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 Component-based Construction and Deadlock Checking

Saddek Bensalem *Thanh-Hung Nguyen* *Joseph Sifakis*
Rongjie Yan

Verimag Research Report n^o TR-2009-12

September 09, 2009

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

Incremental Component-based Construction and Deadlock Checking

Saddek Bensalem *Thanh-Hung Nguyen* *Joseph Sifakis* *Rongjie Yan*

September 09, 2009

Abstract

We study a methodology for checking incrementally deadlock-freedom of component-based systems. A system is obtained as the composition of atomic components by using interactions. Each interaction expresses strong synchronization (rendezvous) between actions of the components. We improve the heuristic verification method applied by the D-Finder tool to BIP components. The method consists in computing symbolically global invariants for composite components by solving a set of boolean behavioral constraints.
The new incremental verification method allows the computation of these global invariants by using a decomposition of the constraints based on the structure of a composite component. We formalize the construction process of a composite component from a set of atomic components. We provide results relating invariants of constituent components used in the construction process, to global invariants. In particular, we show how the boolean behavioral constraints of the composed components are related to those of the product system.
Experimental results by using the D-Finder tool for checking deadlock-freedom show significant gains in performance with respect to the global verification technique.

Keywords: component-based, BIP, connectors, incremental construction, incremental verification, deadlock-freedom.

How to cite this report:

```
@techreport { Verimag-TR-2009-12,
title = { Incremental Component-based Construction and Deadlock Checking},
authors = { Saddek Bensalem  Thanh-Hung Nguyen  Joseph Sifakis  Rongjie Yan},
institution = {  Verimag Research Report },
number = {TR-2009-12},
year = { 2009},
}
```

1 Introduction

Component-based design techniques confer numerous advantages, in particular through reuse of existing components. A key issue is the existence of composition frameworks ensuring the correctness of composite components. In particular, we need frameworks allowing not only reuse of components but also reuse of their properties for establishing global properties of composite components from properties of their constituent components. This should help cope with the complexity of global monolithic verification techniques.

Compositionality allows inferring global properties of complex systems from properties of their components. One approach to compositional verification is by assume-guarantee where properties are decomposed into two parts. One is an assumption about the global behavior of the environment of a component; the other is a property guaranteed by the component when the assumption about its environment holds. This approach has been extensively studied (see for example [2, 1, 11, 9, 15, 17, 19, 22]). Many issues make the application of assume-guarantee rules difficult. These are discussed in detail in [12] 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.

A different verification method is presented in [5, 6]. It allows computing invariants of a composite system by using a n-ary composition operation parameterized by a set of interactions. The computed invariants are the conjunction of individual invariants of the composed components and invariants induced by interactions. These invariants can be computed automatically from a set of *boolean behavioral constraints* specifying the effect of interactions on the global state space. This method has been implemented in the D-Finder [6] verification tool. It has been successfully applied for checking deadlock-freedom of complex systems described in the BIP (Behavior, Interaction, Priority) [4] language.

In this paper, instead of computing boolean constraints globally, we study a significant improvement of this method based on a construction process leading to a composite component through a sequence of constituent components. The sequence starts from a set of atomic components and applies incrementally synchronization constraints to preserve the global invariants. *Incremental verification* relates the verification process to system construction, which covers the cases of the increasing number of interactions and the tighter of synchronizations. It takes advantage of the system structure for coping with complexity of monolithic verification.

Besides reusing the invariants from lower levels, the incrementality can also be obtained by the separated computation of boolean behavioral constraints for every *increment*, then the generation of global invariants from the partial solutions generated from the boolean behavioral constraints. It shows a significant speedup over deadlock-freedom checking. The contribution opens the way for step-wise and modular verification methodologies tightly coupled with system construction methodologies.

The organization of the paper is as follows.

In Section 2, we use a subset of BIP to formalize the incremental construction process. A composite component is obtained as the composition of a set of atomic components. Composition is by synchronization between specified atomic components. We also present the symbolic method used by D-Finder for computing global invariants for composite components.

In Section 3, the main contribution of the paper, we formalize the process of incremental construction based on the operation of increment of a connector. At some stage of the construction, a component can be transformed only by an increment operation which enforces synchronization between interactions of its connectors. The construction is hierarchical: increments can be applied either at the same level or at different levels. We associate with incremental construction, a method for incremental computation of global invariants of a composite component from the invariants of its constituent components.

In Section 4, we present the symbolic implementation of the incremental verification method. The experimental results show significant improvement of performance for incremental verification with respect to existing techniques.

Finally we discuss related works in Section 5 and conclude in Section 6.

2 Preliminaries

BIP is a component framework for constructing systems by superposing three layers of modeling: Behavior, Interaction, and Priority. In BIP, component behavior is described by transition systems. Interactions express coordination and communication between independent components. In this section, we present basic definitions for BIP used in the paper, and its verification methodology.

2.1 Basic Model for BIP

We provide a formalization of atomic components in BIP.

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

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

Given a transition $\tau = (l, p, l') \in \mathcal{T}$, l and l' are respectively, the *source* and the *target* state denoted respectively by ${}^\bullet\tau$ and $\tau^\bullet$. We extend this notation for ports: ${}^\bullet p = \{{}^\bullet\tau | \tau = (l, p, l')\}$ and $p^\bullet = \{\tau^\bullet | \tau = (l, p, l')\}$ are respectively the set of source and target states of the transitions with port p.

Definition 2 (Path) *Let $B = (L, P, \mathcal{T})$ be a component and two states $l,\ l' \in L$. We say there is a path from l to l', denoted by $l \xrightarrow[B]{*} l'$, if there exists a sequence $\sigma = l_0 \xrightarrow{p_0} l_1 \xrightarrow{p_1} \cdots \xrightarrow{p_{n-1}} l_n$ such that $l = l_0$ and $l' = l_n$, where $p_i \in P$ and $l_i \xrightarrow{p_i} l_{i+1} \in \mathcal{T}$ for $0 \leq i < n$. The set of states of the path σ is denoted by $L_{|\sigma}$.*

A component is said to have a deadlock, if there exists some $l \in L$ such that no outgoing transitions from l can be executed.

Next we define parallel composition for components parameterized by a set of interactions.

Definition 3 (Interactions) *Given a set of components $B_1, B_2, \ldots, B_n$, where $B_i = (L_i, P_i, \mathcal{T}_i)$, an interaction a is a set of ports, i.e., a subset of $\bigcup_{i=1}^n P_i$, such that $\forall i = 1, \ldots, n.\ |a \cap P_i| \leq 1$.*

Definition 4 (Parallel Composition) *Given n components $B_i = (L_i, P_i, \mathcal{T}_i)$, we define the parallel composition $B = \gamma(B_1, \ldots, B_n)$ as the component $(L, \gamma, \mathcal{T})$, where:*

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

Notation: We use $\gamma_\perp(B)$ to denote the interleaving of individual components, where $\gamma_\perp = \bigcup_{i=1}^n P_i$.

Example 1 *[Client-Server] Figure 1 gives the simplified model for Client-Server system for one server and two clients. Every client has to send a request (r_i) to the server then wait for an acknowledgement (a_i) from the server for every action, where $0 \leq i \leq 2$. We require the synchronization of sending and receiving actions between the clients and the server. The interactions for this model are $\{r_0,\ r_1\}$, $\{r_0,\ r_2\}$, $\{a_0,\ a_1\}$, $\{a_0,\ a_2\}$.*

To avoid heavy set-theoretic notation, we define a simple algebraic notation for representing connectors on a set of ports P.

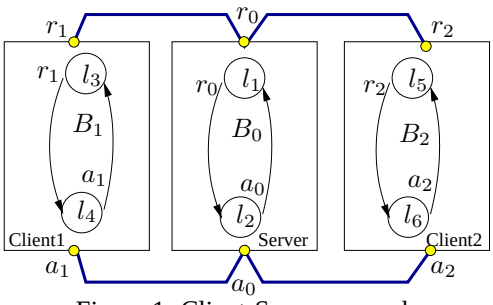


Figure 1: Client-Server example

Definition 5 (Connectors) *[7] Let P be a set of ports, the syntax of the algebra of connectors, $\mathcal{AC}(P)$, is defined by*

$$\gamma ::= p \mid \gamma \mid \gamma \cdot \gamma \mid \gamma + \gamma$$

for $p \in P$, where $+$ and $\cdot$ are binary operators.

Definition 6 (Semantics of Connectors) *The semantics of connectors is given by the function $||\cdot|| : \mathcal{AC}(P) \rightarrow 2^{2^P}$, defined by*

$$||p|| = \{\{p\}\} \text{ for any } p \in P,$$
$$||\gamma_1 + \gamma_2|| = ||\gamma_1|| \cup ||\gamma_2||, \quad ||\gamma_1 \cdot \gamma_2|| = \{\{a_1 \cup a_2\} \mid a_1 \in ||\gamma_1||, a_2 \in ||\gamma_2||\}$$

where $\gamma_1, \gamma_2 \in \mathcal{AC}(P)$, and a_1, a_2 are interactions.

We omit fusion operator $\cdot$ to simplify notation. For example, the connector $\{\{r_0, r_1\}, \{r_0, r_2\}, \{a_0, a_1\}, \{a_0, a_2\}\}$ is represented by the term $r_0\ r_1 + r_0\ r_2 + a_0\ a_1 + a_0\ a_2$.

2.2 Boolean Behavioral Constraints

In [5] we have presented the verification method for component-based systems in BIP. The method uses a heuristic to compute symbolically invariants of a composite component. These invariants are used in particular for proving deadlock-freedom. For this, it is sufficient to find an invariant that does not contain deadlock states. The method is iterative. It allows computing progressively stronger invariants as the result of an iterative process.

Definition 7 (Invariants) *Given $B = (L, P, \mathcal{T})$, a state predicate I is an invariant of B, denoted by $inv(B, I)$, if for any state $l \in L$ and any port $p \in P$, $I(l)$ and $l \xrightarrow[B]{p} l' \in \mathcal{T}$ imply $I(l')$.*

Clearly, if $I_1,\ I_2$ are invariants of B then $I_1 \wedge I_2$ and $I_1 \vee I_2$ are invariants of B.

In the rest of the paper we consider states of components as boolean variables. We use $Bool[L]$ to denote the free boolean algebra generated by the set of states L. We extend the notation ${}^\bullet p,\ p^\bullet$ to interactions. If $a = p_1 \cdots p_m$ is an interaction we take ${}^\bullet a = \bigcup_{i=1}^m {}^\bullet p_i$ and $a^\bullet = \bigcup_{i=1}^m p_i^\bullet$.

Definition 8 (Boolean Behavioral Constraints (BBCs)) *Let γ be a connector over a tuple of components $B = (B_1, \cdots, B_n)$. The boolean behavioral constraints for a component $\gamma(B)$ with set of states L, are defined by a function $|\cdot| : \gamma(B) \rightarrow Bool[L]$ such that:*

$$|\gamma(B)| = \bigwedge_{a \in \gamma} |a(B)|, \quad |a(B)| = \bigwedge_{l_i \in {}^\bullet a} (l_i \Rightarrow \bigvee_{l'_j \in a^\bullet} l'_j)$$

If $\gamma = \emptyset$, then $|\gamma(B)| = true$, which means that no interactions between the components of B will be considered. In Figure 2, if $a = p_1 \cdots p_i \cdots p_m \in \gamma$ is an interaction between transitions $l_i \xrightarrow{p_i} l'_i$ for $i = 1, \cdots, m$, the corresponding constraint is $\bigwedge_{i=1}^m (l_i \Rightarrow \bigvee_{j=1}^m l'_j)$.

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

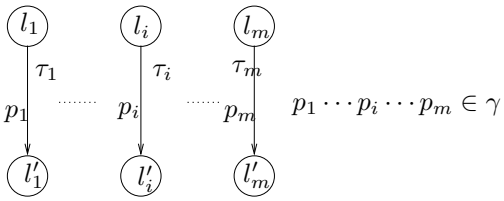


Figure 2: Interactions and implications

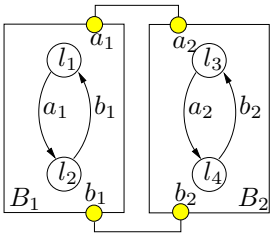


Figure 3: Example for global invariants

Proof 1 *According to Definition 8, the constraints are 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 $l = (l_1, \cdots, l_n)$, there exists l_i such that $v(l_i) = true$. If from l_i there is an interaction a such that $l_i \in {}^\bullet a$, then there exists $l'_j \in a^\bullet$, such that $v(l'_j) = true$ by Definition 8. So any successor state of l by an interaction a satisfies the invariant.*

This theorem allows the computation of invariants of $\gamma(B)$ that we call BBC-invariants. We call global invariant of $\gamma(B)$ the conjunction of all its BBC-invariants.

Definition 9 (Positive Mapping) *Given two set of variables X, Y such that $X \subseteq Y$, and a boolean function $f \in Bool[L]$ expressed as the disjunction of monomials. We define the function that deletes all the negative variables that do not belong to X, denoted by $f^{p(X)}$:*

$$(\bigwedge_{l_i \in Y} l_i \wedge \bigwedge_{l_j \in X} \bar{l_j} \bigwedge_{l_k \in Y - X} \bar{l_k})^{p(X)} = \bigwedge_{l_i \in Y} l_i \wedge \bigwedge_{l_j \in X} \bar{l_j}, \;\; (f_1 \vee f_2)^{p(X)} = f_1^{p(X)} \vee f_2^{p(X)}$$

When X is empty, the positive mapping will remove all the negative variables in f, which is denoted by f^p. If all the variables are negative in f, we have $f^p = false$.

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

Proof 2 *(Sketch). $|\gamma(B)|$ can be written as the disjunction of monomials, that is $|\gamma(B)| = \bigvee_{i \in I} m_i$, where m_i is of the form $m_i = \bigwedge_{j \in I} l_{i_j} \wedge \bigwedge_{k \in K} \overline{l_{i_k}}$. For each m_i, the set of positive literals defines a solution of BBC. And $\widetilde{m_i^p}$ is the corresponding invariant according to Theorem 1.*

Example 2 *We use Figure 3 to illustrate the computation of invariants, where $B = (B_1, B_2)$ and $\gamma = a_1 a_2 + b_1 b_2$. The boolean behavioral constraint for $\gamma(B)$ is $|\gamma(B)| = (l_1 \Rightarrow l_2 \vee l_4) \wedge (l_2 \Rightarrow l_1 \vee l_3) \wedge (l_3 \Rightarrow l_2 \vee l_4) \wedge (l_4 \Rightarrow l_1 \vee l_3) = (\bar{l_1} \wedge \bar{l_2} \wedge \bar{l_3} \wedge \bar{l_4}) \vee (l_1 \wedge l_2) \vee (l_2 \wedge l_3) \vee (l_1 \wedge l_4) \vee (l_3 \wedge l_4)$. The positive mapping deletes negative variables. Thus the global invariant is $\widetilde{|\gamma(B)|^p} = (l_1 \vee l_2) \wedge (l_3 \vee l_4) \wedge (l_1 \vee l_4) \wedge (l_2 \vee l_3)$.*

3 Incremental Construction and Verification

In component-based systems, the construction of a composite component is hierarchical and step-wise. We assume that a system is obtained from a set of atomic components represented by their behavior by adding progressively interactions. At some stage of the construction we have a component $\gamma(B)$ and a set of established invariants. We want to make sure that for each new interaction - an incremental modification of the behavior - the already established invariants are preserved.

3.1 Looser Synchronization Preorder

In this subsection, we will formalize the *looser synchronization preorder* between connectors. Then, we present properties for invariant preservation for this preorder.

Definition 10 (Looser synchronization Preorder) *Given a set of ports P, we define the looser synchronization relation $\preccurlyeq \subseteq 2^{2^P} \times 2^{2^P}$. For two connectors γ_1, γ_2, $\gamma_1 \preccurlyeq \gamma_2$ if for any interaction $b \in \gamma_2$, there exists some interaction $a \in \gamma_1$, such that $a \subseteq b$. We simply say that γ_1 is looser than γ_2.*

Proposition 1 *The looser synchronization relation $\preccurlyeq$ is a preorder. Furthermore, if $\gamma_1,\ \gamma_2, \gamma_3,\ \gamma_4$ are connectors such that $\gamma_1 \preccurlyeq \gamma_2$, and $\gamma_3 \preccurlyeq \gamma_4$, then we have $\gamma_1 \cdot \gamma_3 \preccurlyeq \gamma_2 \cdot \gamma_4$ and $\gamma_1 + \gamma_3 \preccurlyeq \gamma_2 + \gamma_4$.*

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

Proof 3 *If $inv(\gamma_1(B), I)$, for any $l \in L_{|_{l \xrightarrow[\gamma_1(B)]{*} l'}}$, we have $I(l)$. Because $\gamma_1 \preccurlyeq \gamma_2$, we have $L_{|_{l \xrightarrow[\gamma_2(B)]{*} l'}} \subseteq L_{|_{l \xrightarrow[\gamma_1(B)]{*} l'}}$. We can conclude that for any $l \in L_{|_{l \xrightarrow[\gamma_2(B)]{*} l'}}$, $I(l)$ is true. Hence $inv(\gamma_1(B), I) \Rightarrow inv(\gamma_2(B), I)$.*

In the next subsection, we propose a method for building composite components from simpler ones by incremental transformations. These transformations merge connectors and build new ones that synchronize stronger than the connectors of the constituents.

3.2 Incremental Construction

In the incremental construction of component-based systems, layers of connectors are applied to build the system bottom-up. $\gamma_\perp(B)$ can be viewed as the initial system obtained as the interleaving of individual components, where $B = (B_1, \ldots, B_n)$. If at some stage of the construction, we have obtained a system $\gamma(B)$. We transform this system by enforcing new synchronizations between elements of γ. We call these synchronizations *increments*. In doing this bottom-up construction, it is essential that some already enforced synchronizations are not relaxed. For this we need the notion of forbidden interactions for a connector γ.

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 the interactions strictly contained in all the interactions of γ. That is $\gamma^f = \gamma^c - \gamma$.*

Clearly, for connectors γ_1 and γ_2 we have $(\gamma_1+\gamma_2)^c = \gamma_1^c+\gamma_2^c$ and $(\gamma_1+\gamma_2)^f = (\gamma_1+\gamma_2)^c-\gamma_1-\gamma_2$.

Definition 12 (Incremental Construction) *Given a connector γ, an increment δ is any set of interactions obtained by fusion of interactions of γ. For $\delta \subseteq 2^\gamma$, we define $\delta\gamma = (\gamma - \delta^f) + \delta$ the incremental modification of γ by δ.*

The above definition describes one-layer incremental construction. By the successive application of increments, we can construct the system with multiple layers.

The following proposition shows the looser synchronization relation between the connectors involved in the incremental construction process, which preserves the invariants.

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

Proof 4 *As $\delta \subseteq 2^\gamma$, for every interaction $a \in \delta$, there exists at least one interaction $b \in \gamma$, such that $b \subseteq a$. Therefore, it follows that $\gamma \preccurlyeq \delta$. Because $\gamma \preccurlyeq \gamma - \delta^f$, we have $\gamma \preccurlyeq (\gamma - \delta^f) + \delta = \delta\gamma$.*

The independent application of two increments δ_1 and δ_2 on the same connector γ may lead to inconsistencies. For example, if δ_1 enforces synchronization ab that is a and b alone are forbidden, it may happen that δ_2 allows one of the forbidden interactions. Given two increments δ_1 and δ_2, if $\delta_1 \cap \delta_2^f \neq \emptyset$ or $\delta_2 \cap \delta_1^f \neq \emptyset$, there is *interference* between δ_1 and δ_2. In case of interference between δ_1 and δ_2, the invariants induced by δ_1 may not be preserved by δ_2. We define the operation $\oplus$ of superposition of increments which eliminates the interference between increments.

Definition 13 (Superposition) *Given two increments $\delta_1, \delta_2 \subseteq 2^\gamma$, the operation of superposition between δ_1 and δ_2, denoted by $\delta_1 \oplus \delta_2$, is defined by: $\delta_1 \oplus \delta_2 = (\delta_1 - \delta_2^f) + (\delta_2 - \delta_1^f)$.*

Remark 1 *As $\delta \cap \delta^f = \emptyset$, we have $(\delta_1 - \delta_2^f) + (\delta_2 - \delta_1^f) = (\delta_1 + \delta_2) - (\delta_1^f + \delta_2^f)$. In case $\delta_1 \cap \delta_2^f = \emptyset$ and $\delta_2 \cap \delta_1^f = \emptyset$, we have* interference-free superposition *$\delta_1 \oplus \delta_2 = \delta_1 + \delta_2$.*

Proposition 4 *For δ_1, δ_2, δ_3 three increments, we have: (1) $(\delta_1 \oplus \delta_2)^f = \delta_1^f + \delta_2^f$, (2) associativity: $(\delta_1 \oplus \delta_2) \oplus \delta_3 = \delta_1 \oplus (\delta_2 \oplus \delta_3)$, and (3) commutativity: $\delta_1 \oplus \delta_2 = \delta_2 \oplus \delta_1$.*

Proof 5 *The third proposition can be proved from the definition of superposition. We provide the proofs for the first and the second propositions.*

1. *According to the definition of forbidden interactions, $(\delta_1 \oplus \delta_2)^f = (\delta_1 \oplus \delta_2)^c - (\delta_1 \oplus \delta_2)$. By the definition of superposition, we have $(\delta_1 \oplus \delta_2)^f = ((\delta_1 + \delta_2) - (\delta_1^f + \delta_2^f))^c - ((\delta_1 + \delta_2) - (\delta_1^f + \delta_2^f))$. According to Definition 11, for any interaction $a \in \delta_1$ (or δ_2), there always exists some interaction $b \in (\delta_1 + \delta_2) - (\delta_1^f + \delta_2^f)$ such that $a \subseteq b$. Therefore we have $((\delta_1 + \delta_2) - (\delta_1^f + \delta_2^f))^c = (\delta_1 + \delta_2)^c$. Then we have $(\delta_1 \oplus \delta_2)^f = (\delta_1 + \delta_2)^c - ((\delta_1 + \delta_2) - (\delta_1^f + \delta_2^f)) = \delta_1^f + \delta_2^f$.*

2. *According to the definition of superposition, we have $(\delta_1 \oplus \delta_2) \oplus \delta_3 = ((\delta_1 \oplus \delta_2) - \delta_3^f) + (\delta_3 - (\delta_1 \oplus \delta_2)^f)$. As $(\delta_1 \oplus \delta_2)^f = \delta_1^f + \delta_2^f$, we have $(\delta_1 \oplus \delta_2) \oplus \delta_3 = (((\delta_1 + \delta_2) - (\delta_1^f + \delta_2^f)) + \delta_3) - (\delta_1^f + \delta_2^f + \delta_3^f) = (\delta_1 + \delta_2 + \delta_3) - (\delta_1^f + \delta_2^f + \delta_3^f)$. Similarly, we have $\delta_1 \oplus (\delta_2 \oplus \delta_3) = (\delta_1 + ((\delta_2 + \delta_3) - (\delta_2^f + \delta_3^f))) - (\delta_1^f + \delta_2^f + \delta_3^f) = (\delta_1 + \delta_2 + \delta_3) - (\delta_1^f + \delta_2^f + \delta_3^f)$. Then we conclude that $(\delta_1 \oplus \delta_2) \oplus \delta_3 = \delta_1 \oplus (\delta_2 \oplus \delta_3)$ and $\oplus$ is associative.*

Since $\oplus$ is associative, it can be extended to the superposition of any finite number of increments.

Lemma 1 *Given n increments $\{\delta_i\}_{1 \leq i \leq n}$, the superposition of n increments is*

$$\oplus_{i=1}^n \delta_i = \sum_{i=1}^{n} (\delta_i - \sum_{j \neq i} \delta_j^f) \tag{1}$$

Proof 6 *The proof can be constructed by the induction of n.*
If $n = 1$, δ_1 is unchanged. Suppose that $\oplus_{i=1}^{k-1} \delta_i = \sum_{i=1}^{k-1} (\delta_i - \sum_{j \neq i} \delta_i^f)$ for $n = k - 1$. We try to prove that $\oplus_{i=1}^{k} \delta_i = \sum_{i=1}^{k} (\delta_i - \sum_{j \neq i} \delta_j^f)$ for $n = k$.
By the definition of superposition, we have $(\oplus_{i=1}^{k-1} \delta_i) \oplus \delta_k = (\oplus_{i=1}^{k-1} \delta_i - \delta_k^f) + (\delta_k - (\oplus_{i=1}^{k-1} \delta_i)^f)$. By the assumption above and Proposition 4, we obtain $(\oplus_{i=1}^{k-1} \delta_i) \oplus \delta_k = (\sum_{i=1}^{k-1} (\delta_i - \sum_{j \neq i} \delta_i^f) - \delta_k^f) + (\delta_k - \sum_{i=1}^{k-1} \delta_i^f)$. By moving δ_k^f into $\sum_{i=1}^{k-1} (\delta_i - \sum_{j \neq i} \delta_i^f)$, we have $(\oplus_{i=1}^{k-1} \delta_i) \oplus \delta_k = \sum_{i=1}^{k-1} (\delta_i - (\sum_{j \neq i} \delta_i^f + \delta_k^f)) + (\delta_k - \sum_{i=1}^{k-1} \delta_i^f) = \sum_{i=1}^{k} (\delta_i - \sum_{i \neq j} \delta_i^f)$.

Proposition 5 *Let γ be a connector over B, the incremental construction by the superposition of n increments $\{\delta_i\}_{1 \leq i \leq n}$ is*

$$(\oplus_{i=1}^n \delta_i)\gamma = (\gamma - \sum_{i=1}^{n} \delta_i^f) + \sum_{i=1}^{n} (\delta_i - \sum_{j \neq i} \delta_j^f) \tag{2}$$

Proof 7 *(Sketch). By application of Definition 12 and Lemma 1, we get the correctness of the formula.*

Example 3 *In Example 1, if we consider the components together, the initial connector is the set of ports in Figure 1, this is $\gamma_\perp = r_0 + r_1 + r_2 + a_0 + a_1 + a_2$. Two increments $\delta_1, \delta_2 \subseteq 2^{\gamma_\perp}$ are the set of interactions for the two clients to access to the server and get the response, where $\delta_1 = r_0\ r_1 + a_0\ a_1$, $\delta_2 = r_0\ r_2 + a_0\ a_2$. As $\delta_1 \cap \delta_2^f = \emptyset$ and $\delta_2 \cap \delta_1^f = \emptyset$, we have $\gamma' = (\delta_1 \oplus \delta_2)\gamma_\perp = r_0\ r_1 + r_0\ r_2 + a_0\ a_1 + a_0\ a_2$.*

3.3 Incremental Computation of BBCs

The conjunction between the invariants of the constituents of a composite component fails to take into account the effect of superposition. Therefore, we provide an incremental method to obtain global BBCs in this subsection.

Lemma 2 *Given two connectors γ_1, γ_2 over B, we have*

$$|(\gamma_1 + \gamma_2)(B)| = |\gamma_1(B)| \wedge |\gamma_2(B)|$$

Proof 8 *By Definition 8, we have* $|(\gamma_1+\gamma_2)(B)| = \bigwedge_{a\in(\gamma_1+\gamma_2)} |a(B)| = \bigwedge_{a\in\gamma_1} |a(B)| \wedge \bigwedge_{a\in\gamma_2} |a(B)| = |\gamma_1(B)| \wedge |\gamma_2(B)|$.

The following proposition provides the methods for obtaining the boolean behavioral constraints taking into account component structure.

Proposition 6 *Let γ be a connector over B, the boolean behavioral constraint for the system obtained by superposition of n increments $\{\delta_i\}_{1\leq i\leq n}$ can be written as*

$$|(\oplus_{i=1}^n \delta_i)\gamma(B)| = |(\gamma - \sum_{i=1}^n \delta_i^f)(B)| \wedge \bigwedge_{i=1}^n |(\delta_i - \sum_{i\neq j} \delta_j^f)(B)| \tag{3}$$

Proof 9 *By Equation 2, the union of $\gamma - \sum_{i=1}^n \delta_i^f$ and $\sum_{i=1}^n(\delta_i - \sum_{i\neq j} \delta_j^f)$ is the set of interactions from the superposition of increments $\{\delta_i\}_{1\leq i\leq n}$ over γ. The proof can be concluded by the application of Lemma 2.*

Example 4 *For Example 1, consider the components $\delta_1\gamma_\perp(B)$ and $\delta_2\gamma_\perp(B)$ where $\delta_1 = r_0\ r_1 + a_0\ a_1,\ \delta_2 = r_0\ r_2 + a_0\ a_2$.*

The composite component is $\gamma'(B)$ where $\gamma' = (\delta_1 \oplus \delta_2)\gamma_\perp = r_0\ r_1 + a_0\ a_1 + r_0\ r_2 + a_0\ a_2$.

1. *The BBC for interactions of δ_1 (involving $Client_1$) are:*

$$r_0\ r_1: \quad (l_1 \Rightarrow l_2 \vee l_4) \wedge (l_3 \Rightarrow l_2 \vee l_4); \quad a_0\ a_1: \quad (l_2 \Rightarrow l_1 \vee l_3) \wedge (l_4 \Rightarrow l_1 \vee l_3)$$

2. *The BBC for interactions of δ_2 (involving $Client_2$) are:*

$$r_0\ r_2: \quad (l_1 \Rightarrow l_2 \vee l_6) \wedge (l_5 \Rightarrow l_2 \vee l_6); \quad a_0\ a_2: \quad (l_2 \Rightarrow l_1 \vee l_5) \wedge (l_6 \Rightarrow l_1 \vee l_5)$$

Because $\gamma_\perp - (\delta_1^f + \delta_2^f) = \emptyset$, $\delta_1 \cap \delta_2^f = \emptyset$ and $\delta_2 \cap \delta_1^f = \emptyset$, we have $|(\delta_1 \oplus \delta_2)\gamma_\perp(B)| = |\delta_1\gamma_\perp(B)| \wedge |\delta_2\gamma_\perp(B)|$ where $|\delta_1\gamma_\perp(B)|$ and $|\delta_2\gamma_\perp(B)|$ are the BBCs above.

By application of Theorem 2, we can obtain the invariants for $\gamma'(B)$. For example, $l_1 \vee l_4 \vee l_6$ is one of the computed invariants.

3.4 Incremental Computation of Global Invariants

In the previous section we have shown how to compute incrementally BBC. The BBC of a composite component can be obtained as the conjunction of BBCs of constituent components. For incremental computation of global invariants, we try to apply Theorem 2 to each one of the conjuncts. The computation of the positive mapping of all the variables over the conjunction of multiple BBCs can also be partitioned into two steps:

1. the positive mapping over the variables shared by more than one BBC for the partitioned interactions, that is, all the negative variables belonging to only one BBC will be removed.

2. removing all remaining negative variables over the conjunction of all BBCs.

To distinguish the shared variables within BBCs, we define the common state variables shared by multiple connectors as follows:

Definition 14 (Common State Variables L_c) *Given $\{\gamma_i\}_{1\leq i\leq n}$, a set of connectors , we define the set of common state variables by $L_c = \bigcup_{i,j\in[1,n]\wedge i\neq j} support(\gamma_i) \cap support(\gamma_j)$, where $support(\gamma) = \bigcup_{a\in\gamma} {}^\bullet a \cup a^\bullet$, the set of states involved in some interaction a of γ.*

Proposition 6 provides the way to decompose the computation of positive mappings over common state variables for boolean behavioral constraints. Decomposition is based on the fact that different increments describe the interactions between different components.

Proposition 7 *Given a component $\gamma(B)$, $\{\delta_i\}_{1\leq i\leq n}$ a set of increments of γ, and L_c the set of common state variables for the set of connectors $\gamma - \sum_{i=1}^{n}\delta_i^f, \{\delta_i - \sum_{i\neq j}\delta_j^f\}_{1\leq i\leq n}$, we have*

$$|(\oplus_{i=1}^{n}\delta_i)\gamma(B)|^p = (|(\gamma - \sum_{i=1}^{n}\delta_i^f)(B)|^{p(L_c)} \wedge \bigwedge_{i=1}^{n}|(\delta_i - \sum_{i\neq j}\delta_j^f)(B)|^{p(L_c)})^p \quad (4)$$

Each non common variable occurs only in one of the BBCs. This allows deleting the non common negative variables separately, which drastically reduces complexity of computation.

4 Symbolic Implementation and Experimental Results

In this section, we explain the symbolic implementation of incremental verification. Experimental results show the benefits from the partitioned interactions in computing the invariants.

4.1 Boolean Representation

As shown in [7], interactions and connectors can be represented by boolean functions on the set of ports P. Boolean representations can facilitate the comparison between the connectors and their implementation.

$\mathcal{AC}(P)$ can be bijectively mapped to the free boolean algebra $Bool[P]$ generated by P, which is shown in the following definition.

Definition 15 (Boolean Representation for Connectors) *We define a mapping $\beta : \mathcal{AC}(P) \rightarrow Bool[P]$ by setting:*

$$\beta(p_1 \dots p_k) = \bigwedge_{i=1}^{k} p_i \wedge \bigwedge_{j\notin[1,k]} \bar{p_j}, \quad \beta(\gamma_1 + \gamma_2) = \beta(\gamma_1) \vee \beta(\gamma_2)$$

for $\gamma_1, \gamma_2 \in \mathcal{AC}(P)$, $p_1, \dots, p_k \in P$, where, in the right-hand side, the elements of P are considered to be boolean variables.

The mapping β is an isomorphism between the set the interactions and the boolean algebra on P [7]. Any boolean expression on P defines a set of interactions. The interaction $p_1 \cdots p_k$ is possible if the expression is true for the valuation that makes true all the ports belonging to the interaction and makes false all the ports do not belonging to the interaction. Each expression $\gamma \in \mathcal{AC}(P)$ represents exactly the set of interactions corresponding to boolean valuations of P satisfying $\beta(\gamma)$. With this mapping, other related concepts can also be given boolean representations, which are defined as follows.

1. The boolean function for the closure of the interactions is defined by

$$\beta^c(p_1 \cdots p_k) = \bigvee_{i=1}^{k} p_i \wedge \bigwedge_{j\notin[1,k]} \bar{p_j}, \quad \beta^c(\gamma_1 + \gamma_2) = \beta^c(\gamma_1) \vee \beta^c(\gamma_2).$$

2. The boolean function for the forbidden interactions is defined by

$$\beta^f(p_1 \dots p_k) = \beta^c(p_1 \cdots p_k) \wedge \overline{\beta(p_1 \cdots p_k)}, \ \beta^f(\gamma_1 + \gamma_2) = \beta^c(\gamma_1 + \gamma_2) \wedge \overline{\beta(\gamma_1)} \wedge \overline{\beta(\gamma_2)}.$$

Example 5 *For the connector $\gamma = pqr + pq + q$, the boolean representations are:* $\beta(\gamma) = p \wedge q \vee q \wedge \bar{r}$,$\beta^f(\gamma) = ((p \vee q \vee r) \vee (p \vee q) \wedge \bar{r} \vee q \wedge \bar{p} \wedge \bar{r}) \wedge \overline{p \wedge q \wedge r} \wedge \overline{p \wedge q \wedge \bar{r}} \wedge \overline{q \wedge \bar{p} \wedge \bar{r}}$.

With these boolean representations, the boolean representation for the incremental construction with n-ary superposition is

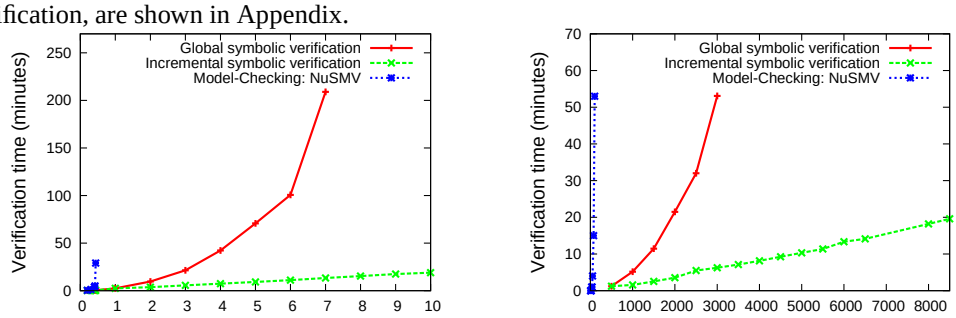
$$\beta((\oplus_{i=1}^{n}\delta_i)\gamma) = \beta(\gamma) \wedge \bigwedge_{i=1}^{n}\overline{\beta^f(\delta_i)} \vee \bigwedge_{i=1}^{n}(\beta(\delta_i) \wedge \bigwedge_{i\neq j}\overline{\beta^f(\delta_j)}) \quad (5)$$

As BBCs are boolean constraints, we have a fully symbolic representation of connectors and BBCs.

4.2 Experimental Results

We have implemented the incremental computation method in the D-Finder tool by using the CUDD package [21].

To compare the performance of incremental method with other verification methods, we take Gas Station and Dining Philosophers as benchmarks by increasing the scale of component-based systems, which cover deadlock and deadlock-free cases. The details of the two cases, together with their way of incremental verification, are shown in Appendix.



(a) Gas Station (b) Dining Philosophers

Figure 4: The comparison between the incremental verification and global verification

Figure 4 shows timing performance for deadlock-freedom verification of two scalable examples by using: incremental symbolic computation of BBC-invariants, global symbolic computation of BBC-invariants and verification by the NuSMV [10] model-checker. Incremental symbolic computation improves performance dramatically.

For Gas Station, D-Finder proves that it is deadlock-free. The verification time is a function of the number N, where if N increases by one, 50 pumps and 500 customers are added. For Dining Philosophers, D-Finder detects the potential deadlocks. When the number of philosophers is odd, we can detect exactly one deadlock. However, we get two more false deadlocks when the number of philosophers is even. The time shown in the figure is the total consumption after obtaining all the potential deadlocks. Besides these two examples, we have applied incremental symbolic verification to industrial case studies. For instance, the verification of the NDD module in the Robot system [3] is completed within 5 minutes, while 3 hours are needed for global symbolic computation. Detailed results for case studies are available at http://www-verimag.imag.fr/~ thnguyen/tool.

5 Related Work

Many works have focused on deadlock detection. It is well known that model checking approaches suffer from state space explosion problem. Compositional verification is one of the ways to alleviate the state space explosion. However, within compositional verification methodologies, the main difficulties of assume-guarantee techniques are to find decompositions into sub-systems and choose adequate assumptions for a particular decomposition.

Some works such as [8, 14], apply machine learning during assume-guarantee reasoning to make the method complete, but the algorithms are complex [12].

The work based on split invariance [13] addresses the incompleteness problem by adding auxiliary variables in case of property violation. However, the premise of the work is that the conjunction of local assertions is an inductive invariant of the composite system, which is not always true in synchronous systems. And no explicit claims on the completeness of deadlock detection is mentioned.

The least fixpoint approximation method proposed in [18] computes the tighter upper bound of the reachable states locally, by repeated image computation of a set of boolean functions from some initial states. However, the decomposition of state space is an important issue to get the tighter approximation of reachable states. The structure of the system will influence the approximation and the iteration times.

A recent paper [20] proposes a compositional deadlock detection technique for a kind of concurrent language. The method composes tasks one by one explicitly to detect deadlocks. Both the time and memory consumption will increase greatly with the increasing number of tasks.

Earlier work in [5, 6] is primarily on global computation of boolean behavior constraints to obtain the global invariants. During the system construction, the invariants for every stage have to be re-computed to check the deadlock-freedom. The incremental method proposed in the paper is capable of reusing the available invariants, and computing the global invariants incrementally from its increments. As the approximation of the reachable state space, both methods are sound but not complete. If we obtain some potential deadlocks, we are not sure whether they are real deadlocks without checking their reachability.

During the incremental construction process, the feature of the incremental method is to compute invariants incrementally in a totally symbolic way. The computed invariants will not differ from different partition of connectors.

6 Conclusion

The paper studies a method for checking deadlock-freedom of a component-based system. The method takes advantage of properties of the construction process based on the assumption that composite components can be obtained from a set of atomic components by superposition of increments.

The method is an adaptation of a heuristic for computing invariants of component-based systems. Its implementation is fully symbolic. In addition to boolean behavioral constraints it also uses boolean representations for connectors, for their increments. The benchmarks obtained with D-Finder show significant performance improvements with respect to global verification.

The application of incremental verification methods to systems with data will require the use of abstractions for atomic components, following the same approach as in [5, 6]. Although our method is explained in the environment of BIP, it can be broadly applied to the verification of concurrent systems. The proposed view opens the way for step-wise and modular verification methodologies tightly coupled with system construction methodologies.

Currently, the deadlocks obtained from D-Finder are potential deadlocks. To remove these potential deadlocks, we should start from the potential deadlocks and find constraints to strengthen the global invariants. We plan to investigate this issue in the future.

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. Basu, S. Bensalem, M. Gallien, F. Ingrand, C. Lesire, T.-H. Nguyen, and J. Sifakis. Incremental component-based construction and verification of a robotic system. In *18th European Conf. on Artificial Intelligence (ECAI)*, 2008. 4.2

[4] A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in bip. In *SEFM '06*, pages 3–12, Washington, DC, USA, 2006. 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, 2.2, 5, 6

[6] S. Bensalem, M. Bozga, T.-H. Nguyen, and J. Sifakis. D-Finder: A tool for compositional deadlock detection and verification. In *CAV*, Grenoble, France, 2009. 1, 5, 6

[7] S. Bliudze and J. Sifakis. The algeba of connectors – structuring interaction in BIP. *IEEE Transactions on Computers*, 57:1315–1330, October 2008. 5, 4.1, 4.1

[8] S. Chaki, E. Clarke, N. Sinha, and P. Thati. Automated assume-guarantee reasoning for simulation conformance. In *CAV*, volume 3576, pages 534–547. Springer-Verlag, 2005. 5

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

[10] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new symbolic model checker. *Int. Journal on STTT*, 2:2000, 2000. 4.2

[11] 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

[12] 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):1–52, 2008. 1, 5

[13] A. Cohen and K. S. Namjoshi. Local proofs for global safety properties. In W. Damm and H. Hermanns, editors, *CAV*, volume 4590 of *Lecture Notes in Computer Science*, pages 55–67. Springer, 2007. 5

[14] D. Giannakopoulou and C. S. Pasareanu. Learning-based assume-guarantee verification. In *SPIN*, volume 3639, pages 282–287, 2005. 5

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

[16] D. Heimbold and D. Luckham. Debugging Ada tasking programs. *IEEE Softw.*, 2(2):47–57, 1985. A.1

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

[18] I.-H. Moon, J. Kukula, T. Shiple, and F. Somenzi. Least fixpoint approximations for reachability analysis. In *ICCAD '99: Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design*, pages 41–44, Piscataway, NJ, USA, 1999. IEEE Press. 5

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

[20] B. Shao, N. Vasudevan, and S. A. Edwards. Compositional deadlock detection for rendezvous communication. In *EMSOFT*, 2009. 5

[21] F. Somenzi. CUDD: CU decision diagram package release 2.2.0. 4.2

[22] 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

A Examples

A.1 Gas Station

Gas Station [16] consists of an Operator with a computer, a set of pumps, and a set of customers. Each pump can be used by a fixed number of customers. The set of the atomic components involved in a system with n pumps and m customers for each pump is denoted by $B[n,m] = \{ Operator, \{pump_i\}_{1\leq i\leq n}, \{customer_{ij}\}_{1\leq i\leq n, 1\leq j\leq m}\}$.

Before using a pump, each customer has to prepay for the transaction. Then the customer uses the pump, collects his change and goes to a state from which he may start a new transaction.

Before being used by a customer, the pumps have to be activated by the Operator. When a pump is shut off, it can be re-activated for the next operation.

Figure 5 gives the model for Gas Station system for one pump and two customers. The Operator has two states and three ports.The transition labelled with $prepay$ accepts a customer's prepay and activates the pump for the customer. When a customer is served, the transition labelled with $finish$ will synchronize

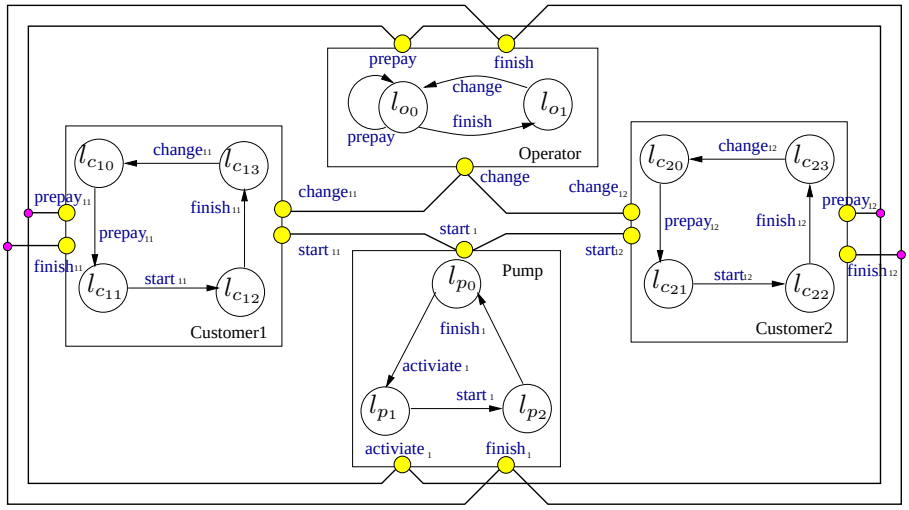


Figure 5: Sketch of Gas Station

the pump and the customer. A pump has three states and three ports. Besides the synchronization between the Operator and customer through $activate$ and $finish$ ports, a pump and a customer are synchronized through $start$ ports.

We abbreviate port names by using only their first three letters. The interactions for a system of n pumps, each one used by m customers, are $\gamma[n,m] = \Sigma_{i=1}^{n}(\Sigma_{j=1}^{m}(pre\ act_i\ pre_{ij} + sta_i\ sta_{ij} + fin\ fin_i\ fin_{ij} + cha\ cha_{ij}))$. For instance, the interactions for $customer_1$ are $pre\ act_1\ pre_{11} + sta_1\ sta_{11} + fin\ fin_1\ fin_{11} + cha\ cha_{11}$.

if we consider the two customers together, the initial connector is the set of ports in Figure 5, that is $\gamma_\perp = pre+fin+cha+act_1+sta_1+fin_1+pre_{11}+sta_{11}+fin_{11}+cha_{11}+pre_{12}+sta_{12}+fin_{12}+cha_{12}$. Two increments $\delta_1, \delta_2 \subseteq 2^{\gamma_\perp}$ are the set of interactions for the two customers to access to the pump and get the service, where

$$\delta_1 = pre\ act_1\ pre_{11} + sta_1\ sta_{11} + fin\ fin_1\ fin_{11} + cha\ cha_{11}$$
$$\delta_2 = pre\ act_1\ pre_{12} + sta_1\ sta_{12} + fin\ fin_1\ fin_{12} + cha\ cha_{12}$$

As $\delta_1 \cap \delta_2^f = \emptyset$ and $\delta_2 \cap \delta_1^f = \emptyset$, we have $\gamma' = (\delta_1 \oplus \delta_2)\gamma_\perp = pre\ act_1\ pre_{11} + sta_1\ sta_{11} + fin\ fin_1\ fin_{11} + cha\ cha_{11} + pre\ act_1\ pre_{12} + sta_1\ sta_{12} + fin\ fin_1\ fin_{12} + cha\ cha_{12}$.

The composite component is $\gamma'(B)$ where $\gamma' = (\delta_1 \oplus \delta_2)\gamma_\perp = pre\ act_1\ pre_{11} + sta_1\ sta_{11} + fin\ fin_1\ fin_{11} + cha\ cha_{11} + pre\ act_1\ pre_{12} + sta_1\ sta_{12} + fin\ fin_1\ fin_{12} + cha\ cha_{12}$.

1. The BBC for interactions of δ_1 (involving $customer_1$) are:

$$\begin{array}{ll} pre\ act_1\ pre_{11}: & (l_{o_0} \Rightarrow l_{o_0} \vee l_{p_1} \vee l_{c_{11}}) \wedge (l_{p_0} \Rightarrow l_{o_0} \vee l_{p_1} \vee l_{c_{11}}) \wedge (l_{c_{10}} \Rightarrow l_{o_0} \vee l_{p_1} \vee l_{c_{11}}) \\ sta_1\ sta_{11}: & (l_{p_1} \Rightarrow l_{p_2} \vee l_{c_{12}}) \wedge (l_{c_{11}} \Rightarrow l_{p_2} \vee l_{c_{12}}) \\ fin\ fin_1\ fin_{11}: & (l_{o_0} \Rightarrow l_{o_1} \vee l_{p_0} \vee l_{c_{13}}) \wedge (l_{p_2} \Rightarrow l_{o_1} \vee l_{p_0} \vee l_{c_{13}}) \wedge (l_{c_{12}} \Rightarrow l_{o_1} \vee l_{p_0} \vee l_{c_{13}}) \\ cha\ cha_{11}: & (l_{o_1} \Rightarrow l_{o_0} \vee l_{c_{10}}) \wedge (l_{c_{13}} \Rightarrow l_{o_0} \vee l_{c_{10}}) \end{array}$$

2. The BBC for interactions of δ_2 (involving $customer_2$) are:

$$\begin{array}{ll} pre\ act_1\ pre_{12}: & (l_{o_0} \Rightarrow l_{o_0} \vee l_{p_1} \vee l_{c_{21}}) \wedge (l_{p_0} \Rightarrow l_{o_0} \vee l_{p_1} \vee l_{c_{21}}) \wedge (l_{c_{20}} \Rightarrow l_{o_0} \vee l_{p_1} \vee l_{c_{21}}) \\ sta_1\ sta_{12}: & (l_{p_1} \Rightarrow l_{p_2} \vee l_{c_{22}}) \wedge (l_{c_{21}} \Rightarrow l_{p_2} \vee l_{c_{22}}) \\ fin\ fin_1\ fin_{12}: & (l_{o_0} \Rightarrow l_{o_1} \vee l_{p_0} \vee l_{c_{23}}) \wedge (l_{p_2} \Rightarrow l_{o_1} \vee l_{p_0} \vee l_{c_{23}}) \wedge (l_{c_{22}} \Rightarrow l_{o_1} \vee l_{p_0} \vee l_{c_{23}}) \\ cha\ cha_{12}: & (l_{o_1} \Rightarrow l_{o_0} \vee l_{c_{20}}) \wedge (l_{c_{23}} \Rightarrow l_{o_0} \vee l_{c_{20}}) \end{array}$$

Because $\gamma_\perp - (\delta_1^f + \delta_2^f) = \emptyset$, $\delta_1 \cap \delta_2^f = \emptyset$ and $\delta_2 \cap \delta_1^f = \emptyset$, we have $|(\delta_1 \oplus \delta_2)\gamma_\perp(B)| = |\delta_1\gamma_\perp(B)| \wedge |\delta_2\gamma_\perp(B)|$ where $|\delta_1\gamma_\perp(B)|$ and $|\delta_2\gamma_\perp(B)|$ are the BBCs above.

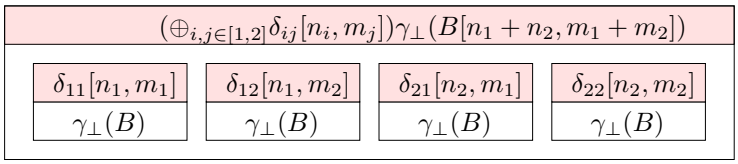


Figure 6: Incremental construction of Gas Station

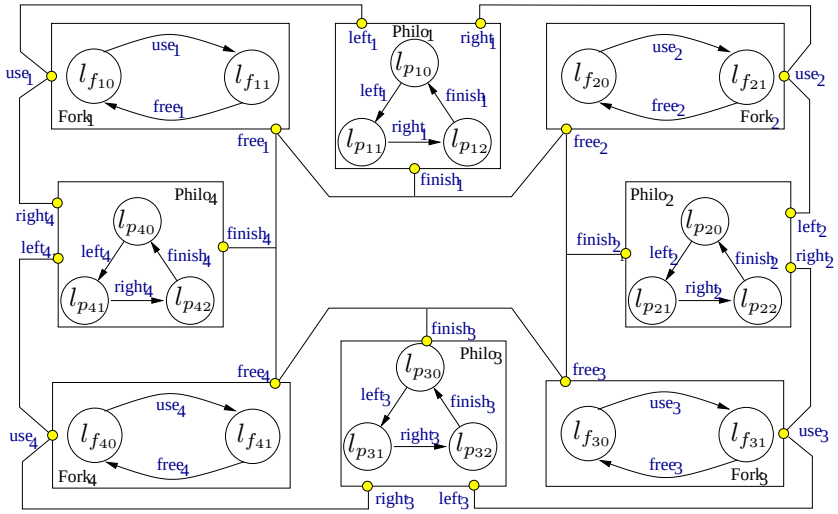


Figure 7: Dining Philosophers

By application of Theorem 2, we can obtain the invariants for $\gamma'(B)$. For example, $l_{o_1} \vee l_{p_1} \vee l_{c_{10}} \vee l_{c_{12}} \vee l_{c_{20}} \vee l_{c_{22}}$ is one of the computed invariants.

The incremental construction of Gas Station, with $n_1 + n_2$ pumps and $m_1 + m_2$ customers for each pump, can be described as follows:

$$\begin{aligned}&\gamma[n_1+n_2,m_1+m_2](B[n_1+n_2,m_1+m_2])\\&=(\delta_1[n_1,m_1+m_2]\oplus\delta_2[n_2,m_1+m_2])\gamma_\perp(B[n_1+n_2,m_1+m_2])\\&=(\delta_{11}[n_1,m_1]\oplus\delta_{12}[n_1,m_2]\oplus\delta_{21}[n_2,m_1]\oplus\delta_{22}[n_2,m_2])\gamma_\perp(B[n_1+n_2,m_1+m_2])\end{aligned}$$

where $\delta[n,m]$ is the set of interactions for n pumps and m customers for each pump in Gas Station. Figure 6 shows the principle for incremental construction for Gas Station with $n_1 + n_2$ pumps and $m_1 + m_2$ customers for each pump.Since there is no interference between the increments $\delta_{ij}[n_i,m_j]$ for $1 \leq i,j \leq 2$, we have $|(\oplus_{i,j\in[1,2]}\delta_{ij}[n_i,m_j])\gamma_\perp(B[n_1+n_2,m_1+m_2])|^p = (\bigwedge_{i,j\in[1,2]}|\delta_{ij}[n_i,m_j](B[n_1+n_2, m_1+m_2])|^{p(L_c)})^p$.

A.2 Dining Philosophers

Figure 7 shows the model for Dining Philosophers. Given n philosophers and n forks, the interactions are:

$$\gamma[n]=\sum_{i=1}^{n}(left_i\ use_i+right_i\ use_{(i\bmod n)+1}+finish_i\ free_i\ free_{(i\bmod n)+1})$$

These interactions can be obtained from the superposition of the following increments

$$\delta_{[1,m]}=\sum_{i=1}^{m}(left_i\ use_i+right_i\ use_{i+1}+finish_i\ free_i\ free_{i+1})$$

and

$$\delta_{[m+1,n]} = \sum_{i=m+1}^{n} (left_i\ use_i + right_i\ use_{(i\ mod\ n)+1} + finish_i\ free_i\ free_{(i\ mod\ n)+1})$$

where $\delta_{[1,m]}$ are the interactions between the first m philosophers, and $\delta_{[m+1,n]}$ are the interactions between the last $m+1$ to n philosophers.

For example, the set of interactions for a system of 4 philosophers is the superposition of the increment $\delta_{[1,2]} = left_1\ use_1 + right_1\ use_2 + finish_1\ free_1\ free_2 + left_2\ use_2 + right_2\ use_3 + finish_2\ free_2\ free_3$ and the increment $\delta_{[3,4]} = left_3\ use_3 + right_3\ use_4 + finish_3\ free_3\ free_4 + left_4\ use_4 + right_4\ use_1 + finish_4\ free_4\ free_1$.