$.*

Example 3. *We consider the components, connectors, and BBCs introduced in Example 2. The positive mapping removes variables with negations from $|\delta_1\gamma(B)|$ and $|(\delta_1+\delta_2)\gamma(B)|$. We obtain that $\widetilde{|\delta_1\gamma(B)|^p} = (l_1 \vee l_2) \wedge (l_3 \vee l_4)$, and $\widetilde{|(\delta_1+\delta_2)\gamma(B)|^p} = (l_1\vee l_2)\wedge(l_3\vee l_4)\wedge(l_1\vee l_4)\wedge(l_2\vee l_3)$. If we specify $Init = l_1\wedge l_3$, every invariant of system $\langle\delta_1\gamma(B), Init\rangle$ and $\langle(\delta_1+\delta_2)\gamma(B), Init\rangle$ should contain either l_1 or l_3. Therefore $(l_1 \vee l_2) \wedge (l_3 \vee l_4)$ is the interaction invariant of $\langle\delta_1\gamma(B), Init\rangle$, and $(l_1 \vee l_2) \wedge (l_3 \vee l_4) \wedge (l_1 \vee l_4) \wedge (l_2 \vee l_3)$ is the interaction invariant of $\langle(\delta_1+\delta_2)\gamma(B), Init\rangle$.*

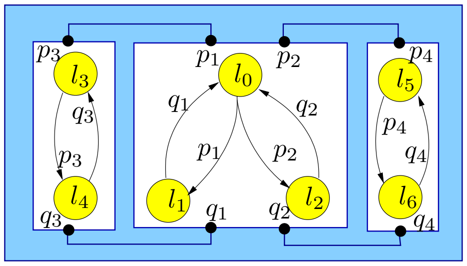

Fig. 3. An example for incremental computation of invariants

B. Incremental Computation of BBCs

In the previous section, we have shown that interaction invariants can be computed from the solutions of Boolean Behavioral Constraints. In this section, we show how to reuse existing invariants when new increments are added to the system. We first give a decomposition form for BBC and then show how this decomposition can be used to save computation time.

Proposition 5. *Let γ be a connector over B, the Boolean behavioral constraint for the composite component obtained by superposition of n increments $\{\delta_i\}_{1\leq i\leq n}$ can be written as*

$$|(\sum_{i=1}^{n}\delta_i)\gamma(B)| = |(\gamma - (\sum_{i=1}^{n}\delta_i)^f)(B)| \wedge \bigwedge_{i=1}^{n}|\delta_i(B)| \quad (2)$$

Proposition 5 provides a way to decompose the computation of BBCs with respect to increments. The decomposition is based on the fact that different increments describe the interactions between different components. To simplify the notation, $\gamma - (\Sigma_{i=1}^n\delta_i)^f$ is represented by δ_0. We have the following example.

Example 4. *[Incremental BBC computation] In the example of Figure 3, let $\gamma = p_1 + p_2 + p_3 + p_4 + q_1 + q_2 + q_3 + q_4$. Two increments over γ are $\delta_1 = p_1\ p_3 + q_1\ q_3$ and $\delta_2 = p_2\ p_4 + q_2\ q_4$. The new connector obtained by applying δ_1 and δ_2 to γ is given by $(\delta_1+\delta_2)\gamma = p_1\ p_3 + q_1\ q_3 + p_2\ p_4 + q_2\ q_4$. The BBC $|\delta_1(B)|$ and $|\delta_2(B)|$ are respectively given by:*

$$\begin{aligned}|\delta_1(B)| = \quad & (l_0 \Rightarrow l_1 \vee l_4) \wedge (l_1 \Rightarrow l_0 \vee l_3)\wedge \\ & (l_3 \Rightarrow l_1 \vee l_4) \wedge (l_4 \Rightarrow l_0 \vee l_3), \\ |\delta_2(B)| = \quad & (l_0 \Rightarrow l_2 \vee l_6) \wedge (l_2 \Rightarrow l_0 \vee l_5)\wedge \\ & (l_5 \Rightarrow l_2 \vee l_6) \wedge (l_6 \Rightarrow l_0 \vee l_5)\end{aligned}$$

Since $\gamma-(\delta_1+\delta_2)^f = \emptyset$, we have $|(\delta_1+\delta_2)\gamma(B)| = |\delta_1(B)|\wedge|\delta_2(B)|$.

We now switch to the problem of computing invariants while taking incremental design into account. We propose the following definition that will help in the process of reusing existing invariants.

Definition 15 (Common Location Variables L_c)**.** *The set of common location variables of a set of connectors $\{\gamma_i\}_{1\leq i\leq n}$ is defined by $L_c = \bigcup_{i,j\in[1,n]\wedge i\neq j} support(\gamma_i) \cap support(\gamma_j)$, where $support(\gamma) = \bigcup_{a\in\gamma} {}^{\bullet}a \cup a^{\bullet}$, the set of locations involved in some interaction a of γ.*

Our incremental method assumes that an invariant has already been computed for a set of interactions (We use $\mathcal{I}_\delta$

to denote the BBC-invariant of $|\delta(B)|$). This information is exploited to improve the efficiency. The idea is as follows. According to Equation 1, the superposition of a set of increments $\{\delta_i\}_{1\leq i\leq n}$ over a connector γ can be regarded as separately applying increments over theirs constituents. We propose the following proposition, which builds on Equation 2.

Proposition 6. *Consider a composite component B. Let γ be a connector for B and assume a set of increments $\{\delta_i\}_{1\leq i\leq n}$ over $\gamma(B)$. Let $\delta_0 = \gamma - (\sum_{i=1}^{n} \delta_i)^f$, $\mathcal{I}_{\delta_i} = \bigwedge_{k\in I_i} \phi_k$, for $i = 0, \ldots, n$, be the BBC-invariants for each $|\delta_i(B)|$, $S_{\delta_i} = \bigvee_{k\in I_i} m_k$, for $i = 0, \ldots, n$, be the corresponding BBC-solutions, and let*

- *L_ϕ be the set of location variables in invariant ϕ,*
- *L_c be the common location variables between $\{\delta_0, \delta_1, \ldots, \delta_n\}$.*

Then the interaction invariant of $(\Sigma_{i=1}^{n}\delta_i)\gamma(B)$ is obtained as follows:

$$\mathcal{I} = \left(\bigwedge_{i=0}^{n} \bigwedge_{\substack{k\in I_i \wedge \\ L_c\cap L_{\phi_k}=\emptyset}} \phi_k\right) \wedge \left(\bigwedge_{(k_{i1},\ldots,k_{ir})\in\mathbb{D}} \bigvee_{j=1}^{r} \phi_{k_{ij}}\right)$$

where
$\mathbb{D} = \{(k_{i1}, \ldots, k_{ir})|\ (\forall j = 1 \ldots r \wedge k_{ij} \in I_{ij}) \wedge (L_{\phi_{k_{ij}}} \cap L_c \neq \emptyset) \wedge (\bigwedge_{j=1}^{r} m_{k_{ij}} \neq false) \wedge ((k_{i1}, \ldots, k_{ir})$ *is maximal*$)\}$.

The proposition simply says that one can take the conjunctions of BBC-invariants that do not share common variables, while one has to take the disjunction of the remaining invariants. This is to guarantee that common location variables will not change the satisfiability of the formulae. Observe that each non common variable occurs only in the solutions of one BBC. This allows deleting the non common variables with negations separately by using the positive mapping of common variables in every BBC-solutions, which reduces complexity of computation significantly.

Example 5. *[Incremental invariant computation] In Example 4, we have computed the BBCs for the two increments. Here we show how to compute the invariants from BBC-invariants of the increments. By Definition 15, we obtain that $L_c = \{l_0\}$. Let S_{δ_1}, S_{δ_2} be the BBC-solutions for $|\delta_1(B)|$ and $|\delta_2(B)|$ respectively, and $\mathcal{I}_{\delta_1}, \mathcal{I}_{\delta_2}$ be their BBC-invariants, we have:*
$S_{\delta_1} = (\bar{l_0} \wedge \bar{l_1} \wedge \bar{l_3} \wedge \bar{l_4}) \vee (l_0 \wedge l_1) \vee (l_1 \wedge l_3) \vee (l_0 \wedge l_4) \vee (l_3 \wedge l_4)$,
$S_{\delta_2} = (\bar{l_0} \wedge \bar{l_2} \wedge \bar{l_5} \wedge \bar{l_6}) \vee (l_0 \wedge l_2) \vee (l_2 \wedge l_5) \vee (l_0 \wedge l_6) \vee (l_5 \wedge l_6)$,
$\mathcal{I}_{\delta_1} = (l_0 \vee l_1) \wedge (l_0 \vee l_4) \wedge (l_1 \vee l_3) \wedge (l_3 \vee l_4)$,
$\mathcal{I}_{\delta_2} = (l_0 \vee l_2) \wedge (l_0 \vee l_6) \wedge (l_2 \vee l_5) \wedge (l_5 \vee l_6)$
Because $\mathcal{I}_{(\delta_1+\delta_2)\gamma(B)} = \mathcal{I}_{((\gamma-(\delta_1+\delta_2)^f)+\delta_1+\delta_2)(B)}$ and $\gamma - (\delta_1 + \delta_2)^f = \emptyset$, we have $\mathcal{I}_{(\delta_1+\delta_2)\gamma(B)} = \mathcal{I}_{(\delta_1+\delta_2)(B)}$.

Among the BBC-invariants, $(l_1 \vee l_3), (l_3 \vee l_4), (l_2 \vee l_5), (l_5 \vee l_6)$ do not contain any common location variables, so they will remain in the global computation. BBC-invariants $(l_0 \vee l_1), (l_0 \vee l_4)$, $(l_0 \vee l_2)$ and $(l_0 \vee l_6)$ contain l_0 as the common location variable, and the conjunction between every monomial from two groups of solutions are not false. So the final result is $(l_0 \vee l_1 \vee l_2) \wedge (l_0 \vee l_4 \vee l_6) \wedge (l_0 \vee l_1 \vee l_6) \wedge (l_0 \vee l_2 \vee l_4) \wedge (l_1 \vee l_3) \wedge (l_3 \vee l_4) \wedge (l_2 \vee l_5) \wedge (l_5 \vee l_6)$.

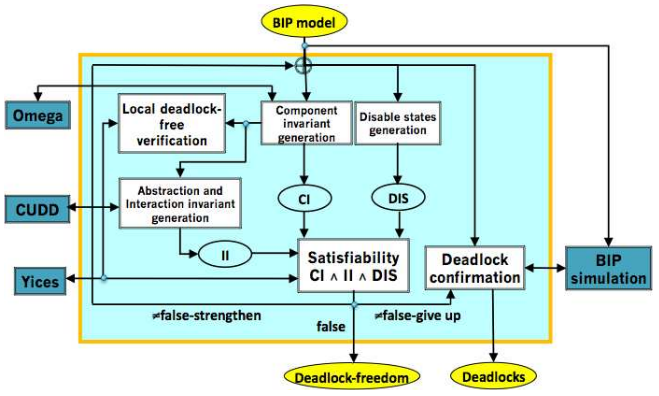
BIP model
Omega
Local deadlock-free verification
Component invariant generation
Disable states generation
CUDD
Abstraction and Interaction invariant generation
CI
DIS
II
Yices
Satisfiability CI ∧ II ∧ DIS
Deadlock confirmation
BIP simulation
≠false-strengthen
false
≠false-give up
Deadlock-freedom
Deadlocks

Fig. 4. D-Finder tool

V. Experiments

Our methodology for computing interaction invariants and deciding invariant preservation has been implemented in the D-Finder toolset [2].

In this section, we start with a brief introduction to the the D-Finder tool and explain what are the modifications that have. Then we show the experimental results obtained by implementing the methods discussed in this paper.

A. D-Finder Structure

D-Finder is an extension of the BIP toolset [7] – BIP can be used to define components and component interactions. D-Finder can verify both safety and deadlock-freedom properties of systems by using the techniques of this paper and of [3, 6].

We use $global$ to refer to the method of [3], FP for the incremental method of [6], and $Incr$ to refer to our new incremental technique.

The tool provides symbolic-representations-based methods for computing interaction invariants, namely the $Incr$ methods presented in this paper, the fixed point based method and its incremental method FP proposed in [6] as well as the $global$ method presented in [3] and discussed in Section II. D-Finder relies on the CUDD package [15] and represents sets of locations by BDDs. D-Finder also proposes techniques to compute component invariants. Those techniques, which are described in [3], relies on the Yices [11] and Omega [16] toolsets for the cases in where a component can manipulate data. A general overview of the structure of the tool is given in Figure 4.

D-Finder is mainly used to check safety properties of composite components. In this paper, we will be concerned with the verification of deadlock properites. We let DIS be the set of global states in where a deadlock can occur. The tool will progressively find and eliminate potential deadlocks as follows. D-Finder starts with an input a BIP model and computes component invariants CI by using the technique outlined in [3]. 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[1].

B. Implementation of the Incremental Method

We build on the symbolic implementation of the method in [3] that computes the interaction invariant of an entire system with all the interactions within the connector. The implementation relies on the CUDD package [15] and represents sets of locations by BDDs.

We have employed the following steps to integrate the incremental computation into the D-Finder tool. First we compute a set of common location variables from all the increments. Then we compute the BBC-solutions for every increment instead of computing the solutions for the connector in $global$ method, and apply positive mapping to remove the location variables with negations that do not belong to the set of common location variables, to reduce the size of BDDs for BBC-solutions. We can either integrate existing solutions from the already computed BBCs progressively or integrate all the solutions when all the increments have been explored. Finally we apply positive mapping to remove all the remaining common location variables with negations and call the dual operation to obtain interaction invariant.

C. Experimental Results

We have compared the performance of the three methods on several case studies. All our experiments have been conducted with a 2.4GHz Duo CPU Mac laptop with 2GB of RAM.

We started by considering verification of deadlock properties. The case studies we consider are the Gas Station [12], the Smoker [13], the Automatic Teller Machine (ATM) [8] and the classical example of Producer/Consumer. Regarding the Gas Station example, we assume that every pump has 10 customers. Hence, if there are 50 pumps in a Gas Station, then we have 500 customers and the number of components including the operator is thus 551. In the ATM example, every ATM machine is associated to one user. Therefore, if we have 10 machines, then the number of components will be 22 (including the two components that describe the Bank). The computation times and memory usages for the application of the three methods on these case studies are given in Table I. Regarding the legend of the table, $scale$ is the "size" of examples; $location$ denotes the total number of control locations; $interaction$ is for the total number of interactions. The computation time is given in minutes. The timeout, i.e., "-" is one hour. The memory usage is given in Megabyte (MB). Our technique is always faster than $global$. This means that we are also faster than tools such as NuSMV and SPIN that are known to be much slower than $global$ on these case studies [3, 2]. Our $Incr$ technique is faster than FP except for the Gas Station and it always consumes less memory.

[1] D-Finder is also connected to the state-space exploration tool of the BIP platform, for finer analysis when the heuristic fails to prove deadlock-freedom.

TABLE I
COMPARISON FOR ACYCLIC TOPOLOGIES.

Component information			Time (minutes)			Memory (MB)		
scale	location	interaction	$global$	FP	$Incr$	$global$	FP	$Incr$
Gas Station								
50 pumps	2152	2000	0:50	0:17	0:49	48	53	47
100 pumps	4302	4000	2:58	0:52	1:51	76	52	47
200 pumps	8602	8000	11:34	1:55	2:26	135	65	47
400 pumps	17202	16000	47:38	3:51	5:43	270	93	76
500 pumps	21502	20000	-	4:43	7:21	-	101	86
600 pumps	25802	24000	-	5:53	9:05	-	115	97
700 pumps	30102	28000	-	7:14	11:44	-	138	107
Smoker								
300 smokers	907	903	0:07	0:07	0:07	44	11	7
600 smokers	1807	1803	0:13	0:14	0:13	46	26	8
1500 smokers	4507	4503	1:38	0:44	0:34	65	54	18
3000 smokers	9007	9003	6:21	1:57	1:14	113	86	28
6000 smokers	18007	18003	27:03	5:57	3:24	222	172	55
7500 smokers	22507	22503	41:38	8:29	4:51	270	209	60
9000 smokers	27007	27003	-	11:36	6:34	319	247	96
ATM								
50 machines	1104	902	10:49	2:20	1:23	81	86	22
100 machines	2204	1802	43:00	6:00	1:57	142	271	44
250 machines	5504	4002	-	17:16	4:46	-	670	65
350 machines	7704	6302	-	27:54	8:18	-	938	77
600 machines	13204	10802	-	-	24:14	-	-	119
Producer/Consumer								
2000 consumers	4004	4003	0:27	0:33	0:31	57	16	11
4000 consumers	8004	8003	1:27	1:18	1:05	90	28	20
6000 consumers	12004	12003	3:01	2:32	2:03	126	37	31
8000 consumers	16004	16003	5:35	4:22	2:33	164	40	35
10000 consumers	20004	20003	8:44	6:12	3:15	218	66	56
12000 consumers	24004	24003	12:06	8:37	5:38	257	75	66

TABLE II
COMPARISON BETWEEN DIFFERENT METHODS ON DINING PHILOSOPHERS

Component information			Time (minutes)			Memory (MB)		
scale	location	interaction	$global$	FP	$Incr$	$global$	FP	$Incr$
500 philos	3000	2500	4:01	9:18	0:34	61	60	29
1000 philos	6000	5000	17:09	-	2:04	105	-	60
1500 philos	9000	7500	39:40	-	3:09	148	-	74
2000 philos	12000	10000	-	-	4:14	-	-	96
4000 philos	24000	20000	-	-	8:37	-	-	192
6000 philos	36000	30000	-	-	14:26	-	-	382
9000 philos	53000	45000	-	-	24:16	-	-	581

In Table II, we also provide results on checking deadlock-freedom for the dining philosopher algorithm. Contrary to the above examples, the dining philosopher algorithm has a cyclic topology, which cannot be efficiently managed with FP (this is the only case for which $global$ was faster than FP.

Our results have also been applied on a complex case study that directly comes from an industrial application. More precisely, we have been capable of checking safety and deadlock-freedom properties on the modules in the functional level of the *DALA robot* [5]. DALA is an autonomous robot with modules described in the BIP language running at the functional level. Every module is in a hierarchy of composite components.

All together the embedded code of DALA in the functional level contains more than 500 000 lines of C code. The topology of the modules and the description of the behaviors of the components are complex. This is beyond the scope of tools such as NuSMV or SPIN. We first checked deadlock properties of individual modules. Both $global$ and FP fail to check for deadlock-freedom (Antenna is the only module that can be checked by using $global$). However, by using $Incr$, we can always generate the invariants and check the deadlock-freedom of all the modules. Table III shows the time consumption in computing invariants for deadlock-freedom checking of seven modules by the incremental method; it also gives the number of states per module. In these modules we have successively detected (and corrected) two deadlocks within Antenna and

TABLE III
DEADLOCK-FREEDOM CHECKING ON DALA BY $Incr$ METHOD

module	component	location	interaction	states	time (minutes)
SICK	43	213	202	$2^{20} \times 3^{29} \times 34$	1:22
Aspect	29	160	117	$2^{17} \times 3^{23}$	0:39
NDD	27	152	117	$2^{22} \times 3^{14} \times 5$	8:16
RFLEX	56	308	227	$2^{34} \times 3^{35} \times 1045$	9:39
Battery	30	176	138	$2^{22} \times 3^{17} \times 5$	0:26
Heating	26	149	116	$2^{17} \times 3^{14} \times 145$	0:17
Platine	37	174	151	$2^{19} \times 3^{22} \times 35$	0:59

NDD, respectively.

Aside from the deadlock-freedom requirement, some modules also have safety property requirements such as causality (a service can be triggered only after a certain service has been running successfully, i.e., only if the variable corresponding to this service is set to true). In checking the causality requirement between different services, we need to compute invariants according to different causality requirement. Inspired from the invariant preservation properties introduced in Section III, we removed some tight synchronizations between some components[2] that would not synchronize directly with the components involved in the property and obtained a module with looser synchronized interactions. As the invariant of the module with looser synchronizations is preserved by the one with tighter synchronizations, if a property is satisfied in the former, then it is satisfied in the latter. Based on this fact, we could obtain the satisfied causality property in 17 seconds, while it took 1003 seconds before using the preorder. A more detailed description of DALA and other properties verified with our $Incr$ and invariant preservation methods can be found in [4].

VI. CONCLUSION

We present new incremental techniques for computing interaction invariants of composite systems defined in the BIP framework. In addition, we propose sufficient conditions that guarantee invariant preservation when new interactions are added to the system. Our techniques have been implemented in the D-Finder toolset and have been applied to complex case studies that are beyond the scope of existing tools.

As we have seen in Section V, our new techniques and the ones in [3, 6] are complementary. As a future work, we plan to set up a series of new experiments to give a deeper comparison between these techniques. This should help the user to select the technique to be used depending on the case study. Other future works include to extend our contribution to liveness properties and abstraction.

Acknowledgment. We are grateful to the reviewers for their careful work and their valuable and insightful comments and suggestions.

REFERENCES

[1] A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in BIP. In *Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods*, pages 3–12, Washington, DC, USA, 2006. IEEE Computer Society.

[2] S. Bensalem, M. Bozga, T.-H. Nguyen, and J. Sifakis. D-Finder: A tool for compositional deadlock detection and verification. In *Proceedings of the 21st International Conference on Computer Aided Verification*, pages 614–619, Berlin, Heidelberg, 2009. Springer-Verlag.

[3] S. Bensalem, M. Bozga, J. Sifakis, and T.-H. Nguyen. Compositional verification for component-based systems and application. In *Proceedings of the 6th International Symposium on Automated Technology for Verification and Analysis*, pages 64–79, Berlin, Heidelberg, 2008. Springer-Verlag.

[4] S. Bensalem, L. de Silva, M. Gallien, F. Ingrand, and R. Yan. “Rock solid” software: A verifiable and correct by construction controller for rover and spacecraft functional layers. In *Proceedings of the 10th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2010.

[5] S. Bensalem, M. Gallien, F. Ingrand, I. Kahloul, and T.-H. Nguyen. Toward a more dependable software architecture for autonomous robots. *IEEE Robotics and Automation Magazine*, 16(1):1–11, 2009.

[6] S. Bensalem, A. Legay, T.-H. Nguyen, J. Sifakis, and R. Yan. Incremental invariant generation for compositional design. In *Proceedings of the 4th IEEE International Symposium on Theoretical Aspects of Software Engineering*, 2010.

[7] BIP. http://www-verimag.imag.fr/BIP,196.html?

[8] M. R. V. Chaudron, E. M. Eskenazi, A. V. Fioukov, and D. K. Hammer. A framework for formal component-based software architecting. In *Proceedings of Specification and Verification of Component-Based Systems Workshop*, pages 73–80, 2001.

[9] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2:410–425, 2000.

[10] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. The MIT Press, 1999.

[11] B. Dutertre and L. de Moura. A fast linear-arithmetic solver for DPLL(T). In *Proceedings of the 18th Computer-Aided Verification conference*, volume 4144 of *LNCS*, pages 81–94. Springer-Verlag, 2006.

[12] D. Heimbold and D. Luckham. Debugging Ada tasking programs. *IEEE Softw.*, 2(2):47–57, 1985.

[13] S. S. Patil. *Limitations and Capabilities of Dijkstra's Semaphore Primitives for Coordination among Processes*. Cambridge, Mass.: MIT, Project MAC, Computation Structures Group Memo 57, Feb, 1971.

[14] J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proceedings of the 5th Colloquium on International Symposium on Programming*, pages 337–351. Springer-Verlag, 1982.

[15] F. Somenzi. CUDD: CU decision diagram package.

[16] O. Team. The Omega library, 1996.

[2]The latter can be seen as an abstraction of the component in where some services have been removed.