Unité Mixte de Recherche 5104 CNRS - INPG - UJF
Centre Equation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

Incremental Invariant Generation for Compositional Design

Saddek Bensalem, Axel Legay, Thanh-Hung Nguyen, Joseph Sifakis, Rongjie Yan

Verimag Research Report n^o TR-2010-6

February, 2010

Reports are downloadable at the following address
http://www-verimag.imag.fr

Incremental Invariant Generation for Compositional Design

Saddek Bensalem, Axel Legay, Thanh-Hung Nguyen, Joseph Sifakis, Rongjie Yan

February, 2010

Abstract

We consider a compositional method for the verification of component-based systems described in a subset of the BIP language encompassing multi-party interactions. The method is based on the use of two kinds of invariants. Component invariants are over-approximations of components' reachability sets. Interaction invariants are constraints on the states of components involved in interactions. In this paper we propose fixed point characterization for computing interaction invariants. We also propose a new technique that takes the incremental design of the system into account. In many situations, the technique will help to avoid redoing all the verification process each time an interaction is added in the design. Our two techniques have been implemented as extension of the D-Finder toolset. The result has been applied to check deadlock-freedom on several case studies. Our experiments show that our new methodology is generally much faster than existing ones.

Keywords: Incremental, invariants, fixed point, BIP.

How to cite this report:

@techreport { TR-2010-fixed-point,
title = { Incremental Invariant Generation for Compositional Design},
authors = { Saddek Bensalem, Axel Legay, Thanh-Hung Nguyen, Joseph Sifakis, Rongjie Yan },
institution = { Verimag Research Report },
number = {TR-2010-6},
year = { 2010},
}

1 Introduction

Model checking [22, 15, 34] is by now a well-known method for proving properties of programs. The main reason for its success is that it works fully automatically, i.e. without any manual intervention of the user. Unfortunately, the method may suffer from the so called *state-space explosion* problem, i.e., the number of states of the system may be too big to be analyzed automatically.

There have been a lot of work on developing new methodologies to cope with the state-space explosion problem. Among them, one finds *partial order reduction* techniques [27, 35] or *symbolic representations* based techniques [23]. Unfortunately, the situation gets even worse when one has to apply model checking techniques to the verification of concurrent systems. The problem being that concurrency often requires to compute the product of the individual systems by using both interleaving and synchronization. In general, the size of this structure is prohibitive and cannot be handled without manual interventions.

In a series of recent works, it has been advocated that *compositional verification techniques* could be used to cope with state explosion in concurrent systems. The idea is to apply divide-and-conquer approaches to infer global properties of complex systems from properties of their components. Separate verification of components limits state explosion. Nonetheless, components mutually interact in a system and their behavior and properties are inter-related. This is a major difficulty in designing compositional techniques. As explained in [24], compositional rules are in general of the form

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

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

One approach to compositional verification is by *assume-guarantee* where properties are decomposed into two parts. One is an assumption about the global behavior of the system within which a component is interacting; the other is a property guaranteed by the component when the assumption about its environment holds. This approach has been extensively studied (see for example [2, 1, 14, 11, 20, 25, 29, 32]). Many issues make the application of assume-guarantee rules difficult. These are discussed in detail in a recent paper [16] which provides an evaluation of automated assume-guarantee techniques. The main difficulties are finding decompositions into sub-systems and choosing adequate assumptions for a particular decomposition.

In a recent paper [5], Bensalem et al. proposed a new approach for compositional reasoning. This approach is based on the following rule:

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

The rule allows to prove invariance of Φ for systems obtained by using a n-ary composition operation parameterized by a set of interactions γ. It uses global invariants which are the conjunction of individual invariants of components Φ_i and an interaction invariant Ψ. The latter expresses constraints on the global state space induced by interactions between components. It can be computed automatically from abstractions of the system to be verified. These are the composition of finite state abstractions B_i^α of the components B_i with respect to their invariants Φ_i. The approach has been implemented in the D-Finder toolset [6] and applied to check deadlock-freedom on several case studies. The results of these experiments show that D-Finder is exponentially faster than well-established tools such as NuSMV [13].

The methods in [5], which can be implemented in a symbolic manner, requires to handle complex formulas that represent both the behaviors of components and the interactions between components. Moreover, it does not allow to reuse existing verification results when one adds new interactions in the design. This is problematic as components are generally developed by independent teams, which call for successive verifications until the whole design (all the interactions) has been built.

In this paper, we go one step further and propose a fixed point characterization for interaction invariants. The technique starts by computing successors for each component location and, at each iteration, constrains the result by using existing interactions and the partially computed information for other locations. The resulting formula represents exactly the same interaction invariant than the one produced by the method in

[5]. However, it is generally smaller and easier to handle than the one obtained with the method in [5] that directly mixes all the invariants with all the interactions.

Incremental system design methodologies often work by adding new interactions to existing sets of components. Each time an interaction is added, one may be interested to verify whether the resulting system satisfies some given property. Indeed, it is important to report an error as soon as it appears. However, each verification step may be time consuming, which means that intermediary verification steps are generally avoided. The situation could be improved if the result of the verification process could be reused when new interactions are added. Unfortunately, existing techniques do not focus on such aspects. In this paper, we propose a solution to this problem, which takes the form of a new technique that takes the incremental design of the system into account. In many situations, the technique will help to avoid redoing all the verification process each time an interaction is added in the design. Our solution is developed for the fixed point formula, but the principle could be generalized to other techniques.

Our two techniques have been implemented as extensions of the D-Finder toolset. The result has been applied to check deadlock-freedom on several case studies. Our experiments show that our new methodology is generally much faster than the one proposed in [5]. In particular, we have been capable to verify deadlock-freedom of Utopar, an automated transportation system developed by one of our industrial partners within the COMBEST European project. This case study is beyond the scope of existing verification techniques for checking deadlock-freedom. We have also identified case studies for which our technique is less efficient than the one in [5].

The methods presented in this paper and in [5] are fairly simple, but they allow to verify deadlock-freedom for concurrent systems that are beyond the scope of most existing techniques. The Utopar case study has long been a challenge for algorithmic approaches developed by our partners in COMBEST and was handled without any specific tailoring of our fairly simple approach. What is really exciting about this direction of work is that so much could be achieved with so little.

Structure of the paper. In Section 2, we recap the concepts that will be used through the rest of the paper. Section 3 presents a fixed point extension of the technique presented in [5], while Section 4 presents the incremental version. Experimental results are discussed in Section 5. Finally, Section 6 concludes the paper.

2 Preliminaries

In this section, we recap concepts that will be used through the rest of the paper. We start with the concepts of *components* and *parallel composition of components*. Then we discuss *systems* and *invariants*.

2.1 Components and Parallel Composition

In the paper, we will be working with a simplified model for component-based design, which is used in the *Behavior-Interaction-Priority* (BIP) component framework developed at Verimag [19, 30]. This framework has been implemented in a language and a toolset called BIP [4]. The BIP language offers primitives and constructs for modelling and composing components starting from atomic components.

Roughly speaking, an atomic component is nothing more than a transition system whose transitions' labels are called *ports*. These ports are used to synchronize with other components. Formally, we have the following definition.

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

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

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

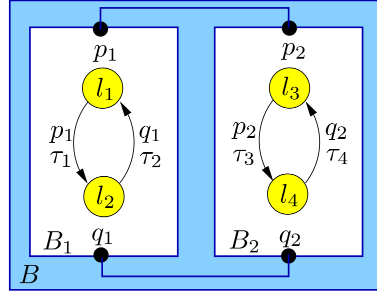


Figure 1: A simple example

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

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

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

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

In the rest of the paper, we simplify the notations and write $p_1 p_2 \ \ldots \ p_k$ instead of $\{p_1, \ldots, p_k\}$. We also write $a_1 + \ldots + a_m$ for the connector $\{a_1, \ldots, a_m\}$. As an example, notation for the connector $\{\{p_1, p_2\}, \{q_1, q_2\}\}$ is $p_1\ p_2 + q_1\ q_2$.

We now propose our definition for parallel composition.

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

- *$\mathcal{L} = L_1 \times L_2 \times \ldots \times L_n$ is the set of* global *locations,*
- *γ is a set of interactions, and*
- *$\mathcal{T} \subseteq \mathcal{L} \times \gamma \times \mathcal{L}$ contains all transitions $\tau = ((l_1, \ldots, l_n), a, (l'_1, \ldots, l'_n))$ obtained by synchronization of sets of transitions $\{\tau_i = (l_i, p_i, l'_i) \in \mathcal{T}_i\}_{i \in I}$ such that $\{p_i\}_{i \in I} = a \in \gamma$ and $l'_j = l_j$ if $j \notin I$.*

The idea is that components communicate by synchronization with respect to interactions. Given an interaction a, only those components that are involved in a can make a step. This is done by following a transition labelled by the corresponding port involved in a. If a component does not participate to the interaction, then it has to remain in the same state. In the rest of the paper, a component that is obtained by composing several components will be called a *composite component*. Observe also that the parallel composition $\gamma(B_1, \ldots, B_n)$ of $B_1, \ldots, B_n$ can be seen as a *1-safe Petri net* whose set of places is given by $L = \bigcup_{i=1}^n L_i$ and whose transitions relation is given by $\mathcal{T}$. In the rest of the paper, L will be called the *set of locations of* B, while $\mathcal{L}$ is the set of *global locations*.

2.2 Systems and Invariants

We now define the concept of invariants, which can be used to verify properties of (parallel composition of) components. We first propose the definition of *system* that is a component with an initial set of states.

Definition 4 (System)**.** *A system $\mathcal{S}$ is a pair $\langle B, Init\rangle$ where B is a component and $Init$ is a state predicate characterizing the initial states of B.*

Consider the example given in Figure 1. If we assume that $Init = l_1 \wedge l_3$, then we obtain a composite component $\gamma(B_1, B_2)$, where $\gamma = p_1\ p_2 + q_1\ q_2$.
In a similar way, we distinguish invariants of a component from those of a system. Therefore we define invariants for a component and for a system separately.

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

Clearly, if $I_1,\ I_2$ are invariants of B (respectively $\mathcal{S}$) then $I_1 \wedge I_2$ and $I_1 \vee I_2$ are also invariants of B (respectively $\mathcal{S}$).

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

3 Interaction Invariant Computation

A compositional verification method for safety properties of component-based systems is proposed in [5]. The method is based on the following rule:

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

This rule allows to prove invariance of a predicate Φ for a system $\gamma(B_1, \ldots, B_n)$ obtained by using a n-ary parallel composition operation parameterized by a set of interactions γ on a set of components $\{B_i\}_{i=1}^n$. $B_i\langle\Phi_i\rangle$ means that Φ_i is the component invariant for B_i. Ψ belongs to the set II of interaction invariants II of $\gamma(B_1, \ldots, B_n)$ computed automatically from Φ_i and $\gamma(B_1, \ldots, B_n)$. The invariance of Φ is verified by checking tautology $(\bigwedge_i^n \Phi_i) \wedge \Psi \Rightarrow \Phi$.

In the same paper, two methods are provided for computing component invariants and interaction invariants:

1. Component invariants are over-approximations of the set of the reachable states of atomic components and are generated by simple forward propagation techniques. The iteration of the forward propagation allows to compute sequences of increasingly stronger component invariants. A key issue is efficient computation of such invariants as the precise symbolic computation of reachable states requires quantifier elimination. An alternative to quantifier elimination is to compute over-approximations based on syntactic analysis of the predicates occuring in guards and actions. In this case, the obtained invariants may not be inductive. Different strategies are used to derive local assertions, that is, predicates attached to control locations and which are satisfied whenever the computation reaches the corresponding control location. A more detailed presentation, as well as the techniques for generating component invariants are given in [7, 5, 6].

2. Interaction invariants express global synchronization constraints between atomic components. The method, which is called Implication-based (IMP), computes interaction invariants by solving logical equations that come from implications for the conditions of interactions. Their computation consists of the following steps. 1) For given component invariants Φ_i of the atomic components B_i, we compute finite-state abstractions $B_i^{\alpha_i}$ of B_i where α_i is the abstraction induced by the elementary predicates occuring in Φ_i. This step is necessary only for components B_i which are infinite-state. 2) The system $\gamma(B_1^{\alpha_1}, \cdots, B_n^{\alpha_n})$ which is an abstraction of $\gamma(B_1, \cdots, B_n)$, can be considered as a 1-safe Petri net. The set of the traps of the Petri net defines a global invariant which we can compute

in a symbolic manner. 3) The concretization of this invariant gives an interaction invariant of the initial system.

In this section, we propose a new symbolic technique to compute interaction invariants for a composite component. We will first propose a Boolean equation system that links internal behaviors of each component that participate to the composition with the global behavior induced by the connector. Then, we will use this system of equations to derive a fixed point characterization of interaction invariants. Finally, we will also propose a heuristic to improve computation performances.

3.1 Boolean Behavioral Constraints

We start with the following definition (taken from [5]) that for a given location l computes the sets of component transitions involved in some interaction of γ in which transition τ_i issued from l can participate.

Definition 6 (Forward Interaction Set)**.** *Given a set of interactions γ over $B = (B_1, \ldots, B_n)$, where $B_i = (L_i, P_i, \mathcal{T}_i)$ are transition systems, we define for a location $l \in L_i$ its forward interaction set from γ by $l^\bullet_\gamma = \big\{\{\tau_i\}_{i\in I} \mid \forall i.\tau_i \in \mathcal{T}_i \ \wedge\ \exists i.{}^\bullet\tau_i = l \ \wedge\ \{port(\tau_i)\}_{i\in I} \in \gamma\big\}$.*

When it is clear from the context, we write $l^\bullet$ instead of $l^\bullet_\gamma$. As an example, consider the components given in Figure 1. Given $\gamma = p_1\ p_2 + q_1\ q_2$, we have $l_1^\bullet = \{\{\tau_1, \tau_3\}\}$, $l_2^\bullet = \{\{\tau_2, \tau_4\}\}$, $l_3^\bullet = \{\{\tau_1, \tau_3\}\}$ and $l_4^\bullet = \{\{\tau_2, \tau_4\}\}$.

In the rest of the paper, locations of components will be viewed as Boolean variables. We can thus use $Bool[L]$ to denote the free Boolean algebra generated by the set of locations L. We also extend the notation ${}^\bullet\tau$, $\tau^\bullet$ to interactions, that is ${}^\bullet a = \{{}^\bullet\tau \mid port(\tau) \in a\}$, and $a^\bullet = \{\tau^\bullet \mid port(\tau) \in a\}$. For connectors, we have γ, ${}^\bullet\gamma = \bigcup_{a\in\gamma} {}^\bullet a$.

Definition 7 (Boolean Behavioral Constraints (BBCs))**.** *Let γ be a connector over a tuple of components $B = (B_1, \cdots, B_n)$ with $B_i = (L_i, P_i, \mathcal{T}_i)$ and $L = \bigcup_{i=1}^n L_i$. The boolean behavioral constraints for every $l \in L$ in $\gamma(B)$, can be defined by $l \Rightarrow \sigma_l$, where σ_l is a function $l \rightarrow Bool[L]$ such that:*

$$\sigma_l = \begin{cases} \bigwedge_{\{\tau_i\}_{i\in I}\in l^\bullet} \bigvee_{l'\in\{\tau_i^\bullet\}_{i\in I}} l' & l^\bullet \neq \emptyset \\ true & l^\bullet = \emptyset \end{cases} \tag{1}$$

Roughly speaking, σ_l describes the constraints added by the synchronization over l. For the example given in Figure 1, the corresponding constraints of all the locations are as follows:

$$l_1 \Rightarrow l_2 \vee l_4, \quad l_3 \Rightarrow l_2 \vee l_4$$
$$l_2 \Rightarrow l_1 \vee l_3, \quad l_4 \Rightarrow l_1 \vee l_3$$

The logical formula $l_1 \Rightarrow l_2 \vee l_4$ ensures that only locations l_2 or l_4 are reachable by following the interaction $p_1\ p_2$.

Notice that boolean behavioral constraint $l \Rightarrow \sigma_l$ can be written into equation $l = l \wedge \sigma_l$, which is called BBC-equation. Equation $l = l \wedge \sigma_l$ can be viewed as an image function from a local location to l and a set of reachable locations contained in σ_l. Those "image" locations can be reached by following the interactions given in γ. To simplify the notations, we often use σ_i instead of σ_{l_i}.

3.2 Location-based Fixed Point Computation for Interaction Invariants

We now compute interaction invariants by using BBC-equations. The intuition is as follows. If l_j is in σ_i, and $l_j \wedge \sigma_j$ is the sets of set of reachable locations through the interactions from l_j, then we can apply $l_j \wedge \sigma_j$ to $l_i = l_i \wedge \sigma_i$, and obtain an equation that represents the locations that can be reached from l_i via l_j. If we repeat the same operation to all the locations until no more reachable location are added, then the right side of the obtained equation for a given location represents the set of sets of reachable locations 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 now formalize the above process. Since we want to derive a fixed point characterization, we first have to define a preorder on sets of sets of locations. We propose the following definition.

Definition 8. *Given a set of locations L, we define a preorder $\sqsubseteq\subseteq 2^{2^L} \times 2^{2^L}$. Given two sets of S_1, S_2 over 2^{2^L}, $S_1 \sqsubseteq S_2$ if for any $m_2 \in S_2$, there exists $m_1 \in S_1$ such that $m_1 \subseteq m_2$.*

Then we can apply function $Image : 2^{2^L} \times 2^{2^L} \rightarrow 2^{2^L}$ over a set $F = \{f(l)\}_{l \in L}$ to compute the set of sets of locations iteratively, where

$$Image(S_1, F) = \bigvee_{m \in S_1} \bigwedge_{l \in m} f(l)$$

We are now ready to define our set of fixed point equations. We will work with vectors of solutions. We consider a vector $\mathbb{L}$ indexed by the locations in L. We use $\mathbb{L}^0$ to denote the initialization of the vector. Roughly speaking, we update the reachable locations from a location by replacing every location l in σ by its intermediate solutions. Within every iteration, we use $\mathbb{L}^k(l_i)$ to replace every location l_i in σ_l to obtain the reachable locations from every $l \in L$ through l_i. Then $\mathbb{L}^k$ becomes a vector of reachable locations, in which every item is the set of sets of reachable locations from its index location. When the iteration stops, $\mathbb{L}^k$ is a vector containing all the reachable locations from every location. The reachable locations are calculated by considering all the interactions from a location. Since the method is based on the least fixed point computation of locations, we call it Location-based Fixed Point (LFP).

Definition 9 (Location-based Fixed Point Equation)**.** *Given a composite component $\gamma(B)$ with a set of locations L, we define a location-based iteration process over a vector $\mathbb{L}$ of $l \in L$, which starts from $\mathbb{L}^0(l) = l$ and such that*

$$\mathbb{L}^{k+1}(l) = \mathbb{L}^k(l) \wedge Image(\sigma_l, \mathbb{L}^k) \tag{2}$$

The iteration terminates when $\mathbb{L}^{k+1} = \mathbb{L}^k$. In such case, $\mathbb{L}^k$ corresponds to the set of solutions for the BBC-equations of $\gamma(B)$.

Proposition 1. *Consider a composite component $\gamma(B)$ with a set of locations L. Let $\mathbb{L}$ be a vector over L, and σ_l be the BBC for $l \in L$ in $\gamma(B)$, we have $\mathbb{L}^{k+1}(l) = l \wedge Image(\sigma_l, \mathbb{L}^k)$.*

Proof 1. *Since $\mathbb{L}^k(l) \sqsubseteq \mathbb{L}^k(l) \wedge Image(\sigma_l, \mathbb{L}^k)$, we have $\mathbb{L}^k(l) \sqsubseteq \mathbb{L}^{k+1}(l)$. The proof is by induction on k. First, we have $\mathbb{L}^0(l) = l$. Suppose that $\mathbb{L}^k(l) = l \wedge Image(\sigma_l, \mathbb{L}^{k-1})$. Then $\mathbb{L}^{k+1}(l) = \mathbb{L}^k(l) \wedge Image(\sigma_l, \mathbb{L}^k) = l \wedge Image(\sigma_l, \mathbb{L}^{k-1}) \wedge Image(\sigma_l, \mathbb{L}^k)$. Since $\mathbb{L}^{k-1} \sqsubseteq \mathbb{L}^k$, we have $Image(\sigma_l, \mathbb{L}^{k-1}) \wedge Image(\sigma_l, \mathbb{L}^k) = Image(\sigma_l, \mathbb{L}^k)$. Therefore $\mathbb{L}^{k+1}(l) = l \wedge Image(\sigma_l, \mathbb{L}^k)$.*

This proposition shows that the iteration for $\mathbb{L}$ can be focused on $Image(\sigma, \mathbb{L})$ instead of $\mathbb{L} \wedge Image(\sigma, \mathbb{L})$, which can improve the efficiency of the computation.

Proposition 2. *Let γ be a connector over B with a set of locations L. If $\mathbb{L}^k$ is the set of solutions for BBC-equations of $\gamma(B)$, then for each $l \in L$ such that $\mathbb{L}^k(l) = \bigvee_{i \in I} m_i$, 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 2. *First, observe that m_i is a set of reachable locations from l_i. The reason is that m_i contains at least one monomial in σ_i, which enumerates one reachable location for every interaction from l_i. If σ_i contains some l_j, then m_i also contains those locations in σ_j. So m_i is a set of reachable locations from l_i.*

Suppose now that m_j and m_k are solutions of $\mathbb{L}^k(l)$ for $l \in L$ by Definition 9 such that $m_k \subset m_j$, meaning that m_j goes through more locations than m_k. Because $m_k \vee m_j = m_k$. Therefore, m_j does not exist when the iteration stops, and m_k is a minimal set of reachable locations.

We now introduce a new notation $\widetilde{m}$ for the dual of m, which is used to compute invariants from solutions. The following theorem shows that the fixed point solution for the BBC-equations characterizes an interaction invariant.

Theorem 1. *Let γ be a connector over a set of components $B = (B_1, \ldots, B_n)$, with $B_i = (L_i, P_i, \mathcal{T}_i)$ and $L = \bigcup_{i=1}^n L_i$. If $\mathbb{L}^k$ is the set of solutions for BBC-equations of $\gamma(B)$, then for each $l \in L$ such that $\mathbb{L}^k(l) = \bigvee_{i \in I} m_i$, $\widetilde{\mathbb{L}}^k(l) = \bigwedge_{i \in I} \widetilde{m}_i$ is an invariant and $\bigwedge_{l \in L} \widetilde{\mathbb{L}}^k(l)$ is an invariant of $\gamma(B)$.*

Proof 3. *Consider an m in $\mathbb{L}^k(l)$, according to Proposition 2, m gives a minimal set of reachable locations through interactions of γ via l. Assume that for some global state $l = (l_1, \cdots, l_n)$, we have l_i belongs to m, then $\widetilde{m}$ is true. If from l_i there exists an interaction $a \in \gamma$ such that $l_i \in {}^\bullet a$, then there exists $l'_j \in a^\bullet$ with (l_j, p_j, l'_j) and $p_j \in a$, such that l'_j belongs to m and $\widetilde{m}$ is still true. So any successor state of l by an interaction a satisfies $\widetilde{m}$ and $\widetilde{m}$ is an invariant of $\gamma(B)$.*

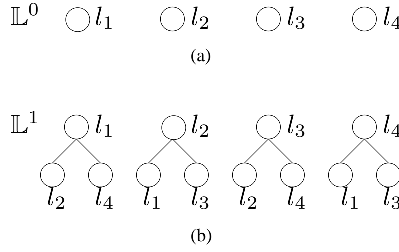


Figure 2: Iteration step

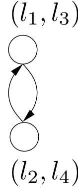


Figure 3: Traditional fixed point computation

Since the conjunction of invariants is still an invariant, $\widetilde{\mathbb{L}}^k(l) = \bigwedge_{i \in I} \widetilde{m}_i$ is an invariant of $\gamma(B)$, and $\bigwedge_{l \in L} \widetilde{\mathbb{L}}^k(l)$ 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 the dual of every solution. In Example 2, we have computed the set of solutions for every location. The union of solutions is $l_1 \wedge l_2 \vee l_1 \wedge l_4 \vee l_2 \wedge l_3 \vee l_3 \wedge l_4$. So the interaction invariant is given by $(l_1 \vee l_2) \wedge (l_1 \vee l_4) \wedge (l_2 \vee l_3) \wedge (l_3 \vee l_4)$.

We conclude the subsection with an example that illustrates our approach.

Example 2 (LFP computation)**.** *Consider the components in Figure 1. Consider also the following connector $\gamma = p_1\ p_2 + q_1\ q_2$. The BBC-equations for the locations of each components are:*

$$l_1 = l_1 \wedge (l_2 \vee l_4), \quad l_2 = l_2 \wedge (l_1 \vee l_3)$$
$$l_3 = l_3 \wedge (l_2 \vee l_4), \quad l_4 = l_4 \wedge (l_1 \vee l_3)$$

The successive solutions for each iteration of the fixed point are represented in Figure 2. In this figure, the outgoing arcs from a node correspond to the literals of the disjunction. Conjunction of all the literals corresponding to the arcs at the path from the root of the tree to a node is associated with the node. As example, in Figure 2 (b), the first tree can be written as $l_1 \wedge l_2 \vee l_1 \wedge l_4$.

The initialization of $\mathbb{L}^0$ is shown in Figure 2 (a). After the first iteration, we obtain the solutions shown in Figure 2 (b). In the second iteration, we apply $\mathbb{L}^1$ to replace the corresponding location in σ_i as follows:

$$\mathbb{L}^2(l_1) = l_1 \wedge (l_2 \wedge (l_1 \vee l_3) \vee l_4 \wedge (l_1 \vee l_3)) = l_1 \wedge l_2 \vee l_1 \wedge l_4$$
$$\mathbb{L}^2(l_2) = l_2 \wedge (l_1 \wedge (l_2 \vee l_4) \vee l_3 \wedge (l_2 \vee l_4)) = l_1 \wedge l_2 \vee l_2 \wedge l_3$$
$$\mathbb{L}^2(l_3) = l_3 \wedge (l_2 \wedge (l_1 \vee l_3) \vee l_4 \wedge (l_1 \vee l_3)) = l_2 \wedge l_3 \vee l_3 \wedge l_4$$
$$\mathbb{L}^2(l_4) = l_4 \wedge (l_1 \wedge (l_2 \vee l_4) \vee l_3 \wedge (l_2 \vee l_4)) = l_3 \wedge l_4 \vee l_1 \wedge l_4$$

Since $\mathbb{L}^2 = \mathbb{L}^1$, the iteration stops, and the solutions are shown in Figure 2 (b).

Observe that the result of the above fixed point computation differs from the set of reachable states of the composite component. Let l_1 and l_3 respectively be the initial locations in B_1 and B_2, Figure 3 shows another invariant that is the set of reachable global states of the composition. This set is more accurate than our computation, but this precision has a cost when working with complex systems. It takes interaction as a global transition and computes exactly the destination locations from each source location in the global state that participates the synchronization. However, in our method, what we need is only one location that can be reached by an interaction.

3.3 Implementation on LFP Computation

The method presented in the above subsection requires to compute the fixed point for each location of each component that participates to the composition. However different locations may have the same solutions. Being able to identify such locations would improve the efficiency of the computation.

In fact, we observe that two locations that are reachable from each other by an interaction do have the same solution. Unfortunately, by using the computation above, the solution is computed twice. A way to this problem is instead to compute the solutions for every location separately, we can put the locations together and compute fixed point induced by BBC-equations defined in the following definition.

Definition 10. *Given a composite component $\gamma(B)$ with a set of locations L, assume also Δ, a vector of BBC-equations for all $l \in L$ with $\Delta(l) = l \wedge \sigma_l$. We define an iteration process by*

$$\begin{array}{l} \mathbb{L}^0 = \bigvee_{l \in L} l \\ \mathbb{L}^{k+1} = Image(\mathbb{L}^k, \Delta) \end{array} \tag{3}$$

When $\mathbb{L}^{k+1} = \mathbb{L}^k$, the iteration stops and we obtain the set of solutions for the BBC-equations of $\gamma(B)$.

Obviously, we have $\mathbb{L}^k \sqsubseteq \mathbb{L}^{k+1}$. Therefore, though we obtain different solutions from different locations, we can always obtain the minimal solutions by using Definition 10.

Proposition 3. *The set of solutions obtained from Definition 9 and 10 are always the same.*

Proof 4. *Both definitions give the solution computation from the same set of locations, based on the same set of BBC-equations, which makes the same set of solutions.*

Example 3. *Consider again the example given in Figure 1, with $\gamma = p_1\ p_2 + q_1\ q_2$. After the first iteration, we have $\mathbb{L}^0 = l_1 \vee l_2 \vee l_3 \vee l_4$, and Δ is given by*

$$\begin{array}{ll} \Delta(l_1) = l_1 \wedge (l_2 \vee l_4), & \Delta(l_2) = l_2 \wedge (l_1 \vee l_3) \\ \Delta(l_3) = l_3 \wedge (l_2 \vee l_4), & \Delta(l_4) = l_4 \wedge (l_1 \vee l_3) \end{array}$$

By Definition 10, $\mathbb{L}^1 = Image(\mathbb{L}^0, \Delta) = l_1 \wedge l_2 \vee l_1 \wedge l_4 \vee l_2 \wedge l_3 \vee l_3 \wedge l_4$. Since $\mathbb{L}^2 = \mathbb{L}^1$, the iteration stops and, according to our definition, $\mathbb{L}^1$ is the set of solutions for the BBC-equations of $\gamma(B)$.

4 Incremental Component-based System Construction and Computation of Interaction Invariants

In component-based design, the construction of a composite component is both step-wise and hierarchical. This means that a system is obtained from a set of atomic components by successive additions of new interactions, also called *increments*. In this section, we propose a methodology that allows to add new interactions to a composite component without breaking existing synchronization. We also show how to reuse intermediary results to increase efficiency.

4.1 Incremental Construction of Component-based Systems

In component-based design, new increments are progressively added during the step-wise construction process. It is important that these new increments do not break the synchronizations that are assumed by interactions before the addition takes place. Our first step is thus to define a notion of forbidden interactions. Formally, we propose the following definition.

Definition 11 (Closure and Forbidden Interactions)**.** *Let γ be a connector.*

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

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

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

Definition 12 (Increments)**.** *Consider a connector γ over a composite component B and let $\delta \subseteq 2^\gamma$ be a set of interactions. We say δ is an increment over γ if for any interaction $a \in \delta$ we have interactions $b_1, \ldots, b_n \in \gamma$ such that $\bigcup_{i=1}^n b_i = a$.*

In the above definition, fusion of existing interactions into new interactions can produce a new increment.

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

Definition 13 (Layering)**.** *Given a connector γ and an increment δ over γ, the new set of interactions obtained by combining δ and γ, also called layering, is given by the following set $\delta\gamma = (\gamma - \delta^f) + \delta$ the incremental construction by layering, that is, the incremental modification of γ by δ.*

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

Definition 14 (Superposition)**.** *Given two increments δ_1, δ_2 over a connector γ, the operation of superposition between δ_1 and δ_2 is defined by $\delta_1 + \delta_2$.*

Superposition can be seen as a composition between increments, with looser coupled relation. If we combine the superposition of increments with the layering proposed in Definition 13, then we obtain an incremental construction from a set of increments. Formally, we have the following proposition.

Proposition 4. *Let γ be a connector over B, the incremental construction by the superposition of n increments $\{\delta_i\}_{1 \le i \le n}$ is given by*

$$(\sum_{i=1}^n \delta_i)\gamma = (\gamma - (\sum_{i=1}^n \delta_i)^f) + \sum_{i=1}^n \delta_i \quad (4)$$

The above proposition provides a way to transform incremental construction by a set of increments into the separate constituents, where $\gamma - (\Sigma_{i=1}^n \delta_i)^f$ is the set of interactions that are not tightened during the incremental construction process.

We conclude the subsection with the following example.

Example 4 (Incremental construction)**.** *Consider again the components given in Figure 1 and let $\gamma = p_1 + p_2 + q_1 + q_2$ be a connector. Let $\delta_1 = p_1\ p_2$ and $\delta_2 = q_1\ q_2$ be two increments over γ. Since $\gamma - (\delta_1 + \delta_2)^f = \emptyset$, we have $(\delta_1 + \delta_2)\gamma = p_1\ p_2 + q_1\ q_2$.*

4.2 Incremental Computation of Interaction Invariants

We now study the modification in the BBC-equations that are needed to take increments added to the original design into consideration. Our first objective consists in extending the BBC-equations of each location. We will assume that the BBC-equations for each increment have already been computed and we will show how to obtain the BBC-equations for the original connector plus the increments. Given a series of increments $\{\delta_i\}_{1 \le i \le n}$ and a location l, σ_{il} denotes the BBC for the location l with respect to Increment δ_i. We use σ_{0l} to denote the BBC with respect to $\gamma - (\Sigma_{i=1}^n \delta_i)^f$.

Proposition 5. *Consider a composite component B with a set of locations L. Let γ be a connector over B and let $\{\delta_i\}_{1 \le i \le n}$ be a set of increments over γ. Assume that $\delta_0 = \gamma - (\Sigma_{i=1}^n \delta_i)^f$, and let $l = l \wedge \sigma_{il}$ be the BBC-equation of l in $\delta_i(B)$ for $0 \le i \le n$. The BBC-equations for $(\Sigma_{i=1}^n \delta_i)\gamma(B)$ are of the following form:*

$$l = l \wedge \bigwedge_{i=0}^n \sigma_{il} \quad (5)$$

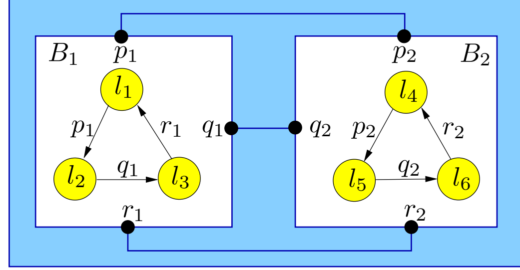


Figure 4: Example for incremental invariant computation

Proof 5. *For any $l \in L$, let $l^\bullet_{\delta_i}$ be the forward interaction set of l under δ_i, we have $l^\bullet_{(\delta_i+\delta_j)} = l^\bullet_{\delta_i} \cup l^\bullet_{\delta_j}$ for any δ_i and δ_j. And its BBC is $\sigma_{il} \wedge \sigma_{jl}$. Therefore the BBC-equations of $(\Sigma_{i=1}^n \delta_i)\gamma(B)$ are of the form $l = l \wedge \bigwedge_{i=0}^n \sigma_{il}$.*

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. By inspecting Equation 5, one can observe that if a location is involved in more than one increment, then the number of forward interaction sets for that location will increase, which is caused by sharing a same port or more than one outgoing transition.

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 Fixed Point ($ILFP$) 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 Equation 4, 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 from the BBC-equations for $(\gamma - (\Sigma_{i=1}^n \delta_i)^f)(B)$. We suggest the following proposition.

Proposition 6 (Incremental LFP Computation)**.** *Consider a composite component B and a set of locations L. Let γ be a connector for B and assume a set of increments $\{\delta_i\}_{1\leq i\leq n}$ over $\gamma(B)$. Let $\delta_0 = \gamma - (\Sigma_{i=1}^n \delta_i)^f$ and $\mathbb{L}_i$ be the solutions of BBC-equations of $\delta_i(B)$ with respect to $\bigvee_{l\in {}^\bullet\delta_i} l$, where $0 \leq i \leq n$. The solutions for the BBC-equations of $(\Sigma_{i=1}^n \delta_i)\gamma(B)$ can be computed with the following equation.*

$$\begin{array}{l} \mathbb{L}^0 = \bigvee_{i=0}^n \mathbb{L}_i \vee \bigvee_{l\in L - \bigcup_{i=0}^n {}^\bullet\delta_i} l \\ \mathbb{L}^{k+1} = Image(\mathbb{L}^k, \Delta) \end{array} \quad (6)$$

where Δ is the vector of BBC-equations for $(\Sigma_{i=1}^n \delta_i)\gamma(B)$.

Proof 6. *The idea behind the proof is to show that we can get the same set of solutions by using Equation 6 or Equation 3. As $Image$ distributes over disjunction, we have $Image(\mathbb{L}^0, \Delta) = \bigvee_{i=0}^n Image(\mathbb{L}_i, \Delta) \vee Image(\bigvee_{l\in L - \bigcup_{i=0}^n {}^\bullet\delta_i} l, \Delta)$. Then, since $\bigvee_{l\in L} l \sqsubseteq \mathbb{L}_i$, and $\Delta(l)$ is monotonic, we have that $\bigvee_{l\in L} l$ and $\bigvee_{i=0}^n \mathbb{L}_i \vee \bigvee_{l\in L - \bigcup_{i=0}^n {}^\bullet\delta_i} l$ have the same least fixed points over Δ.*

Proposition 6 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 solutions.

We conclude the section with an example.

Example 5 (Incremental LFP computation)**.** *Consider the components given in Figure 4 and let $\gamma = p_1 + p_2 + q_1 + q_2 + r_1 + r_2$. Consider also two increments $\delta_1 = p_1\ p_2 + r_1\ r_2$ and $\delta_2 = q_1\ q_2$ that are defined over γ. Since $\gamma - (\delta_1 + \delta_2)^f = \emptyset$, we have $\delta_0 = \emptyset$. The vector of BBC-equations Δ_1 for $\delta_1(B)$ can*

thus be defined as follows:

$$\begin{array}{ll}\Delta_1(l_1) = l_1 \wedge l_2 \vee l_1 \wedge l_5, & \Delta_1(l_4) = l_1 \wedge l_4 \vee l_4 \wedge l_5 \\ \Delta_1(l_2) = l_2, & \Delta_1(l_5) = l_5 \\ \Delta_1(l_3) = l_1 \wedge l_3 \vee l_3 \wedge l_4, & \Delta_1(l_6) = l_1 \wedge l_6 \vee l_4 \wedge l_6\end{array}$$

The vector of BBC-equations Δ_2 for $\delta_2(B)$ is as follows.

$$\begin{array}{ll}\Delta_2(l_1) = l_1, & \Delta_2(l_4) = l_4 \\ \Delta_2(l_2) = l_2 \wedge l_3 \vee l_2 \wedge l_6, & \Delta_2(l_5) = l_5 \wedge l_6 \vee l_3 \wedge l_5 \\ \Delta_2(l_3) = l_3, & \Delta_2(l_6) = l_6\end{array}$$

The set of solutions for BBC-equations are

$$\begin{array}{ll}\delta_1(B): & \mathbb{L}_1 = l_1 \wedge l_2 \vee l_1 \wedge l_5 \vee l_4 \wedge l_5 \vee l_2 \wedge l_4 \\ \delta_2(B): & \mathbb{L}_2 = l_2 \wedge l_3 \vee l_2 \wedge l_6 \vee l_5 \wedge l_6 \vee l_3 \wedge l_5\end{array}$$

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

$$\mathbb{L}^0 = \mathbb{L}_1 \vee \mathbb{L}_2 = l_1 \wedge l_2 \vee l_1 \wedge l_5 \vee l_4 \wedge l_5 \vee l_2 \wedge l_4 \vee l_2 \wedge l_3 \vee l_2 \wedge l_6 \vee l_5 \wedge l_6 \vee l_3 \wedge l_5$$

Applying Δ for $(\delta_1 + \delta_2)\gamma(B)$ into $\mathbb{L}^0$ by $Image$ function, we obtain that

$$\begin{array}{ll}\mathbb{L}^1 = & Image(\mathbb{L}^0, \Delta) = l_1 \wedge l_2 \wedge l_3 \vee l_1 \wedge l_2 \wedge l_6 \vee l_1 \wedge l_5 \wedge l_6 \vee l_1 \wedge l_3 \wedge l_5 \vee l_2 \wedge l_3 \wedge l_4 \\ & \vee l_2 \wedge l_4 \wedge l_6 \vee l_4 \wedge l_5 \wedge l_6 \vee l_3 \wedge l_4 \wedge l_5\end{array}$$

Then $\mathbb{L}^2 = \mathbb{L}^1$, so $\mathbb{L}^1$ are the solutions for the BBC-equations of $(\delta_1 + \delta_2)\gamma(B)$.

5 Experiments

We have implemented both the LFP method (Section 3) and its incremental $ILFP$ version (Section 4) in the *D-Finder toolset* [6].

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

5.1 D-Finder Structure

The D-Finder tool implements a compositional methodology for the verification of component-based systems described in the BIP language [8]. The tool provides three symbolic-representations-based methods for computing interaction invariants, namely the $ILFP$ and the LFP methods presented in this paper as well as the IMP method presented in [5] and discussed in the beginning of Section 3. D-Finder relies on the CUDD package [31] and represents sets of locations by BDDs. D-Finder also proposes techniques to compute component invariants. Those techniques, which are described in [5], relies on the Yices [18] and Omega [33] toolsets for the cases in where a component can manipulate data. A general overview of the structure of the tool is given in Figure 5.

D-Finder is mainly used to check safety properties of composite components. In this paper, we will be concerned with the verification of deadlock properites. We let DIS be the set of global states in where a deadlock can occur. The tool will progressively find and eliminate potential deadlocks as follows. D-Finder starts with an input a BIP model and computes component invariants CI by using the technique[1] outlined in [5]. From the generated component invariants, it computes an abstraction of the BIP model and the corresponding interaction invariants II. Then, it checks satisfiability of the conjunction $II \wedge CI \wedge DIS$. If the conjunction is unsatisfiable, then there is no deadlock else either it generates stronger component and interaction invariants or it tries to confirm the detected deadlocks by using reachability analysis techniques[2].

5.2 Experimental Results

We have compared the three methods ($ILFP$, LFP and IMP) on several case studies. All our experiments have been conducted with a 2.4GHz Duo CPU Mac laptop with 2GB of RAM.

[1] For those systems that do manipulate data, this requires quantifier eliminations, which are performed with the Omega toolset [33]. Deadlock-freedom of components is checked by using the tool Yices [18].

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

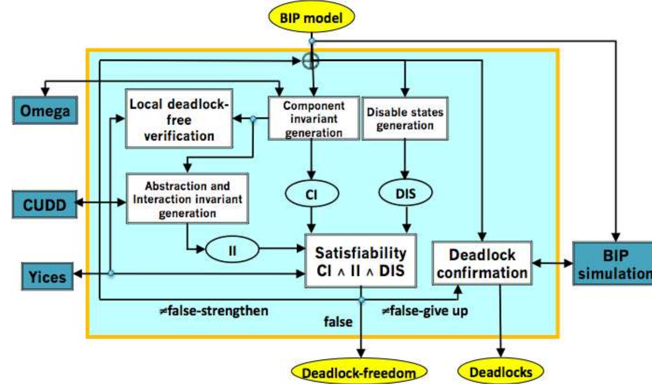


Figure 5: D-Finder tool

The case studies we consider are the Gas Station [21], the Smoker [26], and the ATM [12]. The classical examples of Reader/Writer and Producer/Consumer are also considered. Regarding the Gas Station example, we assume that every pump has 10 customers. Hence, if there are 50 pumps in a Gas Station, then we have 500 customers and the number of components including the operator is thus 551. In the ATM example, every ATM machine is associated to one user. Therefore, if we have 10 machines, then the number of components will be 22 (including the two components that describe the Bank).

The computation times for the application of the three methods on these case studies are given in Table 1. Regarding the legend of the table, $scale$ is the "size" of examples; $location$ denotes the total number of control locations; $interaction$ is for the total number of interactions; IMP provides the amount of time for computing invariants and checking deadlock-freedom using the technique of [5]; LFP is for the LFP method presented in Section 3, and $ILFP$ is for the $ILFP$ method presented in Section 4. The computation time is given in minutes. The timeout, i.e., "-" is one hour. For the incremental method, we start with all the components, and progressively add interactions between them. As an example, for the Smoker case study, we start with interactions for 300 smokers, then add the interactions for 300 more smokers and so on. We observe that, for the classical case studies, $ILFP$ and LFP perform better than the IMP method. In Figures 6(a) and 6(b) , we also compare the memory consumed by the three methods on the Gas Station and Smoker examples. Again, both LFP and $ILFP$ are more efficient than IMP.

In addition to these quite classical benchmarks, we also provide results on *Utopar*, that is an automated transportation system. *Utopar* is one of the two main case studies of the European project COMBEST [17]. A succinct description of the Utopar case study can be found at http://www.combest.eu/home/?link=Application2. Rougly speaking, the Utopar system is the composition of three types of components that are: (1) autonomous vehicles, called U-cars, (2) a centralized Automatic Control System, and (3) Calling Units. The centralized Automatic Control System and the Calling Units have (almost exclusively) discrete behavior. On the other hand, 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. The system is deadlock-free if there always exists some U-car that can respond a request from either a Calling Unit, the Automatic Control System or a Customer inside the U-car. In this paper, we have analyzed a simplified version of Utopar by abstracting from data exchanged between components as well as from continuous dynamics of the U-cars. In this version, each U-car is modeled by a component having 7 control locations and 6 integer variables. The Automatic Control System has 3 control locations and 2 integer variables. The Calling Units have 2 control locations and no variables. In Table 2, one can see that our techniques scale very well, while the IMP one runs out of time quite quickly. Also, we observe that the incremental version is always faster.

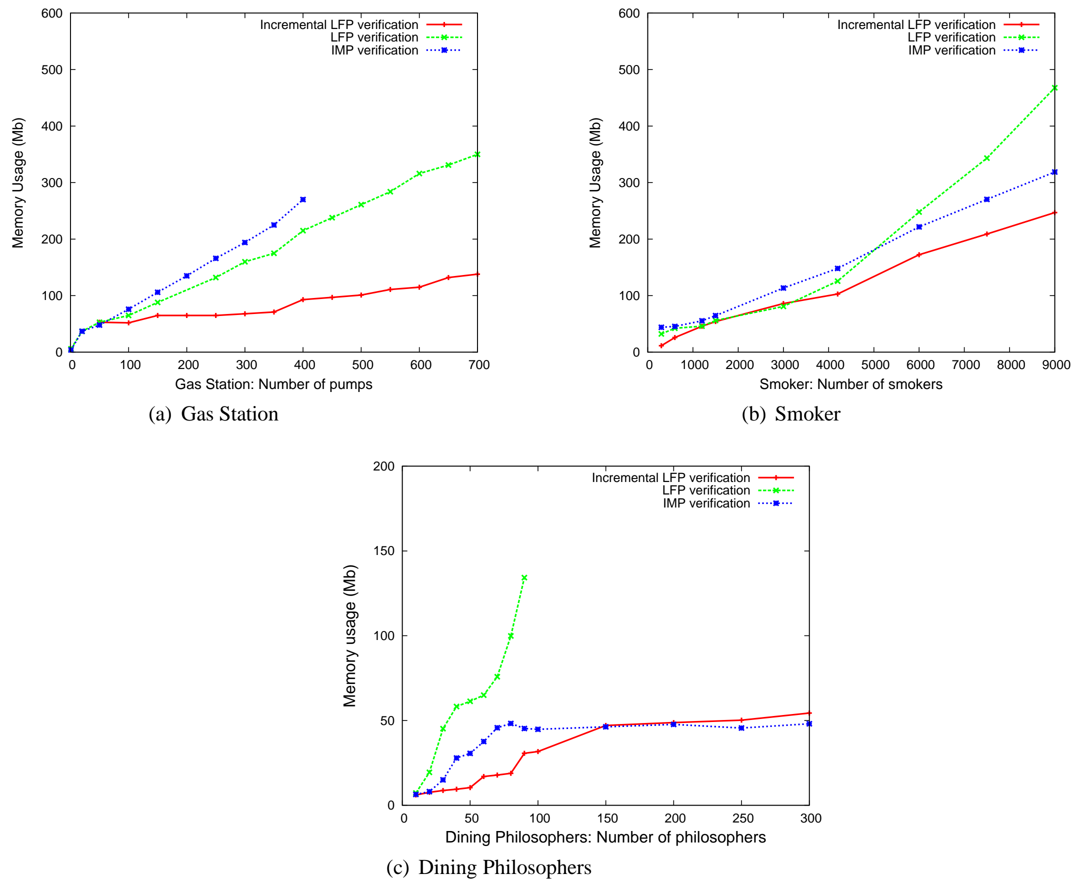
On the bad side, we have observed that for the case of cyclic topologies, the performance may decrease with the increasing number of components. This is illustrated with the Dining Philosophers for which IMP is much faster (see Table 3) and consumes less memory (See Figure 6(c)) than the two methods presented in this paper. The reason is that in the case of cyclic topologies, one often has to compute interaction invariants with constraints that involve all the components in the cycle *at the same time*. The symbolic

Table 1: Comparison between different invariant computation methods for the Gas Station, the Smoker, the Reader/Writer, and the Producer/Consumer case studies.

Gas Station					
scale	location	interaction	*IMP*	*LFP*	*ILFP*
50 pumps	2152	2000	0:50	0:17	0:17
100 pumps	4302	4000	2:58	0:38	0:52
200 pumps	8602	8000	11:34	1:34	1:55
300 pumps	12000	12902	26:00	2:53	2:51
400 pumps	17202	16000	47:38	5:01	3:51
500 pumps	21502	20000	-	7:45	4:43
600 pumps	25802	24000	-	11:21	5:53
700 pumps	30102	28000	-	16:04	7:14
Smoker					
scale	location	interaction	*IMP*	*LFP*	*ILFP*
300 smokers	907	903	0:07	0:06	0:07
600 smokers	1807	1803	0:13	0:18	0:14
1200 smokers	3607	3603	1:06	0:40	0:34
1500 smokers	4507	4503	1:38	0:59	0:44
3000 smokers	9007	9003	6:21	3:43	1:57
4200 smokers	12607	12603	13:21	6:39	3:11
6000 smokers	18007	18003	27:03	14:45	5:57
7500 smokers	22507	22503	41:38	22:44	8:29
9000 smokers	27007	27003	-	32:52	11:36
ATM					
scale	location	interaction	*IMP*	*LFP*	*ILFP*
10 machines	224	182	0:25	0:30	0:27
50 machines	1104	902	10:49	3:17	2:20
100 machines	2204	1802	43:00	6:50	6:00
150 machines	3304	2702	-	10:00	9:30
250 machines	5504	4002	-	17:56	17:16
350 machines	7704	6302	-	39:35	27:54
Reader/Writer					
scale	location	interaction	*IMP*	*LFP*	*ILFP*
400 readers	806	804	0:04	0:06	0:05
1000 readers	2006	2004	0:12	0:11	0:13
2000 readers	4006	4004	0:40	0:31	0:32
3000 readers	6006	6004	1:23	0:56	1:01
5000 readers	10006	10004	4:43	3:02	2:17
6000 readers	12006	12004	5:28	3:40	3:19
Producer/Consumer					
scale	location	interaction	*IMP*	*LFP*	*ILFP*
1000 consumers	2004	2003	0:11	0:12	0:12
2000 consumers	4004	4003	0:27	0:27	0:33
4000 consumers	8004	8003	1:27	1:19	1:18
6000 consumers	12004	12003	3:01	2:40	2:32
8000 consumers	16004	16003	5:35	5:20	4:22
10000 consumers	20004	20003	8:44	8:40	6:12
12000 consumers	24004	24003	12:06	11:02	8:37

Table 2: Comparison between different invariant computation methods on the Utopar case study.

scale	location	interaction	*IMP*	*LFP*	*ILFP*
100 cars, 400 units	1503	40500	3:35	0:59	0:56
200 cars, 400 units	2203	80600	8:05	2:15	1:45
300 cars, 400 units	2303	120700	13:38	3:45	2:29
400 cars, 400 units	2903	160800	20:32	6:08	3:46
100 cars, 900 units	2503	91000	17:52	2:47	2:44
200 cars, 900 units	3203	181100	38:41	7:11	4:59
300 cars, 900 units	3903	271200	-	23:30	7:18
100 cars, 1600 units	3903	161700	59:30	12:02	5:53
200 cars, 1600 units	3903	161700	-	-	17:46



(a) Gas Station

(b) Smoker

(c) Dining Philosophers

Figure 6: Comparison of memory usage between different verification methods

implementation of [5] starts with a formula that tries to interleave the behaviors of all the components simultaneously. In general, the formula has to be refined in order to build the interaction invariant. As stated above, this does not seem to be the case for cyclic topologies. On the other hand, the fixed point characterization will only build the constraints in an interative way and hence it will take some time until the fixed point is reached. This explains the difference in performances: in general $ILFP$ and LFP only work with small formulas that represent the behaviors of a subset of components, but sometimes they have to build a huge one. In such situation, they are less efficient than a technique that directly works with all the components from the very beginning. On the good side, one can see that $ILFP$ can still compete with IMP. Since our incremental principle is quite general, we have good hope to build an incremental version of IMP, which should be much faster than the current version.

The analysis on experimental results also provides us a heuristic on how to construct a component-based system incrementally to facilitate the verification process. We can always choose much related interactions

Table 3: Comparison between different invariant computation methods on Dining Philosophers

scale	location	interaction	IMP	LFP	$ILFP$
50 philosophers	300	250	0:04	4:34	0:05
60 philosophers	360	300	0:06	9:36	0:06
70 philosophers	420	350	0:07	18:59	0:07
80 philosophers	480	400	0:09	30:46	0:09
90 philosophers	540	450	0:11	49:14	0:11
100 philosophers	600	500	0:13	-	0:20
150 philosophers	900	750	0:29	-	0:47
200 philosophers	1200	1000	0:52	-	1:18
250 philosophers	1500	1250	1:23	-	2:20
300 philosophers	1800	1500	2:19	-	3:54

(the interactions specifying the synchronizations within a group of components) into one increment, which may reduce the computation when we put the solutions of increments together to obtain all interaction invariants by the $ILFP$ method.

6 Conclusion

In this paper, we have presented two new techniques $ILFP$ and LFP for computing interaction invariants for a composite component described within the BIP framework. The concept of interaction invariant for BIP models was introduced by Bensalem et al. in [5]. In [5], the authors also propose a global methodology, called IMP, for computing such invariants.

The LFP method computes solutions for interaction invariants by computing local reachable states of components from global interactions. The $ILFP$ method uses directly the solutions from the separate application of set of interactions over the same set of components.

As we have seen, $ILFP$ and LFP work faster and consume less memory than IMP in the case of acyclic topologies. On the other hand, for cyclic topologies, IMP seems to be more efficient.

As a future work, we plan to work on extensions of D-Finder that should be capable to handle systems with stacks. This would allow us to model recursion. Another perspective is to combine D-Finder with existing techniques for computing the set of reachable states of an infinite-state system (which can be viewed as a component invariant) that manipulate unbounded data [3, 9, 10]. Finally, we are also considering extensions of the tool that could handle linear temporal logic properties [28].

References

[1] M. Abadi and L. Lamport. Conjoining specification. *ACM Transactions on Programming Languages and Systems*, 17(3):507–534, 1995. 1

[2] R. Alur and T. Henzinger. Reactive modules. In *Proceedings of the 11th Annual Symposium on LICS*, pages 207–208. IEEE Computer Society Press, 1996. 1

[3] A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *Proc. 12th Int. Conference on Computer Aided Verification (CAV)*, volume 1855 of *Lecture Notes in Computer Science*, pages 419–434. Springer, 2000. 6

[4] A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in BIP. In *SEFM '06*, pages 3–12, Pune, India, 2006. 2.1

[5] S. Bensalem, M. Bozga, T.-H. Nguyen, and J. Sifakis. Compositional verification for component-based systems and application. In *ATVA*, pages 64–79, Seoul, 2008. 1, 3, 1, 3.1, 5.1, 5.2, 5.2, 6

[6] S. Bensalem, M. Bozga, T.-H. Nguyen, and J. Sifakis. D-Finder: A tool for compositional deadlock detection and verification. In *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 614–619. Springer, 2009. 1, 1, 5

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

[8] BIP – incremental component-based construction of real-time systems. http://www-verimag.imag.fr/ async/bip.php. 5.1

[9] B. Boigelot. *Symbolic Methods for Exploring Infinite State Spaces*. Collection des publications de la Faculté des Sciences Appliquées de l'Université de Liège, Liège, Belgium, 1999. 6

[10] B. Boigelot, A. Legay, and P. Wolper. Iterating transducers in the large (extended abstract). In *Proc. 15th Int. Conference on Computer Aided Verification (CAV)*, Lecture Notes in Computer Science, pages 223–235. Springer, 2003. 6

[11] K. Chandy and J.Misra. *Parallel program design: a foundation*. Addison-Wesley Publishing Company, 1988. 1

[12] M. R. V. Chaudron, E. M. Eskenazi, A. V. Fioukov, and D. K. Hammer. A framework for formal component-based software architecting. In *In Workshop on Specification and Verification of Component-Based Systems*, 2001. 5.2

[13] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new symbolic model checker. *Int. Journal on STTT*, 2:2000, 2000. 1

[14] E. Clarke, D. Long, and K. McMillan. Compositional model checking. In *Proc. of the 4th Annual Symposium on LICS*, pages 353–362. IEEE Press, 1989. 1

[15] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. The MIT Press, 1999. 1

[16] J. M. Cobleigh, G. S. Avrunin, and L. A. Clarke. Breaking up is hard to do: An evaluation of automated assume-guarantee reasoning. *ACM Trans. Softw. Eng. Methodol.*, 17(2), 2008. 1

[17] Combest. http://www.combest.eu.com. 5.2

[18] B. Dutertre and L. de Moura. A fast linear-arithmetic solver for DPLL(T). In *CAV'06*, volume 4144 of *LNCS*, pages 81–94, 2006. 5.1, 1

[19] G. Gössler and J. Sifakis. Composition for component-based modeling. *Sci. Comput. Program.*, 55(1-3):161–183, 2005. 2.1

[20] O. Grumberg and D. E. Long. Model checking and modular verification. *ACM Transactions on Programming Languages and Systems*, 16(3):843–871, 1994. 1

[21] D. Heimbold and D. Luckham. Debugging Ada tasking programs. *IEEE Softw.*, 2(2):47–57, 1985. 5.2

[22] J-P.Queille and J.Sifakis. Specification and verification of concurrent systems in cesar. In *Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*. Springer, 1982. 1

[23] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 1–33, Washington, D.C., 1990. IEEE Computer Society Press. 1

[24] O. Kupferman and M. Y. Vardi. An automata-theortetic approach to modular model checking. *ACM Trans. Program. Lang. Syst.*, 22(1):87–128, 2000. 1

[25] K. L. McMillan. A compositional rule for hardware design refinement. In *CAV '97*, pages 24–35. Springer-Verlag, 1997. 1

[26] S. S. Patil. *Limitations and Capabilities of Dijkstra's Semaphore Primitives for Coordination among Processes*. Cambridge, Mass.: MIT, Project MAC, Computation Structures Group Memo 57, Feb, 1971. 5.2

[27] D. Peled. All from one, one for all: on model checking using representatives. In *CAV '93: Proceedings of the 5th International Conference on Computer Aided Verification*, pages 409–423, London, UK, 1993. Springer-Verlag. 1

[28] A. Pnueli. The temporal logic of programs. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:46–57, 1977. 6

[29] A. Pnueli. In transition from global to modular temporal reasoning about programs. *Logics and models of concurrent systems*, pages 123–144, 1985. 1

[30] J. Sifakis. A framework for component-based construction extended abstract. In *SEFM '05: Proceedings of the Third IEEE International Conference on Software Engineering and Formal Methods*, pages 293–300, Washington, DC, USA, 2005. IEEE Computer Society. 2.1

[31] F. Somenzi. CUDD: CU decision diagram package release 2.2.0. 5.1

[32] E. W. Stark. A proof technique for rely/guarantee properties. In *FSTTCS: proceedings of the 5th conference*, volume 206, pages 369–391. Springer-Verlag, 1985. 1

[33] O. Team. The Omega library. Version 1.1.0, November 1996. 5.1, 1

[34] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proc. 2nd IEEE Symposium on Logic in Computer Science (LICS)*, pages 332–344. IEEE Computer Society, 1986. 1

[35] P. Wolper and P. Godefroid. Partial-order methods for temporal verification. In *Proc. 4th Int. Conference on Concurrency Theory (CONCUR)*, volume 715 of *Lecture Notes in Computer Science*, pages 233–246. Springer, 1993. 1