Softw Syst Model (2016) 15:427–451
DOI 10.1007/s10270-014-0410-8

REGULAR PAPER

Component-based verification using incremental design and invariants

Saddek Bensalem · Marius Bozga · Axel Legay · Thanh-Hung Nguyen · Joseph Sifakis · Rongjie Yan

Received: 4 February 2013 / Revised: 8 December 2013 / Accepted: 27 March 2014 / Published online: 24 April 2014

Abstract We propose invariant-based techniques for the efficient verification of safety and deadlock-freedom properties of component-based systems. Components and their interactions are described in the BIP language. Global invariants of composite components are obtained by combining local invariants of their constituent components with interaction invariants that take interactions into account. We study new techniques for computing interaction invariants. Some of these techniques are incremental, i.e., interaction invariants of a composite hierarchically structured component are computed by reusing invariants of its constituents. We formalize incremental construction of components in the BIP language as the process of building progressively complex components by adding interactions (synchronization constraints) to atomic components. We provide sufficient conditions ensuring preservation of invariants when new interactions are added. When these conditions are not satisfied, we propose methods for generating new invariants in an incremental manner by reusing existing invariants from the constituents in the incremental construction. The reuse of existing invariants reduces considerably the overall verification effort. The techniques have been implemented in the D-Finder toolset. Among the experiments conducted, we have been capable of verifying safety properties and deadlock-freedom of sub-systems of the functional level of the DALA autonomous robot. This work goes far beyond the capacity of existing monolithic verification tools.

Keywords Verification method · Invariant · Component-based systems · Incremental design · Verification tools · Deadlock-freedom · BIP

1 Introduction

Component-based design confers numerous advantages, in particular, an increased productivity through reuse of existing components. Nonetheless, establishing the correctness of the designed systems remains an open issue. In contrast to other engineering disciplines, software and system engineering hardly ensures predictability at design time. Consequently, a posteriori verification as well as empirical validation is essential for ensuring correctness of designed systems.

Monolithic verification [34,68] of component-based systems is a challenging problem. It often requires computing for a composite component the product of its constituents by using both interleaving and synchronization. The

Communicated by Prof. Joanne Atlee.

Authors are in alphabetical order by last name.

The present article is an improved version of [23] and [15]. This work has been partially supported by the European COMBEST Project No. 215543, by the "ARTEMIS CROSS-Domain Architecture " ACROSS, by the PRO3D Project No. 248776, by the regional CREATIVE program of the Britany region, by Action de Recherche Collaborative" ARC (TP)I, and by National Science Foundation of China under Grant No. 61100074.

S. Bensalem · M. Bozga · J. Sifakis
UJF-Grenoble 1/CNRS, VERIMAG UMR 5104,
38041 Grenoble, France

A. Legay
INRIA/IRISA, 35042 Rennes, France

T.-H. Nguyen
Department of Software Engineering, Hanoi University of Science and Technology, Hanoi, Vietnam

R. Yan (✉)
State Key Laboratory of Computer Science, Beijing 100190, China
e-mail: yan.rongjie@gmail.com

complexity of the product system is often prohibitive due to state explosion.

In a series of recent works, it has been advocated that *compositional verification techniques* can be used to cope with state explosion [35,36,43,45,49,55,66]. The key to compositional verification techniques is the application of divide-and-conquer techniques to infer global properties of complex systems from properties of their components.

A compositional verification method based on invariant computation is presented in [16,17]. The method is based on the following rule:

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

The rule allows to prove invariance of property Φ for systems obtained by using an n-ary composition operation $||$ on a set of components $\{B_i\}_i$ parameterized by a set of interactions γ. It is based on the computation of a global invariant that is the conjunction of local invariants Φ_i of constituent components B_i and an *interaction invariant* Ψ. The latter expresses constraints on the global state space induced by interactions between components. In [16], we have shown that Ψ can be computed automatically from abstractions of the system to be verified. That is, we provide an effective procedure, denoted II in the rule, which allows to compute interaction invariants from finite-state abstraction of the components B_i with respect to its local invariant Φ_i. The method has been implemented in the D-Finder toolset [17] and applied to check deadlock-freedom on several case studies described in the BIP (Behavior, Interaction, Priority) [12] language. The results of these experiments show that D-Finder is sometimes exponentially faster for checking deadlock-freedom than state-of-the-art verification tools such as NuSMV [33].

This paper introduces new techniques for the computation of interaction invariants. These techniques are extensions of the one introduced earlier in [16,18] that allows the computation of interaction invariants in an enumerative manner. Our first contribution is a new symbolic technique, based on *Boolean Behavioral Constraints* (BBCs), that allows to relate interactions between different components with their internal transitions. BBCs are used to compute interaction invariants by applying two different symbolic techniques (1) by iterative computation of fixpoints and (2) by solution of a set of Boolean equations. Both techniques have been implemented and outperform the enumerative method proposed in [16,18]. We have shown that the efficiency of each technique largely depends on the architecture of the composite components to be verified.

The second contribution is an incremental verification method that further improves the previous BBC-based method. The key idea is to reuse the already computed invariants of the constituents of a composite component in order to compute global invariants. This requires the formalization of the construction of hierarchically structured composite components. For instance, Fig. 1 shows a composite component obtained as the composition of two components by using a set of interactions γ_2. One of these components is the composition of two components by using the set of the interactions γ_1. The computation of invariants of the component $\gamma_1(B_1, B_2)$ is obtained by combining invariants of the atomic components B_1 and B_2 with an interaction invariant characterizing the restriction of the interactions γ_1 on the product of the composed components. Following the incremental construction process, invariants of $\gamma_2(B_3, \gamma_1(B_1, B_2))$ are obtained by combining invariants of $\gamma_1(B_1, B_2)$ and invariants of B_3 with an interaction invariant induced by the application of γ_2.

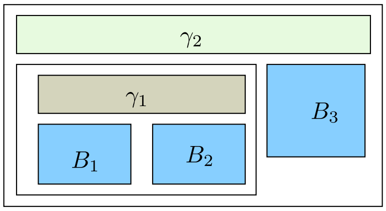

Fig. 1 Incremental design

It is nevertheless important to mention that none of the previously implemented methods take incremental design aspects into account. Incremental system design often works 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 intermediate verification steps are generally avoided. This situation can be improved if the result of the verification process is reused when new interactions are added. Existing methods, including the ones in [16,18], do not focus on such aspects of modular verification.

We present a method for incremental construction and verification of component-based systems. First, we propose a formalization of incremental component-based design. Then we propose 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 techniques for generation of new invariants in an incremental manner, in which understanding the way of construction is a prerequisite to achieve a better performance in the incremental verification. We shall see that, in many situations, the reuse of existing invariants reduces considerably the verification effort. The techniques can also be generalized to verify other models formalized with concurrent processes, coordinated with synchronization and interleaving between different processes.

The proposed techniques have been implemented within the D-Finder toolset [17] and applied on several case studies. The experiments show that these techniques are generally much faster than the ones proposed in [16,18] for both traditional examples and larger case studies.[1] In particular, we have been able to verify deadlock-freedom and safety properties of a large part (tens of components, hundreds of interactions) of the functional level of the DALA autonomous robot.

It is important to notice that from the verified BIP model, it is possible to generate C code. For DALA, BIP generated C code (more than 500,000 lines) for application modules and their coordination at execution level. The generated C code preserves properties of the BIP model and deadlock-freedom in particular.

Related work

System design is nowadays supported by a wide variety of modeling frameworks and tools. These frameworks are usually domain specific and range from hardware modeling (e.g., BlueSpec [27]), hardware/software modeling (e.g., Metropolis [4]), concurrent systems modeling (e.g., UNITY [30], I/O automata [61], reactive modules [3]), embedded systems modeling (e.g., Ptolemy [72]), physical system modeling [48].

All the above cited frameworks are using specific categories of atomic components and particular operators for parallel composition. In general, the choices are driven by domain practices and perfectly fulfill the concrete needs of system designers. In addition, all of them are rigorous as they benefit from solid semantical foundations. In the same spirit, the BIP framework aims at rigorous modeling and design of component-based heterogeneous real-time systems. The relations between BIP and these frameworks have been previously discussed in [1,12,28]. In contrast to all previously cited approaches, the definition of BIP aims to provide a neat separation of concerns between behavior (atomic components) and architecture (expressed as the combination of interactions and priorities). The composition operators used to express architectural constraints in BIP have been chosen such as to provide maximal expressivity. They confer to BIP strong modeling capabilities that cannot be matched by other languages [26].

Incremental design provides the ability to construct progressively a system by adding new components (or subsystems) and interactions. That is, new elements are continuously added without modifying or breaking the already existing sub-systems and components already in place. Incremental design has been already investigated in [40,58,60]. Nevertheless, none of these approaches has considered the integration of incremental construction with verification. The incremental design methodology based on BIP and introduced hereafter provides a sound basis to infer the properties preserved in the incremental design. Moreover, in the case where the rule of incremental construction is violated, it still allows to verify the correctness of system properties by reusing the invariants of the constituents.

A detailed comparison between our verification method [16,18] and methods based on deductive techniques [62] or assume-guarantee techniques [35,36,43,45,49,55,57,66] has already been provided in [16,18]. Deductive techniques rely on finding an inductive invariant of a given program that is stronger than the invariant to be verified. The drawback of these techniques is that there is no good method to find such an invariant. Assume-guarantee methods always have difficulties to find decompositions into subsystems and choosing adequate assumptions for such a particular decomposition. Our method only requires the computation of component invariants and interaction invariants without any additional assumptions and thus avoid this problem.

Rely-guarantee methods are also used to verify safety properties of multi-threaded systems [46,51,67]. Different from assume-guarantee techniques, rely-guarantee methods do not consider decomposition strategies. Model checking over each thread is done with environment assumptions. Traditional model checking or predicate abstraction can be applied for individual threads. Environment assumptions are established according to global variable updates [46] or environment transitions, and their mutual dependencies and refined to check specified properties [51,67]. Our method does not distinguish local transition or global interactions. We consider all the synchronized/interleaving transitions together and compute the invariants as a whole.

A series of recent works [2,41,42,59] propose compositional techniques based on interface theories. The conceptual idea is to check whether a component satisfies a property that is expressed by an interface modeled as an automaton. Using compositional design-based rules, one can infer the interface that will be satisfied by the combination of the components. Interface theories permit richer logical operations than those available in our framework. However, the communication primitive that drives the composition between interfaces is not as expressive as connectors in BIP. Moreover, it is hard to decompose the interface representing the global properties into smaller interfaces on basic components. Finally, representing deadlock with interfaces involves a tedious task.

Incremental verification methods are also widely used in programs analysis [38,56]. The key of these methods is the updated points. Starting from fixed-point algorithms, validity of existing analysis is computed and the change will be propagated according to the type of modifications [56], or the derivation graph [38], until a fixed-point is reached. Our

[1] See Sect. 6 for detailed explanation.

incremental methods are generic and only consider the incremental points in the term of involved locations.

Verifying implementations of concurrent systems is a challenging problem which is intensively studied [5–7,24, 39]. Most of existing solutions consist in performing a static analysis of the code and generate an abstract mathematical model [54] on which properties are verified with powerful tools such as SAT solvers [44,63]. Other approaches use theorem provers [53]. Here we take a completely different approach. Instead of verifying the code, we ensure that the model is correct and its correctness is preserved in the automatically generated code by our tools. We believe that ensuring code correctness via its model is a breakthrough as it allows to take the incremental design into account, which drastically helps simplify verification. Moreover, our verification method is independent of any programming language.

Structure of the paper Section 2 introduces our compositional and incremental design framework. Section 3 proposes the formal definition of invariant and invariant preservation. Section 4 introduces our new techniques to compute interaction invariants. Sections 5 and 6 discuss implementation and experiments that have been conducted. Finally, Sect. 7 concludes the paper and suggests directions for future research.

Notations The following notations are extensively used:

- $I, J, \ldots,$ refer to finite sets of integers.
- We denote by $[i,\ j]$ the set $\{i, i+1, \ldots, j\}$ for any two integers $i,\ j$ such that $i \leq j$.
- For a set of Boolean variables L, the predicates over L are denoted by $Bool[L]$.
- For a Boolean variable $l \in L$, we use $\bar{l}$ to denote its negation, i.e., $\neg l$.
- We use classical algebraic notations for sets where $\oplus$ and $\sum$ represent the union of elements in the set, and $\ominus$ the difference of two sets.

2 Component-based system design

In this section, we introduce concepts that will be used through the rest of the paper. In Sect. 2.1, we introduce the basic models for component-based design. In Sect. 2.2, we propose extensions to model and reason on the incremental design of component-based systems. This is the basis for inferring the preservation of already established properties and efficient incremental verification.

2.1 Components and interactions

The component-based framework used in this paper is a subset of the BIP—*Behavior, Interaction, Priority*—framework [10,25]. BIP supports a component-based modeling methodology based on the assumption that components are obtained as the superposition of three orthogonal layers, that is:

- *behavior*, specified as a set of finite-state automata or 1-safe Petri nets [32] extended with interaction ports, local data (in form of C data variables) and data operations (in form of C functions),
- *multiparty interactions*, used to coordinate the actions of behavior and specified using hierarchically structured connectors,
- *dynamic priorities*, used to schedule among multiple enabled interactions and specified by priority rules.

Components are composed by layered application of multiparty interactions and of priorities. Interactions express synchronization constraints between actions of the composed components, while priorities are used to filter among possible interactions and to schedule system evolution so as to meet performance requirements, e.g., to express scheduling policies. Interactions are described in BIP as the combination of two types of elementary protocols: *rendez-vous* to express strong symmetric synchronization and *broadcast* to express triggered asymmetric synchronization [26].

Observe that all atomic components in BIP are 1-safe Petri nets labeled with ports and extended with local data and data operation. Yet, atomic components represent only the “behavior” layer of BIP models. In addition, BIP provides the composition operators and the methodology for composition using the “interactions” and “priorities” layers. Conceptually, any composite component can be represented as an extended Petri net; however, BIP avoids the explicit construction and manipulation of such nets. Instead, it always considers the layered/structured representation of the model in terms of atomic components and glue operators.

Using less expressive frameworks based on simpler composition operators often leads to intractable models when used to express high-level coordination. Modeling multiparty interaction in frameworks supporting only point-to-point interaction, e.g., binary synchronization as in CCS or function call, requires the use of protocols. This leads to overly complex models with complicated coordination structure. Additionally, interactions and priorities define a clean and abstract concept of *system architecture* which is fully separated from behavior. Architecture in BIP is a first class concept with well-defined semantics that can be analyzed and transformed.

The usefulness of BIP has been also empirically assessed on a large basis of examples and industrial case studies. There exists methods and tools[2] for generating BIP models from programming languages and/or programming models with well-defined operational semantics. They include model

[2] See http://www.bip-components.com for the complete list.

generators for languages such as C, Lustre, Simulink, and NesC/TinyOS or for programming models such as DOL[71], GeNoM[47] and have been illustrated in [8,13,21,29]. Moreover, BIP has been also used to develop from scratch component-based models of complex systems including heterogeneous communication networks [9,11] and multimedia systems [14]. For these successful applications, BIP has also been used in many industrial projects such as ASCENS,[3] PRO3D[4] and ACROSS.[5]

BIP can model component-based heterogeneous systems with features such as clocks and various data types. However, in this paper, we restrict ourselves to a strict subset of BIP, that is, without data and without dynamic priorities.

We have previously shown in [16] how data can be taken into account for computing invariants through abstraction. Regarding priorities, we do not consider them, however, let us remark that priorities preserve invariant properties and deadlock-freedom [50].

Behavior is represented through atomic components. An atomic component is a transition system whose transitions are labeled by *ports*. Ports are used for interacting 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 control locations,
- P is a set of ports,
- $\mathcal{T} \subseteq L \times P \times L$ is a set of transitions.

Given $\tau = (l, p, l') \in \mathcal{T}$, l and l' are the *source* and *destination* locations, respectively. We use ${}^{\bullet}\tau$ and $\tau^{\bullet}$ to refer to the source and destination of τ, respectively. We also denote by $port(\tau)$ the port of a transition τ. In what follows, we shall consider locations to be Boolean variables; the variable for a location being true iff the component is currently in the given location. However, we shall observe that the language of BIP is more general and also allows to attach value of structured variable to a given location. Considering this extension of the language remains beyond the scope of the present paper.

Example 1 Figure 2 presents three atomic components B_1, B_2, and B_3. In the graphic representation, we use circles to describe locations, arrows between circles for transitions and bullets on components for ports. The ports of component B_1 are p_3 and q_3. B_1 has two control locations: l_3 and l_4 and two transitions: $\tau_{11} = (l_3, p_3, l_4)$ and $\tau_{12} = (l_4, q_3, l_3)$.

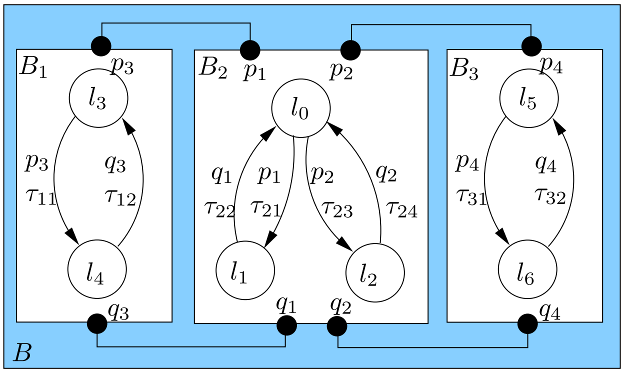

Fig. 2 Running example

In BIP, components are composed via *interactions*, i.e., by synchronization on ports for their corresponding synchronized transitions. An interaction requires that at most one port from any component can join the synchronization.

Definition 2 (*interactions, connectors*) Given a set of n components $\{B_i = (L_i, P_i, \mathcal{T}_i)\}_{1 \le i \le n}$, an interaction a is a set of ports from $\bigcup_{i=1}^{n} P_i$, such that $\forall i \in [1, n]. |a \cap P_i| \le 1$. A connector is a set of interactions.

Interactions are represented by lines connecting the ports in the graphic representation. As an example, the interaction $\{p_1, p_3\}$ between components B_1 and B_2 given in Fig. 2 describes a synchronization between components B_1 and B_2 by ports p_1 and p_3. Another interaction is given by the set $\{q_1, q_3\}$. The connector for B_1, B_2 and B_3 is the set $\{\{p_1, p_3\}, \{q_1, q_3\}, \{p_2, p_4\}, \{q_2, q_4\}\}$. We simplify the notations and write $p_1 p_2 \ \ldots \ p_k$ instead of $\{p_1, \ldots, p_k\}$ for an interaction. We also write $a_1 \oplus \cdots \oplus a_m$ for the connector $\{a_1, \ldots, a_m\}$, where $\oplus$ is associative. The connector $\{\{p_1, p_3\}, \{q_1, q_3\}, \{p_2, p_4\}, \{q_2, q_4\}\}$ becomes $p_1\, p_3 \oplus q_1\, q_3 \oplus p_2\, p_4 \oplus q_2\, q_4$.

We use $I(P)$ to denote the set of all interactions that are defined over a given set of ports P and $\Gamma(P)$ as the set of all connectors over P.

Interactions define communication between components obtained by synchronization of different transitions. The relation between interactions and interacting transitions is captured by the following definition.

We define, respectively, composite component and system.

Definition 3 (*composite component, system*) Given a set of n atomic components $\{B_i = (L_i, P_i, \mathcal{T}_i)\}_{1 \le i \le n}$ and a connector γ, we define the composite component $B = \gamma(B_1, \ldots, B_n)$ as the transition system $(\mathcal{L}, \gamma, \mathcal{T})$, where:

- $\mathcal{L} = L_1 \times L_2 \times \cdots \times L_n$ is the set of *global* states,
- γ is the set of interactions,

[3] http://www-verimag.imag.fr/ASCENS.html.

[4] http://www-verimag.imag.fr/PRO3D.html.

[5] http://www-verimag.imag.fr/ACROSS.html.

- $\mathcal{T} \subseteq \mathcal{L} \times \gamma \times \mathcal{L}$ contains transitions $\tau = ((l_1, \ldots, l_n), a, (l'_1, \ldots, l'_n))$ satisfying the following rule:

$$\frac{I \subseteq [1, n] \qquad a = \{p_i\}_{i \in I} \in \gamma \qquad \forall i \in I \quad l_i \xrightarrow{p_i}_i l'_i \qquad \forall i \notin I \qquad l_i = l'_i}{(l_1, \ldots, l_n) \xrightarrow{a} (l'_1, \ldots, l'_n)}$$

A system $\mathcal{S}$ is a pair $\langle B, Init\rangle$ with $Init \in Bool[\bigcup_{i=1}^n L_i]$ characterizing the initial condition of B where $Bool[\bigcup_{i=1}^n L_i]$ is the set of Boolean expressions on $\bigcup_{i=1}^n L_i$.

Given an interaction a, only those components that are involved in a make a step. If a component does not participate to the interaction, then it remains in the same state.[6] The component $\gamma_\perp(B_1, \ldots, B_n)$, which is obtained by applying the connector $\gamma_\perp = \sum_{i=1}^n (\sum_{p \in P_i} p)$, is the transition system obtained by interleaving the transitions of atomic components.

Observe that any composite component can be viewed as a basic component in more complex designs.

Example 2 The composite component in Fig. 2 is defined by $\gamma(B_1, B_2, B_3)$, where $\gamma = p_1\, p_3 \oplus q_1\, q_3 \oplus p_2\, p_4 \oplus q_2\, q_4$.

We define *Forward Interaction Sets* of a location according to a set of interactions.

Definition 4 (*forward interaction sets*) Let γ be a connector over a set of components $\{B_i = (L_i, P_i, \mathcal{T}_i)\}_{1 \le i \le n}$. We define for every location $l \in \bigcup_{i=1}^n L_i$ its forward interaction set as follows:

$$\begin{aligned} l_\gamma^\bullet = \Big\{\{\tau_{i_1} \ldots \tau_{i_k}\} | (\forall j \in [1, k].i_j \in [1, n] \wedge (\tau_{i_j} \in \mathcal{T}_{i_j})) \\ \wedge (\exists j \in [1, k].{}^\bullet\tau_{i_j} = l) \\ \wedge (\{port(\tau_{i_1}) \ldots port(\tau_{i_k})\} \in \gamma)\Big\}. \end{aligned}$$

That is, $l_\gamma^\bullet$ contains sets of synchronized transitions by interactions of γ having at least one outgoing transition from l.

Example 3 Consider the components given in Fig. 2. Given $\gamma = p_1 p_3 \oplus q_1 q_3 \oplus p_2 p_4 \oplus q_2 q_4$,we have $l^\bullet_{0\gamma} = \{\{\tau_{21}, \tau_{11}\}, \{\tau_{23}, \tau_{31}\}\}$, $l^\bullet_{0(p_1 p_3)} = \{\{\tau_{21}, \tau_{11}\}\}$, and $l^\bullet_{0(p_2 p_4)} = \{\{\tau_{23}, \tau_{31}\}\}$.

An additional example is shown in Fig. 3. This example contains a component (B_2) that is non-deterministic. Given $\gamma = \{p_1 p_2\}$, we have $l^\bullet_{1\gamma} = l^\bullet_{3\gamma} = \{\{\tau_{11}, \tau_{21}\}, \{\tau_{11}, \tau_{22}\}\}$.

Remark 1 Given a set of n components $\{B_i = (L_i, P_i, \mathcal{T}_i)\}_{1 \le i \le n}$ and two interactions $a, \ b \in I(P)$ with $P = \bigcup_{i=1}^n P_i$, we have $l^\bullet_{a \oplus b} = l^\bullet_a \cup l^\bullet_b$ for any $l \in \bigcup_{i=1}^n L_i$.

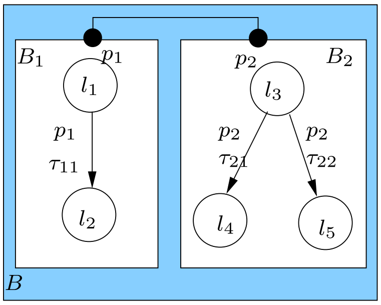

Fig. 3 A non-deterministic example

2.2 Incremental design

In component-based design, the construction of systems is both step-wise and hierarchical. A step consists in adding interaction (increment) at the same layer of the hierarchy in order to produce its next layer. In the rest of the paper, we focus on two kinds of operations that are layering and superposition. The first operation merges existing interactions of a layer with increments, while the second superposes increments over the same layer.

The basic unit for the construction is a set of atomic components available through a library of components that one assumed to be provided to the user. Those components constitute the first layer of the hierarchy. Atomic components are composed together in order to create the second layer of the hierarchy, that is a set of composite components. Those new components can be viewed as atomic components for the construction of the third layer of the hierarchy. The process can be repeated multiple times in order to build arbitrary complex hierarchies in a bottom-up manner. In what follows, we will say that layers of the hierarchy are obtained by adding increments to existing interactions between components. The rest of the section is divided as follows. Section 2.2.1 defines the operations used in incremental design flow of component-based systems. Section 2.2.2 studies relations between the components built along the flow. In what follows, we assume that all interactions are over components $\{B_i = (L_i, P_i, \mathcal{T}_i)\}_{1 \le i \le n}$ with $P = \bigcup_{i=1}^n P_i$.

2.2.1 Incremental construction

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

Definition 5 (*closure and forbidden interactions*) Let γ be a connector.

[6] If an interaction only allows two ports, the composite component returns to the traditional transition system with pair-wise synchronization

- 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 $\gamma^f = \gamma^c \ominus \gamma$, where " $\ominus$ " is set difference.

Example 4 Consider the interaction $p_1 p_3$ of the example given in Fig. 2. According to Definition 5, the closure of $p_1 p_3$ is $(p_1 p_3)^c = \{p_1 p_3, p_1, p_3\}$, and its forbidden set is $(p_1 p_3)^f = (p_1 p_3)^c \ominus p_1 p_3 = \{p_1, p_3\}$.

For two connectors γ_1 and γ_2, we have:

$$(\gamma_1 \oplus \gamma_2)^c = \gamma_1^c \oplus \gamma_2^c \text{ and } (\gamma_1 \oplus \gamma_2)^f = (\gamma_1 \oplus \gamma_2)^c \ominus \gamma_1 \ominus \gamma_2.$$

A connector describes a set of interactions. We assume that composite components are obtained from their constituents by further enforcing synchronization by using increments. Intuitively, an increment is obtained by merging existing interactions of a connector. We have the following definition.

Definition 6 (*increments*) Consider a connector $\gamma \in \Gamma(P)$. We say δ is an increment over γ if for any interaction $a \in \delta$ we have interactions $(b_j)_{j \in J} \subseteq \gamma$ such that $\Pi_{j \in J} b_j = a$, where Π represents the fusion of interactions into one.

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. Those forbidden interactions require weaker synchronization that may violate the stronger synchronization required by the increment. To avoid them, 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 7 (*layering*) Given a connector γ and an increment δ over γ, the new connector obtained by combining δ and γ, also called layering, is given by the following set $\delta\gamma = (\gamma \ominus \delta^f) \oplus \delta$ 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 increments, we can construct a system with multiple layers.

Example 5 Consider the connector $\gamma = \sum_{i=1}^{4} p_i \oplus q_i$ that is defined over the components in Fig. 2. An increment over γ is $\delta_1 = p_1\ p_3 \oplus q_1\ q_3$. According to Definition 7, one layering by increment δ_1 over connector γ is

$$\begin{aligned}
\delta_1\gamma &= (\gamma \ominus \delta_1^f) \oplus \delta_1 \\
&= ((p_1 \oplus p_2 \oplus p_3 \oplus p_4 \oplus q_1 \oplus q_2 \oplus q_3 \oplus q_4) \\
&\qquad \ominus (p_1 \oplus p_3 \oplus q_1 \oplus q_3)) \oplus (p_1 p_3 \oplus q_1 q_3) \\
&= p_1 p_3 \oplus q_1 q_3 \oplus p_2 \oplus q_2 \oplus p_4 \oplus q_4.
\end{aligned}$$

Besides the layering of interactions, incremental construction can also be obtained by first combining increments enforcing synchronization at the same layer. To combine many increments at the same layer into a single increment, we use the operation of *Superposition* defined as follows.

Definition 8 (*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 their union $\delta_1 \oplus \delta_2$.[7]

Notice that in general $(\delta_1 \oplus \delta_2)\gamma$ is different from $\delta_1 \delta_2 \gamma$. In the first term, δ_1 and δ_2 are in the same layer, while the successive application of increments defines two distinct layers.

At this stage, the reader understand that incremental construction involves both layering and superposition. Indeed, when we consider the whole system, we need to take the superposition of increments from different constituents over the same connector. Meanwhile, the concern of a system can be separated to different increments. Formally, we have the following proposition.

Proposition 1 *Let $\gamma \in \Gamma(P)$ be a connector, the incremental construction by the superposition of n increments $\{\delta_i\}_{1 \le i \le n}$ of γ is given by*

$$\left(\sum_{i=1}^{n} \delta_i\right)\gamma = \left(\gamma \ominus \left(\sum_{i=1}^{n} \delta_i\right)^f\right) \oplus \sum_{i=1}^{n} \delta_i. \qquad (1)$$

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

Example 6 Continue Example 5 and let $\delta_2 = p_2\, p_4 \oplus q_2\, q_4$ be the second increment. The incremental construction obtained by superposition of two increments δ_1 and δ_2 over γ is

$$\begin{aligned}
(\delta_1 \oplus \delta_2)\gamma = (&p_1 \oplus p_2 \oplus p_3 \oplus p_4 \oplus q_1 \oplus q_2 \oplus q_3 \oplus q_4 \\
&\ominus (p_1 \oplus p_3 \oplus q_1 \oplus q_3 \oplus p_2 \\
&\oplus p_4 \oplus q_2 \oplus q_4)) \oplus p_1\ p_3 \oplus q_1\ q_3 \oplus p_2 p_4 \\
&\oplus q_2\ q_4 = p_1\ p_3 \oplus q_1\ q_3 \oplus p_2\ p_4 \oplus q_2\ q_4.
\end{aligned}$$

[7] As increments are sets of interactions, we use $\oplus$ for the union between two increments.

2.2.2 Looser synchronization preorder

As we have seen, interactions characterize the behavior of a composite component. We first study relations between interactions and then step on a relation between connectors to reason about the features of incremental design in Sect. 3.2.

We consider the following notation of conflict between interactions.

Definition 9 (*conflict-free interactions*) Given a connector γ and interactions $a_1,\ a_2 \in \gamma$, such that $a_1 \cap a_2 = \emptyset$, we say that there is no conflict between a_1 and a_2.

Definition 10 (*looser synchronization preorder*) We define the looser synchronization preorder $\preccurlyeq \subseteq \Gamma(P) \times \Gamma(P)$. 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 $a = \Pi_{i=1}^{n} b_i$ and there is no conflict between any b_i and b_j, where $i,\ j \in [1,\ n]$ and $i \neq j$. We simply say that γ_1 is looser than γ_2.

The above definition says that stronger synchronization is obtained by the fusion of conflict-free interactions. The reason is that conflicting interactions may interfere, i.e., the execution of one interaction disables another conflicting interaction. If conflicting interactions are synchronized by using increments, this will violate initial design constraints. Notice that connectors $\gamma_1,\ \gamma_2,\ \gamma_3,\ \gamma_4$ such that $\gamma_1 \preccurlyeq \gamma_2$, and $\gamma_3 \preccurlyeq \gamma_4$, we have $\gamma_1 \oplus \gamma_3 \preccurlyeq \gamma_2 \oplus \gamma_4$.

Definition 11 (*interference-free connectors*) We say that two connectors $\gamma_1, \gamma_2 \in \Gamma(P)$ are interference-free if for any $a_1 \in \gamma_1,\ a_2 \in \gamma_2$, either a_1 and a_2 are conflict-free or $a_1 = a_2$.

Observe that if two connectors are interference-free, synchronizations enforced by one will not break or block synchronizations enforced by the other. Though we require that the interactions between γ_1 and γ_2 are conflict-free, γ_1 or γ_2, respectively, may contain conflicting interactions. For example, consider two connectors $\gamma_1 = p_1\ p_2 \oplus p_2\ p_3$ and $\gamma_2 = p_4\ p_5$. γ_1 is not conflict-free, but γ_1 and γ_2 are interference-free.

Lemma 1 *Given two interference-free connectors $\gamma_1,\ \gamma_2 \in \Gamma(P)$, we have $\gamma_1 \cap \gamma_2^f = \emptyset$ and $\gamma_2 \cap \gamma_1^f = \emptyset$, and $(\gamma_1 \oplus \gamma_2)^f = \gamma_1^f \oplus \gamma_2^f$.*

Proof Since γ_1 and γ_2 are interference-free, if $\gamma_1 \cap \gamma_2 = \emptyset$, we have $\gamma_1 \cap \gamma_2^f = \emptyset$ and $\gamma_2 \cap \gamma_1^f = \emptyset$. If $\gamma_1 \cap \gamma_2 \neq \emptyset$, for any $a \in \gamma_1 \cap \gamma_2$, we know that $a \notin \gamma_1^f$ and $a \notin \gamma_2^f$. Therefore, $\gamma_1 \cap \gamma_2^f = \emptyset$ and $\gamma_2 \cap \gamma_1^f = \emptyset$ still hold. According to Definition 5, we have $(\gamma_1 \oplus \gamma_2)^f = \gamma_1^c \oplus \gamma_2^c \ominus (\gamma_1 \oplus \gamma_2) = (\gamma_1^c \ominus (\gamma_1 \oplus \gamma_2)) \oplus (\gamma_2^c \ominus (\gamma_1 \oplus \gamma_2))$. Since γ_1 and γ_2 are interference-free, $\gamma_1^c \ominus (\gamma_1 \oplus \gamma_2) = \gamma_1^c \ominus \gamma_1 = \gamma_1^f$ and $\gamma_2^c \ominus (\gamma_1 \oplus \gamma_2) = \gamma_2^f$. So we have $(\gamma_1 \oplus \gamma_2)^f = \gamma_1^f \oplus \gamma_2^f$. □

Example 7 Consider the interactions in Fig. 2. Let $\gamma_1 = p_1\, p_3 \oplus q_1\, q_3$ and $\gamma_2 = p_2\, p_4 \oplus q_2\, q_4$. We have that γ_1 and γ_2 are interference-free.

3 Invariants and invariant preservation

In this section, we first recap the concept of *invariant* which we will use to verify properties of systems. As we have introduced incremental design and a preorder between two set of interactions in Sect. 2.2, we now relate the preorder relation with invariant concepts to propose sufficient conditions to guarantee that already satisfied invariants are not violated when new interactions are added to the design.

3.1 Component and system invariants

We now define the concept of invariants for components and systems.

Definition 12 (*inductive invariants*) Given a component $B = (L, P, \mathcal{T})$, a predicate Φ on L is an inductive invariant of B, denoted by $inv(B, \Phi)$, if for any location $l \in L$ and any port $p \in P$, $\Phi(l)$ and $l \xrightarrow{p} l' \in \mathcal{T}$ imply $\Phi(l')$, where $\Phi(l)$ means that l satisfies Φ.

Let $B = \gamma(B_1, \ldots, B_n)$ be the composition of n components with $B_i = (L_i,\ P_i, \mathcal{T}_i)$ for $i \in [1,\ n]$. An inductive invariant on B_i is called a *component invariant*, and an invariant involving locations of several components is called an *interaction invariant*. An interaction invariant expresses constraints on the global state space $L_1 \times L_2 \times \cdots \times L_n$ induced by interactions. We will assume that interaction invariants are predicates on $\bigcup_{i=1}^{n} L_i$.

Definition 13 (*system invariant*) For a system $\mathcal{S} = \langle B, Init \rangle$, Φ is a system invariant of $\mathcal{S}$, denoted by $inv(\mathcal{S}, \Phi)$, if there exists an inductive invariant Φ' of B such that $Init \Rightarrow \Phi'$ and $\Phi' \Rightarrow \Phi$.

Intuitively, system invariants are the predicates that are true at any reachable state. An invariant of a system is an over-approximation of its set of reachable states.

Definition 14 (*reachable states*) Given a component $\gamma(B)$ with a set of states $\mathcal{L}$, we define $reach(s,\ \gamma(B)) = \{s_i \in \mathcal{L}\ |\exists a_1, \ldots, a_n \in \gamma.s = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n\}$ the set of reachable states from $s \in \mathcal{L}$ by interactions of γ.

The above definition provides a notation for the set of reachable states from a state s through all possible interactions in $\gamma(B)$. If there is no executable interaction from s, we have that $reach(s, \gamma(B)) = \{s\}$.

Invariants can be used to check deadlock-freedom as explained below. Global deadlocks are states where no interaction can be executed. They depend on the enabling condition of all interactions. For a given interaction a, its enabling condition characterizes all the global states from which it can be executed, that is, all the states from which all the ports involved in the interaction are ready for synchronization. A port is ready if at least one of its transitions is enabled.

Consider a set of components $\{B_i = \{L_i, P_i, \mathcal{T}_i\}\}_{1 \le i \le n}$ with $\mathcal{T} = \bigcup_{i=1}^{n} \mathcal{T}_i$, and a set of transitions $\mathcal{T}' \subseteq \mathcal{T}$ such that $port(\tau_i) \neq port(\tau_j)$ for any distinct $\tau_i,\ \tau_j \in \mathcal{T}'$ and let $port(\mathcal{T}') = \{port(\tau)\,|\tau \in \mathcal{T}'\}$ be the set of ports labeling them. Let $en(a)$ be the set of all the states from which interaction a can be executed, i.e., $en(a) = \bigvee_{port(\mathcal{T}')=a}(\bigwedge_{\tau\in\mathcal{T}'} {}^{\bullet}\tau)$. The predicate $DIS = \bigwedge_{a\in\gamma} \neg en(a)$ characterizes the set of the states of $\gamma(B_1, \ldots, B_n)$ from which all interactions are disabled.

A component is deadlock-free iff the predicate $\neg DIS$ is an invariant. Obviously, in that case, all the reachable states of the component satisfy $\neg DIS$ which means that no state in DIS is reachable.

3.2 Invariant preservation

We extensively use the following well-known result about invariants [16].

Theorem 1 *If $\Phi_1,\ \Phi_2$ are invariants of B (respectively $\mathcal{S}$), then $\Phi_1 \wedge \Phi_2$ and $\Phi_1 \vee \Phi_2$ are also invariants of B (respectively $\mathcal{S}$).*

An invariant is an over-approximation of the set of reachable states. The relation between sets of reachable states, which are obtained by applying, respectively, two connectors over the same set of components, provides a way to reason about invariant preservation.

Lemma 2 *Given two connectors γ_1, γ_2 over B with a set of states $\mathcal{L}$, if $\gamma_1 \preccurlyeq \gamma_2$, we have $reach(s, \gamma_2(B)) \subseteq reach(s, \gamma_1(B))$, for any $s \in \mathcal{L}$.*

Proof Let $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_m} s_m$ be an execution sequence from $s \in \mathcal{L}$ in $\gamma_2(B)$, where $a_i \in \gamma_2$ for $i \in [1, m]$. Since $\gamma_1 \preccurlyeq \gamma_2$, we have that for any a_i, there exists a set of conflict-free interactions $b_j \in \gamma_1$ such that $a_i = \Pi_{j=1}^{k} b_j$. From any state s_i in the sequence started from s in $\gamma_2(B)$, there exists a set of interactions $\bigcup_{j=1}^{k} b_j$ such that $s_i \xrightarrow{b_1} \cdots \xrightarrow{b_k} s_{i+1}$. Therefore, we conclude that $reach(s, \gamma_2(B)) \subseteq reach(s, \gamma_1(B))$ for any $s \in \mathcal{L}$. □

This lemma shows that from the same state, the set of reachable states under a tighter connector is always a subset of reachable states under a looser connector.

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 $inv(\gamma_1(B),\ \Phi) \Rightarrow inv(\gamma_2(B), \Phi)$, for all Φ.*

Proof Consider $reach(s, \gamma_2(B)) \subseteq reach(s, \gamma_1(B))$. For any $s' \in reach(s,\ \gamma_2(B))$, s' is reachable in $\gamma_1(B)$. As $inv(\gamma_1(B), \Phi)$ is true, according to Definition 12, we also have $\Phi(s')$. So we can conclude that $inv(\gamma_2(B), \Phi)$ is true. □

The above proposition, which will be used in the incremental verification, simply says that if an invariant is satisfied, then it will be preserved 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 provide sufficient conditions to guarantee that invariants are preserved by the incremental construction through layering of a tighter increment over its connector.

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

Proof Because $\gamma \preccurlyeq \gamma \ominus \delta^f$, we have $\gamma \preccurlyeq (\gamma \ominus \delta^f) \oplus \delta = \delta\gamma$. □

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 now present the main result of the subsection.

Proposition 4 *Consider two increments $\delta_1,\ \delta_2$ over $\gamma(B)$ such that $\gamma \preccurlyeq \delta_1$ and $\gamma \preccurlyeq \delta_2$. If δ_1 and δ_2 are interference-free, and $inv(\delta_1\gamma(B), \Phi_1),\ \ inv(\delta_2\gamma(B), \Phi_2)$, we have $inv((\delta_1 \oplus \delta_2)\gamma(B), \Phi_1 \wedge \Phi_2)$.*

Proof We will show that $\delta_1\gamma \preccurlyeq (\delta_1 \oplus \delta_2)\gamma$ and $\delta_2\gamma \preccurlyeq (\delta_1 \oplus \delta_2)\gamma$, then the conclusion can be obtained from Proposition 2.

Since δ_1 and δ_2 are interference-free, we have that $(\delta_1 \oplus \delta_2)^f = \delta_1^f \oplus \delta_2^f$ and $\gamma \ominus (\delta_1 \oplus \delta_2)^f = \gamma \ominus (\delta_1^f \oplus \delta_2^f)$. As $\gamma \ominus (\delta_1^f \oplus \delta_2^f) \subseteq \gamma \ominus \delta_1^f$, we obtain that $\gamma \ominus \delta_1^f \preccurlyeq \gamma \ominus (\delta_1^f \oplus \delta_2^f)$ and $\gamma \ominus \delta_1^f \oplus \delta_1 \preccurlyeq \gamma \ominus (\delta_1^f \oplus \delta_2^f) \oplus \delta_1$. Moreover, $\delta_2 \cap \delta_1^f = \emptyset$ and $\gamma \preccurlyeq \delta_2$, and thus $\gamma \ominus \delta_1^f \preccurlyeq \delta_2$. So $\gamma \ominus \delta_1^f \oplus \delta_1 \preccurlyeq \gamma \ominus (\delta_1^f \oplus \delta_2^f) \oplus \delta_1 \oplus \delta_2$. The same rule can be applied to $\delta_2\gamma$. Therefore, we have $\delta_1\gamma \preccurlyeq (\delta_1 \oplus \delta_2)\gamma$ and $\delta_2\gamma \preccurlyeq (\delta_1 \oplus \delta_2)\gamma$, thus $inv((\delta_1 \oplus \delta_2)\gamma(B), \Phi_1 \wedge \Phi_2)$. □

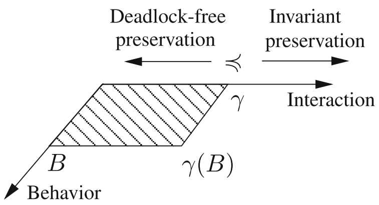

Fig. 4 Invariant preservation for looser synchronization relation

The above proposition extends to 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 by 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 4 shows system construction in a space of two dimensions: $Behavior \times Interactions$, for the refinement relation between behaviors and the preorder relation between interactions. We shall see that the looser synchronization preorder along the interaction axis preserves invariants (Proposition 4). This means that the preorder preserves reachability properties when new interactions are enforced (shown by the right arrow in Fig. 4). On the other hand, the preorder does not preserve deadlock-freedom. Indeed, adding new interactions may lead to the addition of new deadlocks. Given two connectors γ_1 and γ_2 over component B such that γ_2 is tighter than γ_1, 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 Although by using invariant preservation results we can infer that invariants of constituents are also invariants of a composite system, not all the computed invariants of the latter are conjunctions of invariants of the constituents. Consider the example given in Fig. 2 and let $\gamma = \sum_{i=1}^{4} p_i \oplus q_i$, $\delta_1 = p_1\, p_3 \oplus q_1\, q_3$, and $\delta_2 = p_2\, p_4 \oplus q_2\, q_4$. By using the technique presented in the next section, we shall see that a global invariant for $\delta_1\gamma(B)$ is

$$(l_0 + l_1 + l_2)(l_3 + l_4)(l_5 + l_6)(l_1 + l_3)(l_0 + l_2 + l_4).$$

Similarly, a global invariant for $\delta_2\gamma(B)$ is

$$(l_0 + l_1 + l_2)(l_3 + l_4)(l_5 + l_6)(l_2 + l_5)(l_0 + l_1 + l_6).$$

By applying Proposition 4, we obtain that the conjunction of these two invariants is preserved for $(\delta_1 \oplus \delta_2)\gamma(B)$:

$$(l_0 + l_1 + l_2)(l_3 + l_4)(l_5 + l_6)(l_1 + l_3)(l_2 + l_5)\\(l_0 + l_2 + l_4)(l_0 + l_1 + l_6).$$

Nevertheless, this invariant is less strong than the invariant

$$(l_0 + l_1 + l_2)(l_3 + l_4)(l_5 + l_6)(l_1 + l_3)(l_2 + l_5)\\(l_0 + l_2 + l_4)(l_0 + l_1 + l_6)(l_0 + l_4 + l_6)$$

that is directly computed on $(\delta_1 \oplus \delta_2)\gamma(B)$.

4 Efficient computation of interaction invariants

In this section, we will propose two new and efficient techniques to compute globally the interaction invariant of a composite component and propose heuristics to speed up the two global methods.

All the techniques are based on the constraint description of interactions over local components, which is called *Boolean Behavioral Constraints*. The effect of an interaction on a local view is encoded by the implication relation between the locations being one of the preconditions to trigger the interaction and the reachable locations in the involved components by executing the interaction.

The first global technique computes the solutions of all the constraints. The crux of this technique is to transfer the implications in disjunctive normal form (DNF) and take the dual of the final results which only keep positive form of location variables to generate interaction invariants. The name of this method (positive mapping) comes from the mapping on positive location variables in the computation.

The second global technique transfers the constraints in the form of implications to equations. For all the locations, it traces the effect of these equations on the locations and the chain of reachable locations through executing the interactions, until a fixpoint is reached.

By considering incremental construction and reasoning the relationship between interactions from different constituents, we observe that locations involved in the interactions of different constituents play a key role in invariant computation. The two techniques to enhance scalability are adapted on the analysis of these locations in the forms of the two global techniques. The basic idea is to reuse the already established results from constituents and compute the new invariants caused by the integrating of constituents.

The organization of this section is described as follows:

- in Sect. 4.1, the positive mapping technique allows to characterize all interaction invariants by a global symbolic manipulation of the set of Boolean behavioral constraints,
- in Sect. 4.2, the fixpoint-based technique allows to spread the global symbolic computation on the set of locations and interactions,
- in Sect. 4.3, the incremental positive mapping technique is an extension of the positive mapping technique which takes incremental construction into account,

- finally, in Sect. 4.4, the incremental fixpoint technique is an extension of the fixpoint technique which takes incremental construction into account.

We first introduce *Boolean Behavioral Constraints* (BBC). As we shall see in Sects. 4.1 and 4.2, solutions of BBCs can be used to symbolically compute interaction invariants. BBCs highly characterize the effect of interactions of a composite component on the behavior of each one of its constituents by the implications between Boolean location variables. The effect of an interaction starts from a location whose outgoing transition is labeled by some port in the interaction, to the set of local locations that can be reached by triggering the interaction and tracing the involved transitions.

Definition 15 (*Boolean behavioral constraints (BBCs)*) Let γ be a connector over a set of n components $B = (B_1, \ldots, B_n)$ with $B_i = (L_i, P_i, \mathcal{T}_i)$ for $i \in [1, n]$ and $L = \bigcup_{i=1}^{n} L_i$. The Boolean behavioral constraints for component $\gamma(B)$ are given by the function $|\cdot| : \gamma(B) \rightarrow Bool[L]$ such that

$$|\gamma(B)| = \bigwedge_{a\in\gamma} |a(B)|, \ |a(B)|$$

$$= \bigwedge_{l\in L \wedge l_a^{\bullet} \neq \emptyset} \left(l \Rightarrow \bigwedge_{X \in l_a^{\bullet}} \left(\bigvee_{\tau \in X \wedge l' = \tau^{\bullet}} l' \right) \right). \tag{2}$$

For the simplicity of notations, we use $\sigma_l(a)$ to represent the right side of the implication $\bigwedge_{X\in l_a^{\bullet}}(\bigvee_{\tau\in X\wedge l'=\tau^{\bullet}} l')$ in (2). If $\gamma = \emptyset$, then $|\gamma(B)| = true$, which means that no interactions between the components of B will be considered.

As using $\wedge$ and $\vee$ in BBCs increases the length of the representations, in the following examples, we use $+$ instead of $\wedge$ and omit $\vee$ for a succinct representation of computations.

Example 8 Consider the components given in Fig. 2. To obtain the BBC $|(p_1 p_3)(B)|$, we first need to compute the forward interaction sets of interaction $p_1 p_3$ for every location in the components of Fig. 2. According to Definition 4, we have that $l_0{}^{\bullet}_{p_1 p_3} = l_3{}^{\bullet}_{p_1 p_3} = \{\{\tau_{21}, \tau_{11}\}\}$, and the forward interaction sets for other locations are empty. Therefore, we have

$$|(p_1 p_3)(B)| = (l_0 \Rightarrow l_1 + l_4)(l_3 \Rightarrow l_1 + l_4).$$

Consider the example with a non-deterministic component in Fig. 3. The forward interaction sets of interaction $p_1 p_2$ for involved locations are $l_1{}^{\bullet}_{\gamma} = l_3{}^{\bullet}_{\gamma} = \{\{\tau_{11}, \tau_{21}\}, \{\tau_{11}, \tau_{22}\}\}$. We have the BBCs:

$$|(p_1 p_2)(B)| = (l_1 \Rightarrow (l_2 + l_4)(l_2 + l_5))$$
$$(l_3 \Rightarrow (l_2 + l_4)(l_2 + l_5)).$$

Roughly speaking, one implication $l \Rightarrow \sigma_l(a)$ in BBCs describes a constraint on l that is restricted by an interaction of γ issued from l. We will now show how BBCs can be used to compute interaction invariants.

4.1 Positive mapping-based interaction invariant computation

We show how interaction invariants can be computed directly from the solutions of BBCs. Given a solution defined as an assignment of Boolean values to locations, an interaction invariant is the disjunction of the locations assigned to *true*. The method consists in putting a BBC into DNF and taking for each monomial the disjunction of its positive variables. Due to this character, we call this technique computation by Positive Mapping (PM).

The following example illustrates the first step of the method.

Example 9 Consider the components given in Fig. 2 and the following connector $\gamma = p_1 p_3 \oplus p_2 p_4 \oplus q_1 q_3 \oplus q_2 q_4$. The BBC $|(p_1 p_3)(B)|$, $|(p_2 p_4)(B)|$, $|(q_1 q_3)(B)|$, $|(q_2 q_4)(B)|$ are, respectively, given by:

$$|(p_1 p_3)(B)| = (l_0 \Rightarrow l_1 + l_4)(l_3 \Rightarrow l_1 + l_4) = \bar{l_0}\bar{l_3} + l_1 + l_4,$$
$$|(p_2 p_4)(B)| = (l_0 \Rightarrow l_2 + l_6)(l_5 \Rightarrow l_2 + l_6) = \bar{l_0}\bar{l_5} + l_2 + l_6,$$
$$|(q_1 q_3)(B)| = (l_1 \Rightarrow l_0 + l_3)(l_4 \Rightarrow l_0 + l_3) = \bar{l_1}\bar{l_4} + l_0 + l_3,$$
$$|(q_2 q_4)(B)| = (l_2 \Rightarrow l_0 + l_5)(l_6 \Rightarrow l_0 + l_5) = \bar{l_2}\bar{l_6} + l_0 + l_5.$$

The BBC for $\gamma(B)$ is

$$\begin{aligned}|\gamma(B)| &= |(p_1 p_3)(B)| \wedge |(q_1 q_3)(B)| \wedge |(p_2 p_4)(B)| \wedge |(q_2 q_4)(B)| \\ &= \bar{l_0}\bar{l_1}\bar{l_2}\bar{l_3}\bar{l_4}\bar{l_5}\bar{l_6} + \bar{l_0}\bar{l_1}\bar{l_2}\bar{l_3}\bar{l_4}l_2 l_5 + \bar{l_0}\bar{l_1}\bar{l_3}\bar{l_4}l_5 l_6 \\ &\quad + \bar{l_0}\bar{l_2}\bar{l_5}\bar{l_6}l_1 l_3 + \bar{l_0}\bar{l_2}\bar{l_5}\bar{l_6}l_3 l_4 + l_0 l_1 l_2 + l_0 l_1 l_6 + l_1 l_2 l_3 l_5 \\ &\quad + l_1 l_3 l_5 l_6 + l_0 l_2 l_4 + l_0 l_4 l_6 + l_2 l_3 l_4 l_5 + l_3 l_4 l_5 l_6.\end{aligned}$$

Theorem 2 *Let γ be a connector over a set of n components $B = (B_1, \ldots, B_n)$ with $B_i = (L_i, P_i, \mathcal{T}_i)$ for $i \in [1, n]$ and $L = \bigcup_{i=1}^{n} L_i$, and $v : L \rightarrow \{true, false\}$ 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 inductive invariant of $\gamma(B)$.*

Proof According to Definition 15, a BBC is the conjunction of all the implications for interactions of γ. Consider a valuation v such that $|\gamma(B)|(v) = true$. In order to prove that $\bigvee_{v(l)=true} l$ is an invariant, assume that for some global state $(l_1, \ldots, l_n)$, there exists l_i such that $v(l_i) = true$. If from l_i there is an interaction a such that there exists $p_i \in a$ and $l_i \xrightarrow{p_i} l'_i$, then $l_i{}^{\bullet}_a$ is not empty. For every set of transitions $X \in l_i{}^{\bullet}_a$, there exists $\tau \in X$ such that $l'_j = \tau^{\bullet}$ and $v(l'_j) = true$ by Definition 15. So any successor state of $(l_1, \ldots, l_n)$ by an interaction a satisfies $\bigvee_{v(l)=true} l$. □

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

From Theorem 2, interaction invariants are derived as disjunction of positive variables of solutions of $|\gamma(B)|$. This suggests that all the literals with negations should be removed. In some cases, we may keep some negations. The following definition explains how to partially remove negations.

Definition 16 (*positive mapping*) Consider two sets of variables L and L' such that $L' \subseteq L$, and a Boolean formula $\phi = (\bigwedge_{l_i \in L} l_i \wedge \bigwedge_{l_j \in L'} \bar{l_j} \wedge \bigwedge_{l_k \in L-L'} \bar{l_k})$ on L. We define the *Positive Mapping* operation of ϕ, denoted by $\phi^{p(L')}$, by deleting all the negative variables of ϕ that do not belong to L' as follows:

$$\phi^{p(L')} = \bigwedge_{l_i \in L} l_i \wedge \bigwedge_{l_j \in L'} \bar{l_j}.$$

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

Remark 2 Positive mapping is distributive over disjunction. That is, given two Boolean formulas ϕ_1 and ϕ_2 over L, and $L' \subseteq L$, we have:

$$(\phi_1 \vee \phi_2)^{p(L')} = \phi_1^{p(L')} \vee \phi_2^{p(L')}.$$

Example 10 Given a Boolean formula $\phi = l_1 l_2 \bar{l_3} + l_1 \bar{l_2} l_3$ over $L = \{l_1, l_2, l_3\}$ and a subset of variables $L' = \{l_1, l_2\}$, we have positive mapping $\phi^{p(L')} = l_1 l_2 + l_1 \bar{l_2} l_3$ and $\phi^p = l_1 l_2 + l_1 l_3$.

Consider a Boolean formula ϕ_L over a set of variables $L = \{l_1, \ldots, l_n\}$. We denote the dual operation on ϕ_L by $\mathfrak{d}(\phi_L)$. $\mathfrak{d}(\phi_L) = \overline{\phi_{\bar{L}}}$, where $\phi_{\bar{L}}$ is a Boolean formula obtained from ϕ_L by replacing, for each variable $l_i \in L$, its positive form l_i (respectively its negative form $\bar{l_i}$) by its negative form $\bar{l_i}$ (respectively its positive form l_i).

Example 11 The dual of the Boolean formula $\phi = l_1 \bar{l_2} + l_2 \bar{l_3} + l_1 l_3$ over $L = \{l_1, l_2, l_3\}$ is given by $\mathfrak{d}(\phi) = l_1 \bar{l_2} + l_1 \bar{l_3}$.

The following theorem allows to compute an interaction invariant that combines all the solutions of a BBC. 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 take the dual of the BBC.

Theorem 3 *For any connector γ applied to a tuple of n components $B = (B_1, \ldots, B_n)$, a global interaction invariant of $\gamma(B)$ can be obtained as the dual of $|\gamma(B)|^p$, denoted by $\mathfrak{d}(|\gamma(B)|^p)$.*

Proof $|\gamma(B)|$ can be rewritten in the disjunctive normal form, that is $|\gamma(B)| = \bigvee_{i \in I} m_i$, where I is the set of indexes of monomials in $|\gamma(B)|$ and m_i is of the form $m_i = \bigwedge_{j \in I} l_j \wedge \bigwedge_{k \in I \wedge k \neq j} \overline{l_k}$. According to Theorem 2, for any solution m_i of $|\gamma(B)|$, we have $\mathfrak{d}(m_i^p) = \bigvee_{j \in I} l_j$ is an invariant of $\gamma(B)$. Hence $\mathfrak{d}(|\gamma(B)|^p) = \mathfrak{d}((\bigvee_{i \in I} m_i)^p) = \mathfrak{d}(\bigvee_{i \in I} m_i^p) = \bigwedge \mathfrak{d}(m_i^p)$ is the global interaction invariant of $\gamma(B)$. □

Example 12 We consider the components, connectors and BBCs introduced in Example 9. The positive mapping removes variables with negations from $|\gamma(B)|$. We obtain

$$\mathfrak{d}(|\gamma(B)|^p) = (l_0 + l_1 + l_2)(l_0 + l_2 + l_4)(l_3 + l_4)(l_5 + l_6)$$
$$(l_0 + l_1 + l_6)(l_0 + l_4 + l_6)(l_1 + l_3)(l_2 + l_5)$$

which is the global interaction invariant of $\gamma(B)$.

4.2 Fixpoint-based computation of interaction invariants

Interaction invariants can also be iteratively computed by using a fixpoint-based technique. BBC can be regarded as a set of implications $l \Rightarrow \sigma_l(\gamma)$, where $\sigma_l(\gamma) = \bigwedge_{X \in l^\bullet_\gamma} (\bigvee_{\tau \in X \wedge l' = \tau^\bullet} l')$. Instead of unfolding the implication, this set can be equivalently written as a set of equations $\Delta_\gamma = \{l = l \wedge \sigma_l(\gamma)\}$. We call $l = l \wedge \sigma_l(\gamma)$ as BBC-equations in the rest of the paper. We also use σ_l instead of $\sigma_l(\gamma)$ and Δ instead of Δ_γ when we consider all the interactions in γ.

Example 13 Consider again Example 9. The BBC-equations in Δ for the locations of each component are:

$$\Delta = \{l_0 = l_0(l_1 + l_4)(l_2 + l_6),\ l_1 = l_1(l_0 + l_3),\ l_2 = l_2(l_0 + l_5),$$
$$l_3 = l_3(l_1 + l_4),\ l_4 = l_4(l_0 + l_3),\ l_5 = l_5(l_2 + l_6),\ l_6 = l_6(l_0 + l_5)\}$$

We now show how to compute interaction invariants by using BBC-equations. The intuition is as follows. For the equation $l = l \wedge \sigma_l$, if l' occurs in σ_l and $l' \wedge \sigma_{l'}$ is the BBC-equation of l', then we can apply $l' \wedge \sigma_{l'}$ to replace l' in $l \wedge \sigma_l$ and obtain an equation that represents the locations that can be reached from l via l'. If we repeat the same operation to all the locations until no more reachable locations are added, then every monomial in the right side of the obtained equation for a given location is a set of reachable locations through interactions started from this location. In this subsection, we present a method to compute the solutions for BBC-equations and the way to obtain interaction invariants from these solutions.

We extend Δ to formulas in $Bool[L]$, that is, for a formula ϕ over L, $\Delta(\phi) = \phi[l \mapsto \Delta(l)]$ where $l \mapsto \Delta(l)$ is the substitution of l by $\Delta(l)$. Then we can apply the fixpoint computation $\phi^{k+1} = \Delta(\phi^k)$, starting from $\phi^0 = l$ for every location $l \in L$. When $\phi^{k+1} = \phi^k$, the computation terminates and ϕ^k is the solution (fixpoint) of l with respect to the BBC-equations Δ.

The termination of the fixpoint computation comes from the following reasons. First, L is a finite set and the number of formula over a finite domain $Bool[L]$ is finite. Second, we have $\Delta(\phi) \Rightarrow \phi$, therefore, $\phi^j \Rightarrow \phi^i$ with $j > i$. Assume by contradiction that there exist ϕ^i and ϕ^j such that $\phi^i = \phi^j$ and $j > i+1$. Then we have $\phi^i = \phi^{i+1} = \cdots = \phi^j$ and the iteration stops at ϕ^i.

For a set of locations, we can compute their solutions simultaneously. Since the method is based on the least fixpoint computation by considering BBC-equations of all the locations, we call it location-based fixpoint (LFP). We use $\mathbb{L}^k$ to denote the set $\{l^k\}_{l\in L}$ after k iterations.

Example 14 (LFP *computation*) To illustrate the application of the fixpoint computation in computing the solutions of every location with respect to the BBC-equations, we continue Example 13 and consider again the components given in Fig. 2. In the following table, a column corresponds to an iteration. When $\mathbb{L}^3 = \mathbb{L}^2$, the iteration stops.

$\mathbb{L}$	$\mathbb{L}^1 = \Delta(\mathbb{L})$	$\mathbb{L}^2 = \Delta(\mathbb{L}^1)$	$\mathbb{L}^3 = \Delta(\mathbb{L}^2)$
l_0	$l_0l_1l_2 + l_0l_2l_4$ $+ l_0l_4l_6 + l_0l_1l_6$	$l_0l_1l_2 + l_0l_2l_4$ $+ l_0l_4l_6 + l_0l_1l_6$	$l_0l_1l_2 + l_0l_2l_4$ $+ l_0l_4l_6 + l_0l_1l_6$
l_1	$l_0l_1 + l_1l_3$	$l_0l_1l_2 + l_1l_3 + l_0l_1l_6$	$l_0l_1l_2 + l_1l_3 + l_0l_1l_6$
l_2	$l_0l_2 + l_2l_5$	$l_0l_1l_2 + l_2l_5 + l_0l_2l_4$	$l_0l_1l_2 + l_2l_5 + l_0l_2l_4$
l_3	$l_1l_3 + l_3l_4$	$l_1l_3 + l_3l_4$	$l_1l_3 + l_3l_4$
l_4	$l_0l_4 + l_3l_4$	$l_0l_2l_4 + l_3l_4 + l_0l_4l_6$	$l_0l_2l_4 + l_3l_4 + l_0l_4l_6$
l_5	$l_5l_6 + l_2l_5$	$l_5l_6 + l_2l_5$	$l_5l_6 + l_2l_5$
l_6	$l_5l_6 + l_0l_6$	$l_0l_4l_6 + l_5l_6 + l_0l_1l_6$	$l_0l_4l_6 + l_5l_6 + l_0l_1l_6$

Observe that the result of the above fixpoint computation differs from the set of reachable states of the composite component. The set of reachable states is in fact more accurate than our computation, but this precision has a cost when working with complex systems. Reachable states computation takes interaction as a global transition and computes exactly the successor states from each global state that enables the synchronization. However, in our method, what we need is only one location that can be reached by an interaction.

Proposition 5 *Let γ be a connector over B with a set of locations L and let $l^k = \bigvee_{i\in I} m_i$ be the solution of $l \in L$ with respect to BBC-equations Δ of $\gamma(B)$ where I is the set of indexes of monomials in l^k. Then m_i gives a minimal set of reachable locations through a sequence of interactions of γ via location l, where m_i is a monomial with the conjunction of locations.*

Proof We first show that m_i gives a set of reachable locations from l. Indeed, m_i contains at least one monomial in σ_l, which enumerates one reachable location for every interaction from l. If σ_l contains some l', then m_i also contains those locations in $\sigma_{l'}$. So m_i is a set of reachable locations from l.

We now show that m_i is minimal. Assume by contradiction that m_i is not minimal, i.e., that there exists m_j such that $m_i \Rightarrow m_j$ and m_j is a solution of l^j. Since both m_i and m_j are conjunctions of locations, we have $m_i \vee m_j = m_j$. Therefore, the iteration stops when m_j is generated and m_i is not a solution of l^j generated by the method. This is a contradiction. □

Theorem 4 *Let γ be a connector over B with a set of locations L and let $l^k = \bigvee_{i\in I} m_i$ be the solution of $l \in L$ with respect to BBC-equations Δ of $\gamma(B)$ where I is the set of indexes of monomials in l^k. Then $\eth(l^k) = \bigwedge_{i\in I} \eth(m)_i$ is an inductive invariant, and $\bigwedge_{l\in L} \eth(l^k)$ is an inductive invariant of $\gamma(B)$.*

Proof Consider $m \in l^k$. According to Proposition 5, m gives a minimal set of reachable locations through interactions of γ via l. Assume also that for some global state $l = (l_1, \ldots, l_n)$, we have $l_i \in m$. This means that $\eth(m)$ is true. If from l_i there exists an interaction $a \in \gamma$ such that $l_i \in {}^\bullet a$, then there exists $l_i' \in a^\bullet$ such that l_i' belongs to m and $\eth(m)$ is still true. So any successor state of l by an interaction a satisfies $\eth(m)$ and $\eth(m)$ is an invariant of $\gamma(B)$.

Since the conjunction of invariants is still an invariant, $\eth(l^k) = \bigwedge_{i\in I} \eth(m)_i$ is an invariant of $\gamma(B)$, and $\bigwedge_{l\in L} \eth(l^k)$ is also an invariant of $\gamma(B)$. □

This theorem allows us to compute the global interaction invariants of $\gamma(B)$, which is the conjunction of the invariants obtained by taking the dual of each solution of the equations.

Example 15 According to Theorem 4, the dual over the set of solutions of a BBC-equation is an invariant. In Example 14, we have

$$\eth(l_0^2) = (l_0+l_1+l_2)(l_0+l_2+l_4)(l_0+l_4+l_6)(l_0+l_1+l_6)$$

is an invariant of the composite component in Fig. 2, and

$$\bigwedge_{i=0}^{6} \eth(l_i^2) = (l_0+l_1+l_2)(l_0+l_2+l_4)(l_1+l_3)(l_2+l_5)$$
$$(l_0+l_4+l_6)(l_0+l_1+l_6)(l_3+l_4)(l_5+l_6)$$

is also an invariant.

Finally, instead of computing solutions from different locations simultaneously, we can put the locations together by disjunction and compute their fixpoint.

Remark 3 Let γ be a connector over B with a set of locations L, Δ be a set of BBC-equations over L, and ϕ^k be the set of solutions of $\phi = \bigvee_{l\in L} l$ with respect to the BBC-equations Δ of $\gamma(B)$. The interaction invariant of $\gamma(B)$ can be obtained as the dual of ϕ^k. We call ϕ^k the set of solutions for Δ.

Example 16 Consider again the example given in Fig. 2, with $\gamma = p_1\, p_3 \oplus q_1\, q_3 \oplus p_2\, p_4 \oplus q_2\, q_4$. We have the initialization $\phi^0 = l_0 + l_1 + l_2 + l_3 + l_4 + l_5 + l_6$, and Δ is the same as that in Example 14. According to Theorem 3,

$$\begin{aligned}\phi^1 &= \Delta(\phi^0) = l_0l_1 + l_1l_3 + l_0l_2\\ &\quad + l_2l_5 + l_3l_4 + l_0l_4 + l_5l_6 + l_0l_6.\\ \phi^2 &= \Delta(\phi^1) = l_0l_1l_2 + l_0l_2l_4 + l_1l_3 + l_2l_5 + l_0l_4l_6\\ &\quad + l_0l_1l_6 + l_3l_4 + l_5l_6.\end{aligned}$$

Since $\phi^3 = \phi^2$, the iteration stops and, according to our definition, ϕ^2 is the set of solutions for the BBC-equations of $\gamma(B)$. We have

$$\begin{aligned}\eth(\phi^2) = {} & (l_0 + l_1 + l_2)(l_0 + l_2 + l_4)(l_1 + l_3)(l_2 + l_5)\\ & (l_0 + l_4 + l_6)(l_0 + l_1 + l_6)(l_3 + l_4)(l_5 + l_6)\end{aligned}$$

is an invariant of $\gamma(B)$.

4.3 Incremental computation of interaction invariants based on positive mapping method

We show how to reuse already computed invariants when new increments are added to a component. The intuition behind is that we regard the system as a composition of subsystems. We first give a decomposition form for BBC and then show how this decomposition can be used to save computation time.

Lemma 3 *Consider two connectors γ_1, γ_2 over B. We have*

$$|(\gamma_1 \oplus \gamma_2)(B)| = |\gamma_1(B)| \wedge |\gamma_2(B)|.$$

Proof By Definition 15, we have $|(\gamma_1 \oplus \gamma_2)(B)| = \bigwedge_{a \in (\gamma_1 \oplus \gamma_2)} |a(B)| = \bigwedge_{a \in \gamma_1} |a(B)| \wedge \bigwedge_{a \in \gamma_2} |a(B)| = |\gamma_1(B)| \wedge |\gamma_2(B)|$. □

Proposition 6 *Let γ be a connector over B. The Boolean behavioral constraint for the composite component obtained by superposition of n increments $\{\delta_i\}_{1 \le i \le n}$ of γ can be written as*

$$\left|\left(\sum_{i=1}^{n} \delta_i\right)\gamma(B)\right| = \left|\left(\gamma \ominus \left(\sum_{i=1}^{n} \delta_i\right)^f\right)(B)\right| \wedge \bigwedge_{i=1}^{n} |\delta_i(B)|. \tag{3}$$

Proof By (1), the union of $\gamma \ominus (\sum_{i=1}^{n} \delta_i)^f$ and $\sum_{i=1}^{n} \delta_i$ is the result of the superposition of a set of increments $\{\delta_i\}_{1 \le i \le n}$ over γ. Therefore, by applying Lemma 3, we have $|(\sum_{i=1}^{n} \delta_i)\gamma(B)| = |(\gamma \ominus (\sum_{i=1}^{n} \delta_i)^f \oplus \sum_{i=1}^{n} \delta_i)(B)| = |(\gamma \ominus (\sum_{i=1}^{n} \delta_i)^f)(B)| \wedge \bigwedge_{i=1}^{n} |\delta_i(B)|$. □

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, $\gamma \ominus (\sum_{i=1}^{n} \delta_i)^f$ is represented by δ_0.

We now show how to exploit incremental design to speed up the positive mapping method presented in Sect. 4.1.

The positive mapping method considers the BBCs from interactions. It is natural to apply Proposition 6 to reuse invariants computed from the increments for the whole connector.

First, we switch to the problem of computing invariants while taking incremental design into account. Different increments may interfere over same components or locations. Therefore, we should consider their relations and the effect on invariants when they are superposed. We propose the following definition that will help in the process of reusing existing invariants.

Definition 17 (*common location variables L_c*) The set of common location variables of a set of n connectors $\{\gamma_i\}_{1 \le i \le n}$ is defined by:

$$L_c = \bigcup_{i,j \in [1,n] \wedge i \ne j} support(\gamma_i) \cap support(\gamma_j),$$

where $support(\gamma) = \bigcup_{a \in \gamma} {}^{\bullet}a \cup a^{\bullet}$, the set of locations involved in some interaction a of γ.

Assume that an invariant has already been computed for a set of interactions (we use Φ_δ to denote the BBC-invariant of $|\delta(B)|$). This information is exploited to improve the efficiency. According to (1), the superposition of a set of increments $\{\delta_i\}_{1 \le i \le n}$ over a connector γ can be regarded as separately applying increments over their constituents. Interaction invariants can be computed from BBC-invariants by reasoning the relation between location variables involved in BBC-invariants and common location variables. We propose the following proposition, which builds on (3).

Proposition 7 *Consider a composite component $\gamma(B)$. Let assume a set of n increments $\{\delta_i\}_{1 \le i \le n}$ over $\gamma(B)$. Let $\delta_0 = \gamma \ominus (\sum_{i=1}^{n} \delta_i)^f$. Let L_c be the set of common location variables between $\{\delta_0, \delta_1, \ldots, \delta_n\}$. For a formula ϕ, let L_ϕ denote the set of location variables occurring in ϕ. Then:*

- *every BBC-invariant ϕ_i computed for $|\delta_i(B)|$ such that $L_{\phi_i} \cap L_c = \emptyset$, that is ϕ_i does not use any common location variables, is an interaction invariant of $(\sum_{i=1}^{n} \delta_i)\gamma(B)$.*
- *for every set of BBC-invariants $\phi_{i_1}, \ldots, \phi_{i_r}$ computed, respectively, for BBC-solutions $m_{i_1}, \ldots, m_{i_r}$ from $|\delta_{i_1}(B)|, \ldots, |\delta_{i_r}(B)|$ such that:*
  - *(1) $\forall j \in [1, r].\ L_{\phi_{i_j}} \cap L_c \ne \emptyset$, that is each invariant ϕ_{i_j} uses some common location variables,*
  - *(2) $\left(\bigwedge_{j=1}^{r} m_{i_j}\right) \ne false$, that is $\bigwedge_{j=1}^{r} m_{i_j}$ corresponds to a feasible global BBC-solution,*
  - *(3) it is maximal, i.e., it cannot be extended by some other BBC-invariant $\phi_{i_{r+1}}$ such that (1) and (2) still hold,*

$\bigvee_{j=1}^{r} \phi_{i_j}$ *is an interaction invariant of* $(\sum_{i=1}^{n} \delta_i)\gamma(B)$.

Proof For every BBC $|\delta_i(B)|$, there exists a BBC-solution m_{i0} without any variables in the positive form, which has no BBC-invariant corresponding to. For any $\phi_{i_k}, k \in I_i$, there exists m_{i_k} such that $\phi_{i_k} = \mathfrak{d}(m_{i_k})$. According to Proposition 6, the BBC-solution of $|(\sum_{i=1}^{n} \delta_i)\gamma(B)|$ is $\bigwedge_{i=0}^{n} \bigvee_{k \in I_i} m_{i_k} = \bigvee_{k \in \{I_j\}_{j=0}^{n}} \bigwedge_{j=0}^{n} m_{i_k}$ where I_i is the set of indexes of BBC-solutions to $|\delta_i(B)|$.

- If an m_{i_k} does not contain any common location variables, then there exists a BBC-solution m_{j0} containing only negations of $|\delta_j(B)|$ such that:

  $$i \neq j \wedge \left(\bigwedge_{j=0 \wedge j \neq i}^{n} m_{i_k} \wedge m_{j0} \right)^{p} = m_{i_k}^{p}.$$

  This means that ϕ_{i_k} is one of the BBC-invariants of $|(\sum_{i=1}^{n} \delta_i)\gamma(B)|$.
- If there is a maximal set $\{m_{i_1}, \ldots, m_{i_r}\}, \forall j \in [1, r] \wedge i_j \in I_{i_j}$ such that all of them contain common location variables, and $\bigwedge_{j=1}^{r} m_{i_j} = false$, then this set is not a solution of $|(\sum_{i=1}^{n} \delta_i)\gamma(B)|$. If $\bigwedge_{j=1}^{r} m_{i_j} \neq false$, we have:

  $$\mathfrak{d}\left(\left(\bigwedge_{j=1}^{r} m_{i_j}\right)^{p}\right) = \mathfrak{d}\left(\bigwedge_{j=1}^{r} \mathfrak{d}(\phi_{i_j})\right) = \bigvee_{j=1}^{r} \phi_{i_j}.$$

□

The proposition simply says that the BBC-invariants that do not share common variables are also the invariants of $(\sum_{i=1}^{n} \delta_i)\gamma(B)$. Other BBC-invariants that contain common variables need to be combined (by disjunction) together to form a global invariant. This is to guarantee that common location variables will not change the satisfiability of the formula.

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 17 (*incremental invariant computation*) In the example of Fig. 2, let $\gamma = \sum_{i=1}^{4} p_i \oplus q_i$. Two increments over γ are $\delta_1 = p_1\, p_3 \oplus q_1\, q_3$ and $\delta_2 = p_2\, p_4 \oplus q_2\, q_4$. The new connector obtained by applying δ_1 and δ_2 to γ is given by $(\delta_1 \oplus \delta_2)\gamma = p_1\, p_3 \oplus q_1\, q_3 \oplus p_2\, p_4 \oplus q_2\, q_4$. The BBC $|\delta_1(B)|$ and $|\delta_2(B)|$ are, respectively, given by:

$$|\delta_1(B)| = (l_0 \Rightarrow l_1 + l_4)(l_1 \Rightarrow l_0 + l_3)(l_3 \Rightarrow l_1 + l_4)$$
$$(l_4 \Rightarrow l_0 + l_3),$$

$$|\delta_2(B)| = (l_0 \Rightarrow l_2 + l_6)(l_2 \Rightarrow l_0 + l_5)(l_5 \Rightarrow l_2 + l_6)$$
$$(l_6 \Rightarrow l_0 + l_5).$$

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

We show how to compute the invariants from BBC-invariants of the increments. By Definition 17, we obtain that $L_c = \{l_0\}$. We have the BBC-solutions for $|\delta_1(B)|$ and $|\delta_2(B)|$ are:

$$|\delta_1(B)| = \bar{l}_0\bar{l}_1\bar{l}_3\bar{l}_4 + l_0 l_1 + l_1 l_3 + l_0 l_4 + l_3 l_4,$$
$$|\delta_2(B)| = \bar{l}_0\bar{l}_2\bar{l}_5\bar{l}_6 + l_0 l_2 + l_2 l_5 + l_0 l_6 + l_5 l_6.$$

Their corresponding BBC-invariants are:

$$\Phi_{\delta_1} = (l_0 + l_1)(l_0 + l_4)(l_1 + l_3)(l_3 + l_4),$$
$$\Phi_{\delta_2} = (l_0 + l_2)(l_0 + l_6)(l_2 + l_5)(l_5 + l_6).$$

Since $(\delta_1 \oplus \delta_2)\gamma(B) = ((\gamma \ominus (\delta_1 \oplus \delta_2)^f) \oplus \delta_1 \oplus \delta_2)(B)$ and $\gamma \ominus (\delta_1 \oplus \delta_2)^f = \emptyset$, we have $\Phi_{(\delta_1 \oplus \delta_2)\gamma(B)} = \Phi_{(\delta_1 \oplus \delta_2)(B)}$.

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

$$(l_0 + l_1 + l_2)(l_0 + l_4 + l_6)(l_0 + l_1 + l_6)(l_0 + l_2 + l_4)$$
$$(l_1 + l_3)(l_3 + l_4)(l_2 + l_5)(l_5 + l_6).$$

4.4 Incremental computation of interaction invariants based on fixpoint method

According to Proposition 1, the set of interactions $(\sum_{i=1}^{n} \delta_i)\gamma$ can be obtained from the sets of interactions $\gamma \ominus (\sum_{i=1}^{n} \delta_i)^f$ and $\{\delta_i\}_{i=1}^{n}$. From that, we propose a method which allows computing fixpoint of $(\sum_{i=1}^{n} \delta_i)\gamma(B)$ from the fixpoints obtained from $\gamma \ominus (\sum_{i=1}^{n} \delta_i)^f$ and $\{\delta_i\}_{i=1}^{n}$ over B.

First, for a set of increments $\{\delta_i\}_{i=1}^{n}$ over a component $\gamma(B)$, the following proposition allows computing the global BBC-equation of $(\sum_{i=1}^{n} \delta_i)\gamma(B)$ from those of $\delta_i(B)$ for $i \in [0, n]$.

Proposition 8 *Consider a composite component B with a set of locations L. Let* γ *be a connector over B and let* $\{\delta_i\}_{1 \leq i \leq n}$ *be a set of n increments over* γ*. Assume that* $\delta_0 = \gamma \ominus (\Sigma_{i=1}^{n}\delta_i)^f$*, let* $L_\delta = \bigcup_{i=0}^{n} {}^\bullet\delta_i$*, and let* Δ_{δ_i} *be the set of BBC-equation of* $\delta_i(B)$ *for* $i \in [0, n]$*.* $\Delta(l)$*, BBC-equation of* $l \in L$ *for* $(\sum_{i=1}^{n} \delta_i)\gamma(B)$ *is defined by the following form:*

$$\Delta(l) = \begin{cases} \bigwedge_{i=0}^{n} \Delta_{\delta_i}(l) & \mathit{if}\ l \in L_\delta \\ l & \mathit{otherwise} \end{cases} \qquad (4)$$

Proof $\Delta(l) = l \wedge \sigma_l((\sum_{i=1}^{n} \delta_i)\gamma)$. According to Proposition 1, we have $l^\bullet_{(\Sigma_{i=1}^{n}\delta_i)\gamma} = \bigcup_{i=0}^{n} l^\bullet_{\delta_i}$. Therefore, we have $\Delta(l) = \bigwedge_{i=0}^{n}(l \wedge \sigma_l(\delta_i)) = \bigwedge_{i=0}^{n} \Delta_{\delta_i}(l)$. □

The above proposition states that BBC-equations of locations for a superposition of increments can be obtained by taking the conjunction of the corresponding equation for each increment.

We now introduce the incremental computation of solutions that computes the invariant for a composite component and a set of increments. The method, which is called *Incremental Location-based Fixpoint* ($ILFP$), assumes that an invariant has already been computed for a set of interactions. This information is exploited to improve the efficiency. The idea is as follows. According to (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. The incremental computation of solutions is based on the solutions of these increments over their constituents $\delta_i(B)$ and the solutions for the BBC-equations of $(\gamma \ominus (\Sigma_{i=1}^{n}\delta_i)^f)(B)$. We suggest the following proposition.

Proposition 9 (incremental LFP computation) *Consider a composite component B and a set of locations L. Let γ be a connector over B and assume a set of increments $\{\delta_i\}_{1\leq i\leq n}$ over $\gamma(B)$. Let $\delta_0 = \gamma \ominus (\sum_{i=1}^{n}\delta_i)^f$, $L_\delta = \bigcup_{i=0}^{n} {}^\bullet\delta_i$, and ϕ_i be the solution for BBC-equations of $\delta_i(B)$, where $i \in [0, n]$. The solution for the BBC-equations Δ of $(\sum_{i=1}^{n}\delta_i)\gamma(B)$ can be computed by following the iteration:*

$$\phi^0 = \bigvee_{i=0}^{n} \phi_i \vee \bigvee_{l\in L-L_\delta} l,$$

$$\phi^{k+1} = \Delta(\phi^k).$$

Proof Given two sets of monomials S_1, S_2, we denote $S_1 \sqsubseteq S_2$ if for all $s_1 \in S_1$ there exists $s_2 \in S_2$ such that s_2 implies s_1.

By Proposition 8, for $\forall i \in [0, n]$, we have $\Delta_{\delta_i}(l) \sqsubseteq \Delta$ where Δ is the set of BBC-equations for $(\sum_{i=1}^{n}\delta_i)\gamma(B)$. Let l^{k_i} and l^k be, respectively, the fixpoints obtained from l by Δ_{δ_i} and Δ, we have $l \sqsubseteq l^{k_i} \sqsubseteq l^k$, therefore $\phi \sqsubseteq \phi^{k_i} \sqsubseteq \phi^k$. By starting from $\bigvee_{l\in L} l$ and $\bigvee_{i=0}^{n}\Delta_i \vee \bigvee_{l\in L-L_\delta} l$, we have the same least fixpoint over Δ. □

Proposition 9 shows that the invariants computed for the increments (the δ_i) can be reused in other computation where more increments are added. Hence, this invariant can be maintained for further incremental constructions and verifications, which should improve the efficiency of the verification process. Observe that in the case that $l \notin {}^\bullet\gamma$, no outgoing interaction from l will be considered, and it can be regarded as a deadlock location in $\gamma(B)$. As it will not in ${}^\bullet\delta_i$ either, we need to add such locations when we compute the global solution.

We conclude the section with an example.

Example 18 (*incremental LFP computation*) Consider the components given in Fig. 2 and let $\gamma = \sum_{i=1}^{4} p_i \oplus q_i$. Consider also two increments $\delta_1 = p_1\,p_3 \oplus q_1\,q_3$ and $\delta_2 = p_2\,p_4 \oplus q_2\,q_4$ that are defined over γ. Since $\gamma \ominus (\delta_1 \oplus \delta_2)^f = \emptyset$, we have $\delta_0 = \emptyset$. The set of BBC-equations Δ_{δ_1} for $\delta_1(B)$ can thus be defined as follows:

$$\Delta_{\delta_1}(l_0) = l_0(l_1 + l_4),\ \Delta_{\delta_1}(l_1) = l_1(l_0 + l_3),$$
$$\Delta_{\delta_1}(l_3) = l_3(l_1 + l_4),\ \ \Delta_{\delta_1}(l_4) = l_4(l_0 + l_3).$$

The set of BBC-equations Δ_{δ_2} for $\delta_2(B)$ is as follows:

$$\Delta_{\delta_2}(l_0) = l_0(l_2 + l_6),\ \Delta_{\delta_2}(l_2) = l_2(l_0 + l_5),$$
$$\Delta_{\delta_2}(l_5) = l_5(l_2 + l_6),\ \ \Delta_{\delta_2}(l_6) = l_6(l_0 + l_5).$$

The solutions for BBC-equations are

$$\phi_1 = l_0l_1 + l_0l_4 + l_1l_3 + l_3l_4,$$
$$\phi_2 = l_0l_2 + l_0l_6 + l_2l_5 + l_5l_6.$$

For $(\delta_1 + \delta_2)\gamma(B)$, we have $L = {}^\bullet\delta_1 \cup {}^\bullet\delta_2$, so

$$\phi = \phi_1 + \phi_2 = (l_0l_1 + l_0l_4 + l_1l_3 + l_3l_4)$$
$$+(l_0l_2 + l_0l_6 + l_2l_5 + l_5l_6).$$

By applying Δ of $(\delta_1 \oplus \delta_2)\gamma(B)$ over ϕ, we obtain that

$$\phi^1 = \Delta(\phi) = l_0l_1l_2 + l_0l_2l_4 + l_0l_4l_6 + l_0l_1l_6 + l_1l_3$$
$$+l_3l_4 + l_2l_5 + l_5l_6.$$

Then $\phi^2 = \phi^1 = l_0l_1l_2 + l_0l_2l_4 + l_0l_4l_6 + l_0l_1l_6 + l_1l_3 + l_2l_5 + l_3l_4 + l_5l_6$, so ϕ^1 is the solution for the BBC-equations of $(\delta_1 \oplus \delta_2)\gamma(B)$.

5 Implementation

All the techniques presented earlier have been implemented in the D-Finder toolset [17]. As shown in Fig. 5, D-Finder takes as inputs programs written in BIP language. According to the behaviors of local components and interactions between different components, it computes both component and interaction invariants using the enumerative techniques presented in [16, 18] or symbolic techniques presented in this paper. Both the constraints and the computations can be made symbolic by representating sets of locations by BDDs using the CUDD package [69].

In the previous sections, we have focused on abstract finite-state systems without data. The presented techniques can be easily generalized to systems with arbitrary data by applying abstraction techniques (see for instance [18]). Briefly, for systems with data, we apply the following three steps:

1. We compute an abstraction of the system $\mathcal{S} = \langle\gamma(B_1, \ldots, B_n), Init\rangle$ in the following manner:
   (a) for each atomic component B_i with data of the system $\mathcal{S}$, an abstraction is computed to obtain a corresponding abstract atomic component B_i^α without data. This can be done by following the approach introduced in [18].

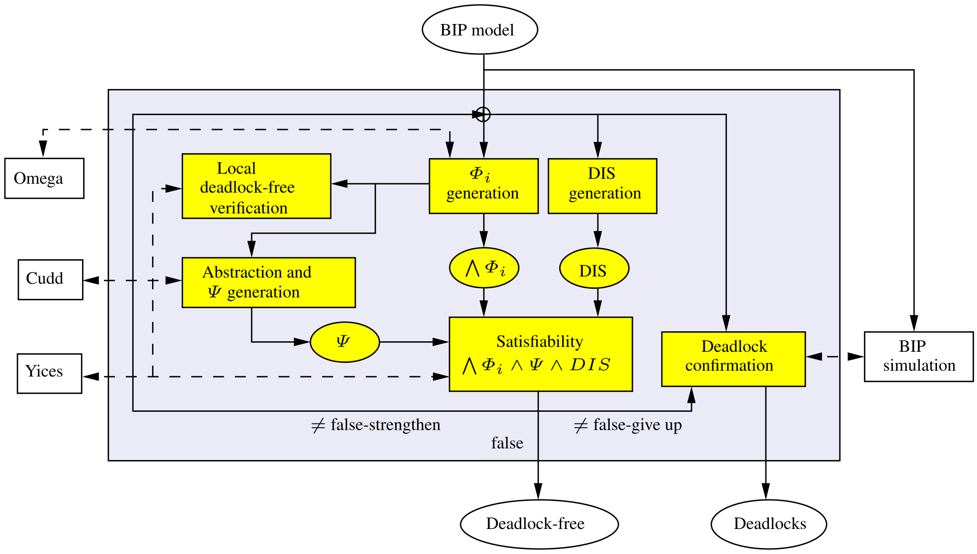

Fig. 5 D-Finder tool

(b) an abstraction γ^α of connector γ. γ^α is obtained by generating for each interaction in γ a corresponding abstract interaction.

(c) abstract initial condition $Init^\alpha$ is obtained from $Init$ as the set of all the abstraction locations such that their corresponding predicates in concrete atomic components satisfy $Init$.

The above method does not change the structure of the system, which allows to switch from the abstract to the concrete domain in an easy way. See [18] for details.

2. The techniques for computing interaction invariants of abstract finite-state systems are applied to the abstract systems S^α. The result is a set of abstract interaction invariants II^α of S^α.
3. Finally, the interaction invariant II for the concrete system S is obtained by concretizing the abstract interaction invariant II^α.

To check deadlock-freedom, we have to show that $\neg DIS$ is an invariant. To check that $\neg DIS$ is an invariant, D-Finder computes the conjunction of component invariants CI with interaction invariant II. Then, checks that $CI \wedge II \Rightarrow \neg DIS$ or equivalently that $CI \wedge II \wedge DIS = false$. For finite-state systems, all these operations can be performed by using BDDs. However, for infinite systems, formulas describe relations between possibly unbounded data. D-Finder needs to concretize predicates back from abstract interaction invariants by recovering the previous formula from BDDs. Then checking $CI \wedge II \wedge DIS = false$ is converted to the satisfiability checking of the formula for $CI \wedge II \wedge DIS$, by using the Yices [44] and Omega [70] toolsets.

We have implemented the presented techniques by using again BDD-based representations. It is easy to see that all the steps of the methods presented earlier can be expressed as Boolean operations on BDDs. The reader who is interested in implementation details is redirected to [64].

For incremental computation of invariants, the way increments are defined may totally change performance. Unfortunately, we could not find rules to determine optimal decompositions.

6 Experimentation

In this section, we present experimental results. We start with a subsection providing benchmarks for classical examples. We then present two non-trivial case studies that are the Utopar Transportation System and the DALA autonomous robot. All our case studies and the D-Finder toolset can be downloaded from: http://www-verimag.imag.fr/dfinder.

6.1 Examples

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

We have checked deadlock-freedom of the following examples: Gas Station [52], Smoker [65], Automatic Teller Machine (ATM) [31], and Producer/Consumer. For Gas Station, we assume that each pump has 10 customers. Hence, if there are 50 pumps in Gas Station, then we have 500 customers and the total number of components including the operator is thus 551. In ATM example, each ATM machine is associated with one user. Therefore, for 10 machines, the total number of components is 22 (including the two components that describe the Bank). Gas Station and Smoker are systems without data, while ATM and Producer/Consumer have integer variables. For the latter, D-Finder first computes finite abstraction for each atomic component following the method presented in [18].

Table 1 collects results on benchmarks with acyclic interaction topology in an open-chain structure. Computation times and memory usages for the application of the four methods on these examples are given in Table 1. In the legend, *scale* is the "size" of examples; *location* denotes the total number of control locations; *interaction* is for the total number of interactions. Computation times are given in minutes. Timeout, i.e., "-" is one hour. Memory usage is given in megabytes (MB). Incremental techniques are always faster than *global* ones.

We can observe that *GFP* is always faster than *GPM*. This is because the size of the components in these four examples is small, and the interactions do not heavily depend on each other. This allows to compute the fixpoint in a few iterations. Furthermore, positive mapping requires to first compute a BDD for the whole set of BBCs (hence considering all the locations), from which the positive variables are extracted by applying, for each location, a cofactor function to the BDD representing all BBCs. Computing the cofactor is often fast, but the repeated application of this function may exceed the time needed for few steps of fixpoint computation. For the incremental verification, *IPM* outperforms *IFP* for all examples, except for Gas Station. The reasons are that (1) incremental design allows to reduce the size of the BDD and thus reduce the overload caused by positive mapping and (2) *IFP* still has to apply the BBC-equations of the entire system. This also explains why *IFP* consumes more memory than *IPM*.

In Table 2, we provide results on checking deadlock-freedom for Dining Philosophers, to compare performance with the enumerative method in D-Finder, and model checker NuSMV.[8] In the table, *NuSMV* stands for results from NuSMV, and *Enum* stands for results from enumerative methods in [16,18]. It is obvious that both time and memory consumption of NuSMV are much higher than those of D-Finder.

Contrary to the other examples, Dining Philosophers has a cyclic topology, which cannot be efficiently managed by *IFP* (this is the only case for which *GPM* was faster than *IFP*). The reason is that for cyclic topologies, one often has to compute interaction invariants with constraints involving all the components in a cycle at the same time. This experiment is the only case that the global enumerative verification is better than the global symbolic methods. The reason is that the number of interaction invariants of this example does not increase exponentially in the size of the systems.

In Fig. 6, we can observe the evolution of the size of the BDDs created for Gas Station and Dining Philosophers from D-Finder and NuSMV. Figure 6a shows for Gas Station that *GPM* consumes more memory than *GFP*. This also gives another justification for the good performances of *GFP*. Without storing all BBC-equations, *IPM* needs less memory than *IFP*. However, the memory occupation for *IFP* is not excessive. Figure 6b shows for Dining Philosophers that *IPM* performs better than the other techniques. In the two cases, however, the size of BDDs in NuSMV increases rapidly to explore the reachable states.

6.2 Case study on Utopar transportation system

We now consider the *Utopar*[9] automated transportation system. This is one of the two main case studies of the European project COMBEST [37]. Utopar can be modeled as the composition of three types of components: (1) autonomous vehicles, called U-cars (UC), (2) a centralized automatic control system, and (3) calling units (CU). U-cars are equipped with a local controller, responsible for handling the U-car sensors and performing various routing and driving computations depending on users' requests. We have analyzed a simplified version of Utopar by abstracting from data exchanged between components. In this version, each U-car is modeled by a component having 7 control locations and 6 integer variables. The automatic control system has 3 control locations and 2 integer variables. Calling units have 2 control locations and no variables. We have successfully proven that the system is deadlock-free. In Table 3, one can see that *IFP* is always faster than *IPM* for this case study.

6.3 Case study on robotic systems

We have also applied our method to a more complex case study that directly comes from an industrial applications. We have been capable of checking safety and deadlock-freedom properties on eight modules of the functional level of the *DALA autonomous robot* [21]. We only briefly present our results. The reader is redirected to [20] for more details. In this example, we encoded both the modules and their communication primitive in BIP, then we checked the absence of deadlock. Finally, we used the BIP tool to generate correct C Code (more than 500,000 lines) to coordinate the modules. Using the BIP workflow, one can claim that this C code does not generate deadlock when coordinating the modules.

[8] We admit that the comparison is unfair for NuSMV. The reason being that D-Finder does not explore the reachable state space.

[9] A succinct description of the Utopar case study can be found at http://www.combest.eu/home/?link=Application2.

Table 1 Comparison for acyclic topologies

Component information			Time (min)				Memory (MB)			
Scale	Location	Interaction	GFP	GPM	IFP	IPM	GFP	GPM	IFP	IPM
Gas Station										
50 pumps	2,152	2,000	0:17	0:50	0:17	0:49	53	48	53	47
100 pumps	4,302	4,000	0:38	2:58	0:52	1:51	65	76	52	47
200 pumps	8,602	8,000	1:34	11:34	1:55	2:26	107	135	65	47
400 pumps	17,202	16,000	5:01	47:38	3:51	5:43	215	270	93	76
500 pumps	21,502	20,000	7:45	–	4:43	7:21	261	–	101	86
600 pumps	25,802	24,000	11:21	–	5:53	9:05	316	–	115	97
700 pumps	30,102	28,000	16:04	–	7:14	11:44	350	–	138	107
Smoker										
300 smokers	907	903	0:06	0:07	0:07	0:07	32	44	11	7
600 smokers	1,807	1,803	0:18	0:13	0:14	0:13	42	46	26	8
1,500 smokers	4,507	4503	0:59	1:38	0:44	0:34	56	65	54	18
3,000 smokers	9,007	9,003	3:43	6:21	1:57	1:14	81	113	86	28
6,000 smokers	18,007	18,003	14:45	27:03	5:57	3:24	248	222	172	55
7,500 smokers	22,507	22,503	22:44	41:38	8:29	4:51	343	270	209	60
9,000 smokers	27,007	27,003	32:52	–	11:36	6:34	468	319	247	96
ATM										
50 machines	1,104	902	3:17	10:49	2:20	1:23	148	81	86	22
100 machines	2,204	1,802	6:50	43:00	6:00	1:57	284	142	271	44
250 machines	5,504	4,002	17:56	–	17:16	4:46	662	–	670	65
350 machines	7,704	6,302	39:35	–	27:54	8:18	937	–	938	77
600 machines	13,204	10,802	–	–	–	24:14	–	–	–	119
Producer/consumer										
2,000 consumers	4,004	4,003	0:27	0:27	0:33	0:31	54	57	16	11
4,000 consumers	8,004	8,003	1:19	1:27	1:18	1:05	110	90	28	20
6,000 consumers	12,004	12,003	2:40	3:01	2:32	2:03	193	126	37	31
8,000 consumers	16,004	16,003	5:20	5:35	4:22	2:33	256	164	40	35
10,000 consumers	20,004	20,003	8:40	8:44	6:12	3:15	369	218	66	56
12,000 consumers	24,004	24,003	11:02	12:06	8:37	5:38	460	257	75	66

Table 2 Comparison between different methods on Dining Philosophers

Component information			Time (min)						Memory (MB)					
Scale	Location	Interaction	NuSMV	Enum	GFP	GPM	IFP	IPM	NuSMV	Enum	GFP	GPM	IFP	IPM
100 philos	600	500	1:32	0:06	22:41	0:13	0:19	0:04	533	34	75	46	32	10
500 philos	3,000	2,500	–	1:51	–	4:01	9:18	0:34	–	55	–	61	60	29
1,000 philos	6,000	5,000	–	7:08	–	17:09	–	2:04	–	90	–	105	–	60
1,500 philos	9,000	7,500	–	19:30	–	39:40	–	3:09	–	126	–	148	–	74
2,000 philos	12,000	10,000	–	28:44	–	–	–	4:14	–	163	–	–	–	96
4,000 philos	24,000	20,000	–	–	–	–	–	8:37	–	–	–	–	–	192
6,000 philos	36,000	30,000	–	–	–	–	–	14:26	–	–	–	–	–	382
9,000 philos	54,000	45,000	–	–	–	–	–	24:16	–	–	–	–	–	581

The modules and their communication primitives provide the following functions: (1) collecting data from the laser sensors (LaserRF), (2) generating an obstacle map (Aspect), (3) navigating using the near diagram approach (NDD), (4) managing the low level robot wheel controller (RFLEX), (5) emulating the communication with an orbiter (Antenna), (6)

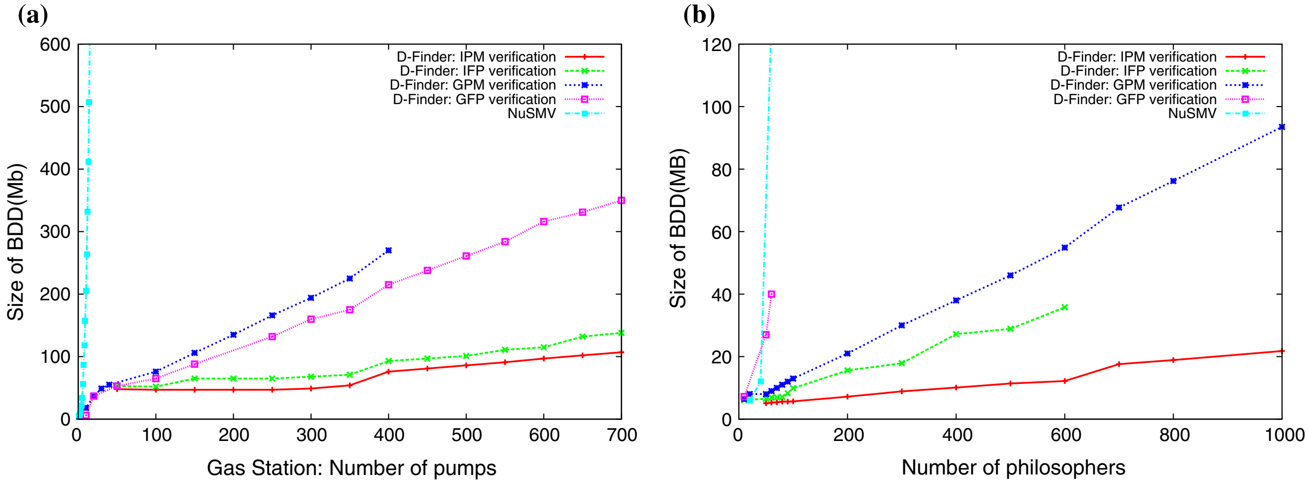

Fig. 6 The comparison on memory consumed by CUDD package. **a** Gas Station, **b** Dining Philosophers

Table 3 Comparison between different methods on the Utopar case study

Component information			Time (min)				Memory (MB)			
Scale	Location	Interaction	GFP	GPM	IFP	IPM	GFP	GPM	IFP	IPM
100 UC, 400 CU	1,503	41,404	0:59	3:35	0:56	2:15	27	50	42	59
200 UC, 400 CU	2,203	82,404	2:15	8:05	1:45	4:13	40	56	42	59
300 UC, 400 CU	2,903	123,404	3:45	13:38	2:29	7:12	52	67	42	59
400 UC, 400 CU	3,603	164,404	6:08	20:32	3:46	8:02	64	79	42	59
100 UC, 900 CU	2,503	91,904	2:47	17:52	2:44	9:56	75	64	66	50
200 UC, 900 CU	3,203	182,904	7:11	38:41	4:59	19:47	117	82	66	50
300 UC, 900 CU	3,903	273,904	–	–	7:18	31:29	–	–	66	50
100 UC, 1600 CU	3,903	162,604	12:02	59:30	5:53	33:02	203	96	160	73
200 UC, 1600 CU	4,603	323,604	–	–	17:46	–	–	–	160	–

providing power and energy for the robot (Battery), (7) heating the robot in a low temperature environment (Heating), and (8) controlling the movement of two cameras (Platine).

We propose the following mapping for the functional level of DALA:

$$\begin{aligned}
&\mathit{Functional\ level} ::= (\mathit{Module})^{+}\\
&\mathit{Module} ::= (\mathit{Execution\ Service})^{+}\\
&\qquad .(\mathit{Execution\ Task})^{+}.(\mathit{Poster})^{+}\\
&\qquad .(\mathit{Control\ Service})^{+}.(\mathit{Interface\ Server})^{+}\\
&\qquad .(\mathit{Lock}).(\mathit{Timer})\\
&\mathit{Execution\ Service} ::= (\mathit{Service\ Controller}).(\mathit{Activity})\\
&\mathit{Execution\ Task} ::= (\mathit{Task\ Controller}).(\mathit{Scheduler})\\
&\qquad .[\mathit{Permanent}].[\mathit{Timer}]\\
&\mathit{Interface\ Server} ::= (\mathit{Message\ Box}).(\mathit{Timer})
\end{aligned}$$

where $^{+}$ (plus) means the presence of one or more subcomponent and . (dot) means the composition of different components and [ ] means the optional components.

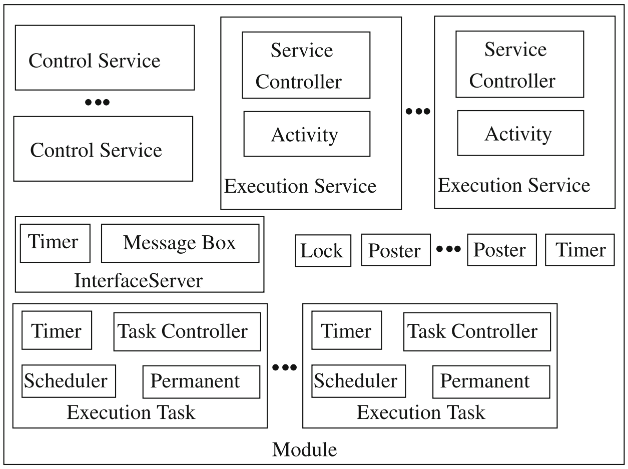

Fig. 7 Module structure in the functional level of DALA

Each module of the functional level of DALA in Fig. 7 is a three-tier composite component with (1) Execution Tasks including a Task Controller which controls to trigger, block,

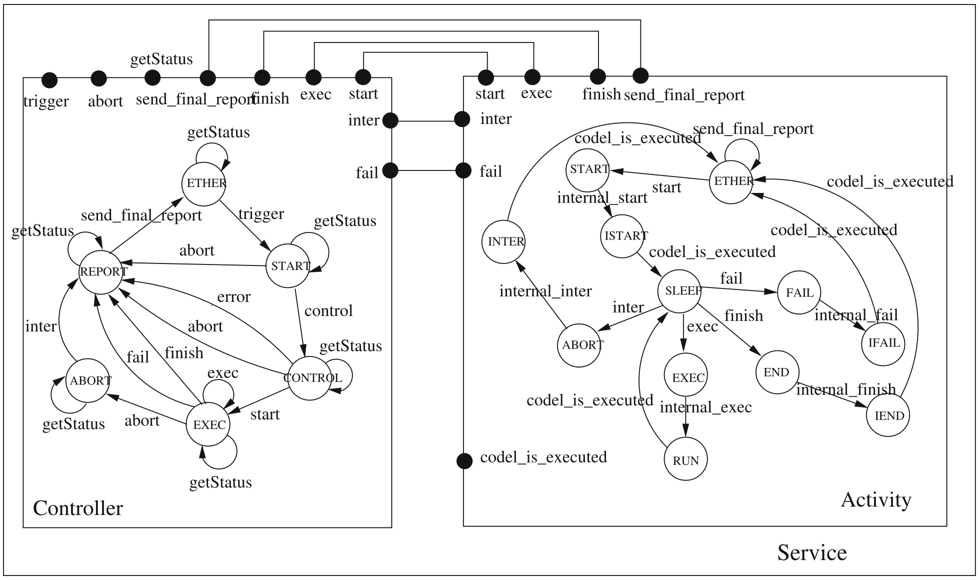

Fig. 8 An execution service in DALA

Table 4 Deadlock-freedom checking on DALA by IPM method

Module	Components	Locations	Interactions	States	Time (min)
LaserRF	43	213	202	2.4×10^{21}	1:22
Aspect	29	160	117	1.2×10^{16}	0:39
NDD	27	152	117	10^{14}	8:16
RFLEX	56	308	227	10^{30}	9:39
Battery	30	176	138	2.7×10^{15}	0:26
Heating	26	149	116	9.1×10^{13}	0:17
Platine	37	174	151	5.8×10^{17}	0:59
LaserRF + Aspect + NDD	97	523	438	2.9×10^{51}	40:57
NDD + RFLEX	82	459	344	10^{44}	73:43

and stop a Service and a Scheduler executes the activities of Services in a cyclic manner, (2) Execution Services, each of which consists of a Service Controller checking the validity of the parameters and the execution of its corresponding activity, and an Activity executing the commands inside the service, (3) Control Services, each of which takes negligible time to execute and is responsible for setting and returning variable values, (4) Interface Server, which is responsible for receiving requests from some external source and then forwarding the requests to the associated service, (5) Posters, which are produced by the corresponding module and can be read by other modules, and (6) Lock, which is a semaphore that ensures the mutual exclusion between different Execution Tasks, Services when manipulating Posters. As there are several modules and most modules contain more than twenty components with complicated interactions, the number of variables required to verify the whole system in the symbolic computation is beyond the capacity of current symbolic methods. So we have to verify the modules involved in certain functionalities to ensure the corresponding correctness.

Each Execution Task and Interface Server have a Timer to control the period of its execution. Also there is a Timer for the posters of a module to control the freshness of the data in the posters.

Observe that the topology of a module in DALA is more complex than those of the other systems we have considered so far. It is well known that an adequate variable ordering can drastically improve performance of symbolic verification. However, the topology is so complex that we cannot always find such an ordering for the computation of invariants in the incremental method. Furthermore, the behavior of components inside a module is much more complex than for the considered examples. In Fig. 8, we present a composite component template for Execution Service. Usually, one module contains several services. And the size of Execution

Task is proportional to the number of services, which results in more location variables.

We first checked deadlock-freedom of individual modules. Both *GPM* and *IFP* failed to check deadlock-freedom for all modules except Antenna that could be checked by using GPM. However, by using *IPM*, we could generate invariants and check the deadlock-freedom of all the modules. Table 4 shows times for computing invariants for checking deadlock-freedom of seven modules by the incremental method. It also gives an estimate of the number of states per module. We have successively detected (and corrected) two deadlocks within Antenna and NDD, respectively. Moreover, using the technique in [18], we could check deadlock-freedom of NDD in several hours. With the proposed incremental methods, we could even verify deadlock-freedom for the composite component including the modules LaserRF, Aspect and NDD, and data-freshness property for the composite component including Aspect and NDD.

Besides checking deadlock-freedom, we have verified for some modules causality properties that is a service can be triggered only after a certain service has completed successfully. Using invariant preservation results introduced in Sect. 3, we removed some tight synchronizations between components[10] that do not synchronize directly with the components involved in the property and obtained a module with looser synchronized interactions. Using invariant preservation results, we could check satisfaction of a causality property in 17 seconds, while it took 1003 seconds for verification on the initial module. A detailed description of DALA and properties verified by combining incremental techniques and invariant preservation results can be found in [19].

7 Conclusion

We present new techniques for computing interaction invariants of composite systems described in BIP. We show how to exploit incremental design and also 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 capabilities of existing tools.

A main advantage of our method is tuning for a particular class of properties that is deadlock-freedom, without enumerating all reachable states. The reason being that interaction invariants can catch well global synchronization which is the cause of global deadlock. To check other safety properties, we need stronger invariants to obtain a better approximation of reachable states. Experimental results show that interaction invariants capture adequately these properties. In contrast to other compositional verification techniques such as Assume/Guarantee techniques, our method scales up smoothly with the complexity of components and is completely automated. Incremental techniques advantageously exploit modularity of hierarchically structured components. Invariant preservation based on the looser synchronization preorder can be used in a more ad hoc manner to cope with complexity as shown for the DALA robot case study.

Interaction invariants are over-approximations of the set of reachable states of the system. When D-Finder detects a deadlock, the reachability of the detected deadlock can be checked automatically [22].

Various other tools can be used to verify concurrent programs (e.g., software model checkers). However, to the best of our knowledge, the ability of BIP and D-Finder to synthesize C code to guarantee that the various components work in a proper way is unique. Also, contrary to existing tools, we take the flow of the design and the hierarchy between the components into account.

There are several directions for future research. As we have seen in Sect. 5, our new techniques 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 characteristics of case study.

Finally, we will extend all our results to the new version of BIP, for real-time systems [1].

References

1. Abdellatif, T., Combaz, J., Sifakis J.: Model-based implementation of real-time applications. In: EMSOFT, pp. 229–238, Scottsdale, AZ, USA (2010)
2. Alfaro, L.D., Henzinger, T.A.: Interface theories for component-based design. In: EMSOFT, pp. 148–165. Springer (2001)
3. Alur, R., Henzinger, T.A.: Reactive modules. Form. Methods Syst. Des. **15**(1), 7–48 (1999)
4. Balarin, F., Lavagno, L., Passerone, C., Sangiovanni-Vincentelli, A., Sgroi, M., Watanabe Y.: Modeling and designing heterogeneous systems. In: Concurrency and Hardware Design, pp. 228–273 (2002)
5. Ball, T., Cook, B., Levin, V., Rajamani, S.K.: SLAM and static driver verifier: Technology transfer of formal methods inside Microsoft. In: IFM, pp. 1–20. Springer (2004)
6. Ball, T., Rajamani, S.K.: The SLAM toolkit. In: CAV, pp. 260–264 (2001)
7. Ball, T., Rajamani S.K.: The SLAM project: debugging system software via static analysis. In: POPL, pp. 1–3 (2002)
8. Basu, A., Bensalem, S., Bozga, M., Bourgos, P., Maheshwari, M., Sifakis, J.: Component assemblies in the context of manycore. In: FMCO, pp. 314–333 (2011)
9. Basu, A., Bensalem, S., Bozga, M., Caillaud, B., Delahaye, B., Legay, A.: Statistical abstraction and model-checking of large heterogeneous systems. In: FMOODS/FORTE, vol. 6117, LNCS, pp. 32–46. Springer (2010)

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

10. Basu, A., Bensalem, S., Bozga, M., Combaz, J., Jaber, M., Nguyen, T.-H., Sifakis, J.: Rigorous component-based system design using the bip framework. IEEE Softw. **28**(3), 41–48 (2011)
11. Basu, A., Bensalem, S., Bozga, M., Delahaye, B., Legay, A., Sifakis, E.: Verification of an afdx infrastructure using simulations and probabilities. In: RV, pp. 330–344. Springer (2010)
12. Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time components in BIP. In: SEFM, pp. 3–12. IEEE Computer Society (2006)
13. Basu, A., Mounier, L., Poulhis, M., Pulou, J., Sifakis, J.: Using BIP for modeling and verification of networked systems—a case study on tinyos-based networks. In: NCA, pp. 257–260 (2007)
14. Bensalem, S., Bozga, M., Delahaye, B., Jegourel, C., Legay, A., Nouri, A.: Statistical model checking qos properties of systems with SBIP. In: ISOLA, pp. 327–341 (2012)
15. Bensalem, S., Bozga, M., Legay, A., Nguyen, T.-H., Sifakis, J., Yan, R.: Incremental component-based construction and verification using invariants. In: FMCAD, pp. 257–265 (2010)
16. Bensalem, S., Bozga, M., Nguyen, T.-H., Sifakis, J.: Compositional verification for component-based systems and application. In: ATVA, pp. 64–79. Springer (2008)
17. Bensalem, S., Bozga, M., Nguyen, T.-H., Sifakis J.: D-Finder: A tool for compositional deadlock detection and verification. In: CAV, pp. 614–619. Springer (2009)
18. Bensalem, S., Bozga, M., Nguyen, T.-H., Sifakis, J.: Compositional verification for component-based systems and application. IET Softw. **4**, 179–235 (2010)
19. Bensalem, S., de Silva, L., Gallien, M., Ingrand, F., Yan, R.: “Rock solid” software: a verifiable and correct by construction controller for rover and spacecraft functional layers. In: ISAIRAS, pp. 859–866 (2010)
20. Bensalem, S., de Silva, L., Griesmayer, A., Ingrand, F., Legay, A., Yan, R.: A formal approach for incremental construction with an application to autonomous robotic systems. In: SC, vol. 6708, pp. 116–132. Springer (2011)
21. Bensalem, S., Gallien, M., Ingrand, F., Kahloul, I., Nguyen, T.-H.: Toward a more dependable software architecture for autonomous robots. IEEE Robot. Autom. Mag. **16**(1), 1–11 (2009)
22. Bensalem, S., Griesmayer, A., Legay, A., Nguyen, T.-H., Peled, D.: Efficient deadlock detection for concurrent systems. In: MEMOCODE, pp. 119–129 (2011)
23. Bensalem, S., Legay, A., Nguyen, T.-H., Sifakis, J., Yan, R.: Incremental invariant generation for compositional design. In: TASE, pp. 157–167 (2010)
24. Beyer, D., Henzinger, T.A., Jhala, R., Majumdar, R.: The software model checker BLAST: applications to software engineering. STTT **9**(5–6), 505–525 (2007)
25. BIP. http://www-verimag.imag.fr/BIP,196.html?
26. Bliudze, S., Sifakis, J.: The algeba of connectors—structuring interaction in BIP. IEEE Trans. Comput. **57**, 1315–1330 (October 2008)
27. Bluespec. http://www.bluespec.com/
28. Bozga, M., Jaber, M., Maris, N., Sifakis, J.: Modeling dynamic architectures using Dy-BIP. In: SC, pp. 1–16. Springer, Berlin (2012)
29. Bozga, M., Sfyrla, V., Sifakis, J.: Modeling synchronous Systems in BIP. In: EMSOFT, pp. 77–86. ACM, October (2009)
30. Chandy, K.M.: Parallel Program Design: A Foundation. Addison-Wesley Longman, Boston (1988)
31. Chaudron, M.R.V., Eskenazi, E.M., Fioukov, A.V., Hammer D.K.: A framework for formal component-based software architecting. In: SAVCBS, pp. 73–80 (2001)
32. Cheng, A., Esparza, J., Palsberg, J.: Complexity results for 1-safe nets. In: FSTTCS, pp. 326–337. Springer, London, UK (1993)
33. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: NuSMV: a new symbolic model checker. Int. J. Softw. Tools Technol. Transf. **2**, 410–425 (2000)
34. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge (1999)
35. Cobleigh, J.M., Avrunin, G.S., Clarke, L.A.: Breaking up is hard to do: an evaluation of automated assume-guarantee reasoning. ACM Trans. Softw. Eng. Methodol. **17**(2), 1–52 (2008)
36. Cobleigh, J.M., Giannakopoulou, D., Pasareanu, C.S.: Learning assumptions for compositional verification. In: TACAS, pp. 331–346 (2003)
37. Combest. http://www.combest.eu/home/
38. Conway, C.L., Namjoshi, K.S., Dams, D., Edwards, S.A.: Incremental algorithms for inter-procedural analysis of safety properties. In: CAV, pp. 449–461. Springer (2005)
39. Cook, B., Podelski, A., Rybalchenko, A.: Terminator: beyond safety. In: CAV, pp. 415–418 (2006)
40. David, A., Larsen, K.G., Legay, A., Nyman, U., Wasowski, A.: Timed i/o automata: A complete specification theory for real-time systems. In: HSCC, pp. 91–100. ACM, New York, NY, USA (2010)
41. David, A., Larsen, K.G., Legay, A., Nyman, U., Wasowski, A.: Timed I/O automata: a complete specification theory for real-time systems. In: HSCC, pp. 91–100 (2010)
42. de Alfaro, L., da Silva, L.D., Faella, M., Legay, A., Roy, P., Sorea, M.: Sociable interfaces. In: FroCos, pp. 81–105 (2005)
43. de Roever, W.-P., de Boer, F., Hannemann, U., Hooman, J., Lakhnech, Y., Poel, M., Zwiers, J.: Concurrency Verification: Introduction to Compositional and Noncompositional Methods. Cambridge University Press, Cambridge (2000)
44. Dutertre, B., de Moura, L.: A fast linear-arithmetic solver for DPLL(T). In: CAV, pp. 81–94. Springer (2006)
45. Farzan, A., Chen, Y.-F., Clarke, E. M., Tsay, Y.-K. , Wang, B.-Y.: Extending automated compositional verification to the full class of omega-regular languages. In: TACAS, pp. 2–17. Springer (2008)
46. Flanagan, C., Qadeer, S.: Thread-modular model checking. In: SPIN, pp. 213–224. Springer (2003)
47. Fleury, S., Herrb, M., Chatila, R.: GenoM: a tool for the specification and the implementation of operating modules in a distributed robot architecture. In: IROS, pp. 842–848 (1997)
48. Fritzson, P., Engelson, V.: Modelica a unified object-oriented language for system modeling and simulation. In: ECOOP, pp. 67–90 (1998)
49. Giannakopoulou, D., Păsăreanu, C. S., Barringer, H.: Assumption generation for software component verification. In: ASE, pp. 3–12. IEEE Computer Society (2002)
50. Gößler, G., Sifakis,. J.: Priority systems. In: FMCO, pp. 314–329 (2003)
51. Gupta, A., Popeea, C., Rybalchenko, A.: Predicate abstraction and refinement for verifying multi-threaded programs. In POPL, pages 331–344. ACM, 2011
52. Heimbold, D., Luckham, D.: Debugging Ada tasking programs. IEEE Softw. **2**(2), 47–57 (1985)
53. Henzinger, T.A., Hottelier, T., Kovács, L., Rybalchenko, A.: Aligators for arrays. In: LPAR, TACAS, pp. 348–356 (2010)
54. Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G.: Lazy abstraction. In: POPL, pp. 58–70. ACM (2002)
55. Henzinger, T.A., Qadeer, S., Rajamani, S. K.: You assume, we guarantee: methodology and case studies. In: CAV, pp. 440–451. Springer (1998)
56. Hermenegildo, M., Puebla, G., Marriott, K., Stuckey, P.J.: Incremental analysis of constraint logic programs. ACM Trans. Program. Lang. Syst. **22**(2), 187–223 (Mar. 2000)
57. Jones, C.B.: Specification and design of (parallel) programs. In: IFIP Congress, pp. 321–332 (1983)
58. Khendek, F., Bochmann, G.V.: Incremental construction approach for distributed system specifications. In: Proceedings of the International Symposium on Formal Description, Techniques, pp. 26–29 (1993)

59. Larsen, K.G.: Modal specifications. In: Automatic Verification Methods for Finite State Systems, pp. 232–246 (1989)
60. Lau, K.-K., Ng, K.-Y., Rana,T., Tran, C.M.: Incremental construction of component-based systems. In: CBSE, pp. 41–50. ACM, New York, NY, USA (2012)
61. Lynch, N.A., Tuttle, M.R.: An introduction to input/output automata. CWI Q. **2**, 219–246 (1989)
62. Manna, Z., Pnueli, A.: Temporal Verification of Reactive Systems: Safety. Springer, New York (1995)
63. Moura, L.D., Bjørner, N.: Z3: an efficient SMT solver. In: TACAS, pp. 337–340. Springer (2008)
64. Nguyen, T.-H.: Constructive Verification of Component-Based Systems. PhD Thesis, Institut National Polytechnique de Grenoble (2010)
65. Patil, S.S.: Limitations and Capabilities of Dijkstra's Semaphore Primitives for Coordination among Processes. Cambridge, MA: MIT, Project MAC, Computation Structures Group Memo 57, Feb (1971)
66. Pnueli, A.: In transition from global to modular temporal reasoning about programs. Log. Models Concurr. Syst. **F13**, 123–144 (1985)
67. Popeea, C., Rybalchenko, A.: Compositional termination proofs for multi-threaded programs. In: TACAS, pp. 237–251 (2012)
68. Queille, J.-P., Sifakis, J.: Specification and verification of concurrent systems in CESAR. In: ICALP, pp. 337–351. Springer (1982)
69. Somenzi, F.: CUDD: CU decision diagram package
70. Team, O.: The Omega, library (1996)
71. Thiele, L., Bacivarov, I., Haid, W., Huang, K.: Mapping Applications to Tiled Multiprocessor Embedded Systems. In: ACSD, pp. 29–40. IEEE Computer Society (2007)
72. Tripakis, S., Stergiou, C., Shaver, C., Lee, E.A.: A modular formal semantics for ptolemy. Math. Struct. Comput. Sci. **23**, 834–881 (2013)

Saddek Bensalem Professor at the University of Joseph Fourier (Grenoble). He received his Ph.D. in Computer Science from INP Grenoble (Institut National Polytechnique de Grenoble). His area of expertise is modeling and validation of real-time systems. He has an excellent background and experience in software and system verification. He has spent two years at the SRI International in Menlo Park California and one year at the University of Stanford as Visiting Scientist. He has consulted for NASA Ames. Currently, he works on component-based modeling. His current work is focused on verification of real-time applications in general and autonomous systems in particular. He is also the technical project leader for Verimag activities in European and industrial projects.

Marius Bozga Research Engineer at Centre national de la recherche scientifique (CNRS). He received his Ph.D. in Computer Science from Joseph Fourier University of Grenoble. His area of expertise covers program verification, modeling, validation and verification of real-time systems. He is the major contributor of many tools such as BIP Toolbox, IF.

Axel Legay Researcher at Institut national de recherche en informatique et en automatique (INRIA Rennes). He received his Ph.D. in Computer Science from University of Liege, Belgium. His main research interests are in formal verification. He is a founder and major contributor of statistical model checking (a statistical variant of model checking effectively used in industry). He is a referee for top journals and conferences in formal verification.

Thanh-Hung Nguyen Lecturer and Researcher at Software Engineering Department, Hanoi University of Science and Technology. He received his Ph.D. and Master degrees in Computer Science from Grenoble Institute of Technology. He has spent six years at Verimag Laboratory, where he focused on construction and verification methods for component-based systems. His research interests include theory, methods and tools for the compositional and verification of component-based systems. He also has the experience in constructing complex component-based systems.

Joseph Sifakis Director of the "Rigorous System Design" Laboratory at EPFL, director of the "Center for Integrative Research" in Grenoble, CNRS researcher and the founder of Verimag laboratory, a leading research laboratory in the area of critical embedded systems established in Grenoble, in 1993. His research interests cover fundamental and applied aspects of embedded systems design. The main focus of his work is on the formalization of system design as a process leading from given requirements to trustworthy, optimized and correct-by-construction implementations. He is a member of the editorial board of several journals, co-founder of the International Conference on Computer Aided Verification (CAV) and a member of the Steering Committee of the EMSOFT (Embedded Software) conference.

Rongjie Yan Assistant researcher at the Institute of Software, Chinese Academy of Sciences. She received her Ph.D. degree from the institute of Software, Chinese Academy of Sciences. She has spent two years at Veriamg, where she focused on compositional and incremental verification methodology, and correctness-by-construction of-component-based systems. Her current reserch interests include modeling and formal verification of embedded systems, verification of real-time systems, and extra-functional analysis of embedded systems.