Incremental Component-based Construction and Verification using Invariants

Saddek Bensalem\textsuperscript{1}  Marius Bozga\textsuperscript{1}  Axel Legay\textsuperscript{2}  Thanh-Hung Nguyen\textsuperscript{1}  Joseph Sifakis\textsuperscript{1}  Rongjie Yan\textsuperscript{1}
\textsuperscript{1} Verimag Laboratory, Université Joseph Fourier Grenoble, CNRS  
\textsuperscript{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 \textsuperscript{10,\textsuperscript{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 resulting from mutually interacting components. A compositional verification method based on invariant computation is presented in \textsuperscript{3,\textsuperscript{2}}. This approach is based on the following rule:

\[
\{B_i < \Phi_i >\}_{i=1} \parallel \Psi \in I\{\{B_i\}_{i=1}, \{\Phi_i\}_{i=1}\}, (\Lambda_{i} \Phi_i) \land \Psi \Rightarrow \Phi
\]

The rule allows to prove invariance of property $\Phi$ for systems obtained by using an $n$-ary composition operation $\parallel$ parameterized by a set of interactions $\gamma$. It uses global invariants that are the conjunction of individual invariants $\Phi_i$ of individual components $B_i$ and an interaction invariant $\Psi$. The latter expresses constraints on the global state space induced by interactions between components. In \textsuperscript{3}, we have shown that $\Psi$ can be computed automatically from abstractions of the system to be verified. These are the composition of finite state abstractions $B_i^\gamma$ of the components $B_i$ with respect to their invariants $\Phi_i$. The approach has been implemented in the D-Finder toolset \textsuperscript{2} and applied to check deadlock-freedom on several case studies described in the BIP (Behavior, Interaction, Priority) \textsuperscript{11} language. The results of these experiments show that D-Finder is exponentially faster than well-established tools such as NuSMV \textsuperscript{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 \textsuperscript{3}, do not focus on such aspects. In a very recent work \textsuperscript{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 \textsuperscript{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
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 50000 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 [3] 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 $B = (L, P, T)$, where:

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

Given $\tau = (l, p, l') \in T$, $l$ and $l'$ are the source and destination locations, respectively. In the rest of the paper, we use $*\tau$ and $\tau^*$ to compute the source and destination of $\tau$, 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 $B_i = (L_i, P_i, T_i)$, 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 $\{a_1, \ldots, a_m\}$. An example, notation for the connector $\{\{p_1, p_2\}, \{q_1, q_2\}\}$ is $p_1 + q_1 q_2$.

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 $B_i = (L_i, P_i, T_i)$ and a connector $\gamma$, we define the parallel composition $B = \gamma(B_1, \ldots, B_n)$ as the transition system $(L, \gamma, T)$, where:

- $L = L_1 \times L_2 \times \ldots \times L_n$ is the set of global locations,
- $\gamma$ is a set of interactions, and
- $T \subseteq L \times \gamma \times L$ contains all transitions $\tau = ((l_1, \ldots, l_n), (a, (l'_1, \ldots, l'_n)))$ obtained by synchronization of sets of transitions $\{\tau_i = (l_i, p_i, l'_i) \in T_i\}_{i \in I}$ 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 $γ(B_1, B_2)$, where $γ = p_1 + p_2 + q_1 + q_2$. Observe that the component $γ_1(B_1, . . . , B_n)$, which is obtained by applying the connector $γ_1 = \sum_{i=1}^n (\sum_{j\in P_i} p_j)$, is the transition system obtained by interleaving the transitions of atomic components. Observe also that the parallel composition $γ(B_1, . . . , B_n)$ of $B_1, . . . , B_n$ can be seen as a l-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 $T$. In the rest of the paper, $L$ will be called the set of locations of $B$, while $L$ 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 $S$ is a pair $(B, \text{Init})$ 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 $S = (B, \text{Init})$ can be obtained from those of $B$ according to the constraint $\text{Init}$. Therefore we define invariants for a component and for a system separately.

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

Clearly, if $I_1, I_2$ are invariants of $B$ (respectively $S$) then $I_1 \land I_2$ and $I_1 \lor I_2$ are also invariants of $B$ (respectively $S$).

Let $γ(B_1, . . . , B_n)$ be the composition of $n$ components with $B_i = (L_i, P_i, T_i)$ for $i = 1, . . . , n$. In the paper, an invariant on $B_i$ is called a component invariant and an invariant on $γ(B_1, . . . , B_n)$ 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 2, 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 $γ^c = \{a \neq \emptyset | \exists b \in γ, a \subseteq b\}$.

- The forbidden interactions $γ^f$ of $γ$ is the set of the interactions strictly contained in all the interactions of $γ$. That is $γ^f = γ^c - γ$. It is easy to see that for two connectors $γ_1$ and $γ_2$, we have $(γ_1 + γ_2)^c = γ_1^c + γ_2^c$ and $(γ_1 + γ_2)^f = (γ_1 + γ_2)^c - γ_1 - γ_2$.

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 $δ \subseteq 2^B$ be a set of interactions. We say $δ$ is an increment over $γ$ if for any interaction $a \in δ$ we have interactions $b_1, . . . , b_n \in γ$ 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 $γ$ over $B$ 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 $δγ = (γ - δ^f) + δ$ 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 $δ_1 + δ_2$.

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
constructions by the superposition of $n$ increments $\{\delta_i\}_{1 \leq i \leq n}$ is given by

$$\left(\sum_{i=1}^{n} \delta_i\right)\gamma = (\gamma - \left(\sum_{i=1}^{n} \delta_i\right)) + \sum_{i=1}^{n} \delta_i$$  \hspace{1cm} (1)$$

The above proposition provides a way to transform incremental construction by a set of increments into the separate constituents, where $\gamma - \left(\sum_{i=1}^{n} \delta_i\right)$ 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 $\gamma$, 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 $\gamma$, we say that $\gamma$ 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 $\preceq\subseteq 2^B \times 2^B$. For two connectors $\gamma_1, \gamma_2$, $\gamma_1 \preceq \gamma_2$ 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 $1 \leq i, j \leq n$ and $i \neq j$. We simply say that $\gamma_1$ is looser than $\gamma_2$.

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 $\gamma_1 \preceq \gamma_2$, and $\gamma_3 \preceq \gamma_4$, then we have $\gamma_1 + \gamma_3 \preceq \gamma_2 + \gamma_4$.

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 $\gamma_1 \preceq \gamma_2$, we have $\text{inv}(\gamma_1(B), I) \Rightarrow \text{inv}(\gamma_2(B), I)$.

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 $\gamma$ be a connector over $B$ and $\delta$ be an increment of $\gamma$ such that $\gamma \preceq \delta$, then we have $\gamma \preceq \delta\gamma$.

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 $\gamma_1, \gamma_2$, 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 $\gamma_1$ and $\gamma_2$ 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 $\gamma_1$ and $\gamma_2$ are conflict-free, $\gamma_1$ or $\gamma_2$ respectively can contain conflict interactions. For example, consider two connectors $\gamma_1 = p_1 p_2 + p_2 p_3, \gamma_2 = p_4 p_5$. $\gamma_1$ is not conflict-free, but $\gamma_1$ and $\gamma_2$ are interference-free.

We now present the main result of the section.

**Proposition 4.** Consider two increments $\delta_1, \delta_2$ over $\gamma$ such that $\gamma \preceq \delta_1$ and $\gamma \preceq \delta_2$, if $\delta_1$ and $\delta_2$ are interference-free, and $\text{inv}(\delta_1\gamma(B), I_1)$, $\text{inv}(\delta_2\gamma(B), I_2)$, we have $\text{inv}(\delta_1 + \delta_2)\gamma(B), I_1 \land I_2)$.

The above proposition considers a set of increments $\{\delta_i\}_{1 \leq i \leq n}$ over $\gamma$ that are interference-free. The proposition says that if for any $\delta_i$ 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 $\gamma$ 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 $\gamma_1$ and $\gamma_2$ over component $B$ such that $\gamma_2$ is tighter than $\gamma_1$, we have $\gamma_1 \preceq \gamma_2$.
We take a_τ consists of sets of component transitions involved in interaction a. As an example, consider the components given in Figure 1. Given γ = p_1 + p_2 + q_1 + q_2, we have (p_1 p_2)_τ = \{ (τ₁, τ₃) \} and (q_1 q_2)_τ = \{ (τ₂, τ₄) \}.

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 *τ to interactions, that is, a_τ = \{ τ | τ ∈ T_i ∧ port(τ) ∈ a \}, and a_σ = \{ σ | σ ∈ T_i ∧ port(σ) ∈ a \}.

Definition 13 (Boolean Behavioral Constraints (BBCs)).
Let γ be a connector over a tuple of components B = (B₁, · · · , B_n) with B_i = (L_i, P_i, T_i) and L = \bigcup_{i=1}^{n} L_i. The Boolean behavioral constraints for component γ(B) are given by the function | · | : γ(B) → Bool[L] such that

|γ(B)| = \bigwedge_{a ∈ γ} | a(B)|,

|a(B)| = \bigwedge_{τ ∈ T_i ∧ port(τ) ∈ a} \bigwedge_{τ'} \big( l ⇒ \bigvee_{τ'} \big)\)

If γ = 0, then |γ(B)| = true, which means that no interactions between the components of B will be considered.

Roughly speaking, one implication l ⇒ \bigvee_{τ ∈ T_i} \big)\big) in |γ(B)| describes a constraint on l that is restricted by an interaction of γ issued from l.

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

Example 2. Consider the components in Figure 7. Consider also the following connector γ = p_1 + p_2 + q_1 + q_2. Two increments over γ are δ_1 = p_1 p_2 and δ_2 = q_1 q_2. According to Definition 5, we have δ_1 γ(B) = p_1 p_2 + q_1 + q_2 when we only consider increment δ_1 over γ. For δ_1 γ(B), the BBC |p₁ p₂(B)|, |q₁(B)| and |q₂(B)| are respectively given by:

|p₁ p₂(B)| = (l₁ ⇒ l₂ ∨ l₄) ∧ (l₃ ⇒ l₂ ∨ l₄),

|q₁(B)| = (l₂ ⇒ l₁),

|q₂(B)| = (l₄ ⇒ l₃).

The BBC for δ₁ γ(B) is |δ₁ γ(B)| = |p₁ p₂(B)| ∧ |q₁(B)| ∧ |q₂(B)| = (l₁ ⇒ l₂ ∨ l₄) ∧ (l₃ ⇒ l₂ ∨ l₄) ∧ (l₂ ⇒ l₁) ∧ (l₄ ⇒ l₃) = (l₁ ∨ l₂ ∨ l₄ ∨ l₅) ∧ (l₃ ∨ l₁ ∨ l₂) ∧ (l₂ ∨ l₃ ∨ l₄) ∧ (l₁ ∨ l₂ ∨ l₃ ∧ l₄).

When we consider two increments together, we have (δ₁ + δ₂) γ(B) = p_1 p_2 + q_1 q_2 by Definition 5 and 9. Because the BBC for interaction q_1 q_2 over B is (l₂ ⇒ l₁ ∨ l₄) ∧ (l₄ ⇒ l₁ ∨ l₃), we obtain that the BBC for (δ₁ + δ₂) γ(B) is |(δ₁ + δ₂) γ(B)| = |p₁ p₂(B)| ∧ |q₁ q₂(B)| = (l₁ ⇒ l₂ ∨ l₄) ∧ (l₂ ⇒ l₁ ∨ l₃) ∧ (l₃ ⇒ l₂ ∨ l₄) ∧ (l₄ ⇒ l₁ ∨ l₃) = (l₁ ∨ l₂ ∨ l₃ ∨ l₄) ∧ (l₂ ∨ l₁ ∨ l₃) ∧ (l₃ ∨ l₂ ∨ l₄) ∧ (l₄ ∨ l₁ ∨ l₃).

Example 2 shows that any BBC |γ(B)| can be rewritten into a disjunctive normal form (DNF), where every conjunctive form is called a monomial. Any satisfiable monomial of |γ(B)| is a solution of |γ(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 = (B₁, · · · , B_n) with B_i = (L_i, P_i, T_i) and L = \bigcup_{i=1}^{n} L_i, and v : L → {true, false} be a Boolean valuation different from false. If v is a solution of |γ(B)|, i.e., |γ(B)|(v) = true, then \bigvee_{v(l)=true} l is an invariant of γ(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 \( \text{4} \), 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 \( \text{6} \) and Section \( \text{3} \)), 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

\[
(\bigwedge_{l_i \in L} l_i \land \bigwedge_{l_j \in L'} \neg l_j \land \bigwedge_{l_k \in L \land L'} \neg l_k)^{p(L')} = \bigwedge_{l_i \in L} l_i \land \bigwedge_{l_j \in L'} l_j \land (f_1 \lor f_2)^{p(L')} = f_1^{p(L')} \lor f_2^{p(L')}
\]

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 \( (\bigwedge_{i \in L} l_i)^p = \text{false} \).

We are now ready to propose an interaction invariant that takes all the solutions of the BBCs into account. We first introduce the notation \( \tilde{f} \) 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 \( \gamma \) applied to a tuple of components \( B = (B_1, \ldots, B_n) \), the interaction invariant of \( \gamma(B) \) can be obtained as the dual of \( |\gamma(B)|^p \), denoted by \( |\gamma(B)|^p \).

**Example 3.** We consider the components, connectors, and BBCs introduced in Example 2. The positive mapping removes variables with negations from \( \delta_1(B) \) and \( \delta_2(B) \).

We obtain that \( |\delta_1(B)|^p = (l_1 \lor l_2) \land (l_3 \lor l_4) \land (l_5 \lor l_6) \land (l_7 \lor l_8) \), and \( |\delta_2(B)|^p = (l_1 \lor l_2) \land (l_3 \lor l_4) \land (l_5 \lor l_6) \land (l_7 \lor l_8) \).

If we specify \( \text{Init} = l_1 \lor l_2 \), every invariant of system \( \delta_1(B) \), \( \text{Init} \) and \( \delta_2(B) \) should contain either \( l_1 \) or \( l_2 \). Therefore \( (l_1 \lor l_2) \land (l_3 \lor l_4) \) is the interaction invariant of \( \delta_1(B) \), \( \text{Init} \), and \( (l_1 \lor l_2) \land (l_3 \lor l_4) \land (l_5 \lor l_6) \land (l_7 \lor l_8) \) is the interaction invariant of \( (l_1 \lor l_2 \lor l_3 \lor l_4) \), \( \text{Init} \).

**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 \( \gamma \) 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

\[
|\bigcup_{i=1}^{n} \delta_i(\gamma) | = |\gamma - \bigcup_{i=1}^{n} \delta_i(\gamma)| \land \bigcap_{i=1}^{n} |\delta_i(B)|
\]

**Proposition 6** 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, \( \gamma - \bigcup_{i=1}^{n} \delta_i \) is represented by \( \delta_0 \). 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 \( \gamma \) are \( \delta_1 = p_1 \land p_3 \land q_4 \) and \( \delta_2 = p_2 \land q_2 \). The new connector obtained by applying \( \delta_1 \) and \( \delta_2 \) to \( \gamma \) is given by \( \delta_1 + \delta_2 \).

The BBC \( |\delta_1(B)| \) and \( |\delta_2(B)| \) are respectively given by:

\[
|\delta_1(B)| = (l_0 \Rightarrow l_1 \lor l_4) \land (l_1 \Rightarrow l_0 \lor l_3) \land (l_3 \Rightarrow l_4 \lor l_0) \land (l_4 \Rightarrow l_0)
\]

\[
|\delta_2(B)| = (l_0 \Rightarrow l_2 \lor l_3) \land (l_2 \Rightarrow l_4 \lor l_0) \land (l_4 \Rightarrow l_3)
\]

Since \( \gamma - (\delta_1 + \delta_2) = \emptyset \), we have \( |(\delta_1 + \delta_2)(\gamma)| = |\delta_1(B)| \land |\delta_2(B)| \).

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 \( L_c = \bigcup_{i \in [1,n]} \text{support}(\gamma_i) \land \text{support}(\gamma_j) \), where \( \text{support}(\gamma) = \bigcup_{a \in \gamma} a \cup a^* \), the set of locations involved in some interaction \( a \) of \( \gamma \).

Our incremental method assumes that an invariant has already been computed for a set of interactions (We use \( L_0 \)
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 $\gamma$ can be regarded as separately applying increments over their constituents. We propose the following proposition, which builds on Equation [2].

**Proposition 6.** Consider a composite component $B$. Let $\gamma$ be a connector for $B$ and assume a set of increments $\{\delta_i\}_{1 \leq i \leq n}$ over $\gamma(B)$. Let $\delta_0 = \gamma - (\sum_{i=1}^{n} \delta_i)^f$, $I_{\delta_i} = \bigwedge_{k \in I_i} \phi_k$, 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

- $L_0$ be the set of location variables in invariant $\phi_0$,
- $L_c$ be the common location variables between $\{\delta_0, \delta_1, \ldots, \delta_n\}$.

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

$$I = \left( \bigwedge_{i=0}^{n} \bigwedge_{k \in I_i} \phi_k \right) \wedge \left( \bigwedge_{(k_1, \ldots, k_r) \in D} \phi_{k_{ij}} \right)$$

where

$$D = \{ (k_{1j}, \ldots, k_{rj}) \mid (\forall j = 1 \ldots r \wedge k_{ij} \in I_{ij}) \wedge (L_{\phi_{k_{ij}}} \cap L_c \neq \emptyset) \wedge (\bigwedge_{j=1}^{r} m_{k_{ij}} \neq false) \wedge ((k_{1j}, \ldots, k_{rj}) \text{ is maximal}) \}.$$ 

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 satisfaction 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 [2] we have computed the BBCs for the two increments. Here we show how to compute the invariants from BBCs of the increments. By Definition [2] we obtain that $L_c = \{l_0\}$. Let $S_{\delta_1}, S_{\delta_2}$ be the BBC-solutions for $|\delta_1(B)|$ and $|\delta_2(B)|$ respectively, and $I_{\delta_1}, I_{\delta_2}$ be their BBC-invariants, we have:

- $S_{\delta_1} = (l_0 \wedge l_1 \wedge l_3 \wedge l_4) \wedge (l_0 \wedge l_1) \wedge (l_2 \wedge l_3) \wedge (l_1 \wedge l_2) \wedge (l_3 \wedge l_4) \wedge (l_0 \wedge l_4) \wedge (l_1 \wedge l_4) \wedge (l_2 \wedge l_4) \wedge (l_1 \wedge l_3) \wedge (l_2 \wedge l_3) \wedge (l_3 \wedge l_4)$,
- $S_{\delta_2} = (l_0 \wedge l_2 \wedge l_3 \wedge l_4) \wedge (l_0 \wedge l_2) \wedge (l_2 \wedge l_3) \wedge (l_0 \wedge l_3) \wedge (l_3 \wedge l_4) \wedge (l_0 \wedge l_4) \wedge (l_2 \wedge l_4) \wedge (l_3 \wedge l_4)$,
- $I_{\delta_1} = (l_0 \wedge l_1) \wedge (l_0 \wedge l_3) \wedge (l_1 \wedge l_3) \wedge (l_0 \wedge l_4) \wedge (l_1 \wedge l_4) \wedge (l_2 \wedge l_4) \wedge (l_3 \wedge l_4)$,
- $I_{\delta_2} = (l_0 \wedge l_2) \wedge (l_0 \wedge l_3) \wedge (l_2 \wedge l_3) \wedge (l_0 \wedge l_4) \wedge (l_2 \wedge l_4) \wedge (l_3 \wedge l_4)$.

Because $I_{\delta_1+\delta_2}(B) = I_{\gamma(\delta_1+\delta_2)(\gamma)} + \delta_1 + \delta_2(B)$ and $\gamma - (\delta_1 + \delta_2)^f = \emptyset$, we have $I_{\delta_1+\delta_2}(B) = I_{\delta_1+\delta_2}(B)$.

Among the BBC-invariants, $(l_1 \wedge l_3), (l_1 \wedge l_4), (l_3 \wedge l_5), (l_5 \wedge l_6)$ do not contain any common location variables, so they will remain in the global computation. BBC-invariants $(l_0 \wedge l_1), (l_0 \wedge l_4), (l_0 \wedge l_2)$ and $(l_0 \wedge l_6)$ 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

**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 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 [8], FP for the incremental method of [4], 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. We will discuss this method in Section [11] 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 [8], 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 properties. 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
is no deadlock else either it generates stronger component and interaction invariants or it tries to confirm the detected deadlocks by using reachability analysis technique.\footnote{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.}

### B. Implementation of the Incremental Method

We build on the symbolic implementation of the method in \footnote{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.} that computes the interaction invariant of an entire system with all the interactions within the connector. The implementation relies on the CUDD package \footnote{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.} 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 \textit{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 \footnote{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.}, the Smoker \footnote{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.}, the Automatic Teller Machine (ATM) \footnote{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.} 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, \textit{scale} is the “size” of examples; \textit{location} denotes the total number of control locations; \textit{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 \textit{global}. The means that we are also faster than tools such as NuSMV and SPIN that are known to be much slower than \textit{global} on these case studies \footnote{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.} \footnote{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.}. Our \textit{Incr} technique is faster than \textit{FP} except for the Gas Station and it always consumes less memory.

### Table I

<table>
<thead>
<tr>
<th>Component information</th>
<th>Gas Station</th>
<th>ATM</th>
<th>Smoker</th>
<th>DALA robot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (minutes)</td>
<td>0:44</td>
<td>0:51</td>
<td>0:40</td>
<td>0:13</td>
</tr>
<tr>
<td>Memory (MB)</td>
<td>38</td>
<td>54</td>
<td>25</td>
<td>9</td>
</tr>
</tbody>
</table>

### Table II

<table>
<thead>
<tr>
<th>Component information</th>
<th>Time (minutes)</th>
<th>Memory (MB)</th>
<th>Global</th>
<th>Incr</th>
<th>Global</th>
<th>Incr</th>
</tr>
</thead>
<tbody>
<tr>
<td>scale</td>
<td>location</td>
<td>interaction</td>
<td>global</td>
<td>FP</td>
<td>Incr</td>
<td>global</td>
</tr>
</tbody>
</table>

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 \textit{FP} (this is the only case for which \textit{global} was faster than \textit{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 \textit{DALA} robot \footnote{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.}. \textit{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 \textit{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 \textit{global} and \textit{FP} fail to check for deadlock-freedom (Antenna is the only module that can be checked by using \textit{global}). However, by using \textit{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...
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 component(s) 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 Iner and invariant preservation methods can be found in §.

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 been seen in Section IV our new techniques and the ones in § 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


TABLE III

<table>
<thead>
<tr>
<th>METHOD</th>
<th>component location interactions</th>
<th>latency</th>
<th>time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SICK</td>
<td>45</td>
<td>202</td>
<td>2.01 × 10^-4</td>
</tr>
<tr>
<td>Aspect</td>
<td>29</td>
<td>160</td>
<td>2.17 × 10^-3</td>
</tr>
<tr>
<td>NDD</td>
<td>27</td>
<td>152</td>
<td>2.99 × 10^-4</td>
</tr>
<tr>
<td>REFLEX</td>
<td>56</td>
<td>300</td>
<td>2.04 × 10^-4</td>
</tr>
<tr>
<td>Battery</td>
<td>30</td>
<td>176</td>
<td>2.22 × 10^-4</td>
</tr>
<tr>
<td>Heating</td>
<td>26</td>
<td>149</td>
<td>2.17 × 10^-4</td>
</tr>
<tr>
<td>Plaine</td>
<td>37</td>
<td>174</td>
<td>2.09 × 10^-4</td>
</tr>
</tbody>
</table>

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