Incremental Generation of Linear Invariants for Component-Based Systems

Saddek Bensalem, Benoit Boyer, Marius Bozga and Axel Legay

Verimag Research Report n^o TR-2012-15

2012 July 1

Incremental Generation of Linear Invariants for Component-Based Systems

Saddek Bensalem, Benoit Boyer, Marius Bozga and Axel Legay

2012 July 1

Abstract

Linear invariant generation has been intensively considered as an effective verification method for concurrent systems. However, none of the existing work on the topic strongly exploits the structure of the system and the algebra that defines the interactions between its components. This not only has an impact on the computation time, but also on the scalability of the method. In a series of recent work, we developed an incremental approach for generating boolean invariants for systems described in the BIP component framework. BIP is an expressive modeling formalism including a rich algebra to describe component interactions. The objective of this paper is to extend and propose new techniques dedicated to the computation of linear interactions invariants, i.e., invariants that are described by linear constraints and that relate states of several components in the system. In particular, we propose an incremental approach that allows to discover and reuse invariants that have already been computed on subparts of the model. Those new techniques have been implemented in DFINDER, a tool for checking deadlock freedom on BIP systems using invariants, and evaluated on several case studies. The experiments show that our approach outperforms classical techniques on a wide range of models.

Keywords: invariant generation, linear invariant, incremental verification, component-based systems

Reviewers: Marius Bozga

Notes: The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7] under grant agreements no 257414 (ASCENS), 288175 (CERTAINTY) and by ARTEMIS Joint Undertaking [JU] under grant agreement No. 2009-1-100208 (ACROSS).

How to cite this report:

```
@techreport {TR-2012-15,
  title = {Incremental Generation of Linear Invariants for Component-Based Systems},
  author = {Saddek Bensalem, Benoit Boyer, Marius Bozga and Axel Legay},
  institution = {{Verimag} Research Report},
  number = {TR-2012-15},
  year = {}
}
```

1 Introduction

Component-based design confers numerous advantages, in particular, an increased productivity through reuse of existing components. Nonetheless, establishing the correctness of the designed systems remains an open issue. In contrast to other engineering disciplines, software and system engineering badly ensures predictability at design time. Consequently, a posteriori verification as well as empirical validation are essential for ensuring correctness. Monolithic verification [1, 2] of component-based systems is a challenging problem. It often requires computing for a composite component the product of its constituents by using both interleaving and synchronization. The complexity of the product system is often prohibitive due to state explosion. A solution to this problem is to generate an invariant that is an abstraction of the state-space of the system.

We observed that most of the existing work on generating invariants for component-based systems are too general and do not strongly exploit the structure of the system and the algebra that defines the interactions between its components. In a series of recent work [3, 4, 5], we proposed novel approaches and the DFINDER tool [6] for generating invariants for systems described in the BIP framework [7]. BIP is an expressive modeling formalism equiped with a rich algebra to describe component interactions. Our techniques start by building invariants for individual components, which can be done with any existing approach for invariant generation on sequential programs. The novel concept in DFINDER is that the invariant for the overall system is then obtained by glueing this set of individual invariants with another one that is an abstraction of the algebra used to define the interactions between the components. By doing so, one avoid building huge part of the state-space before generating the invariant. One of the major advantages of our approach is that it allows for the development of incremental techniques such as [5] capable of reusing invariants that have already been computed on subparts of the model. The incremental approach is particularly useful when multiple instances of the same components (atomic or composite) are used in the system. In such cases, it allows to factorize some part of the analysis. Thus, local invariants established on some part of the system can be automatically lifted to all similar parts within the system.

Until now, the DFINDER approach has been limited to invariants that can be represented by boolean formulas. This has been shown to be convenient in many contexts, going from simple to complex case studies [8]. However, there are situations where boolean invariants may not be appropriate. Consider the state variable at_l_i which monitors that (local) control state l_i of some process is currently active. Whatever the transition relation of the system is, DFINDER will only be able to generate invariants of the form e.g., $at\_l_1 \vee at\_l_2 \vee at\_l_3$. Such an invariant ensures that one of the control states l_1, l_2, l_3 is active, which is sometimes sufficient to infer the deadlock freeness. However, such invariants are not precise enough to prove a mutual exclusion property. Here, an invariant of the form $at\_l_1 + at\_l_2 + at\_l_3 \leq 1$ would be needed, which shows that at maximum one process can be in a critical state at any time.

To reason on such more complex properties, we have to work with invariants capable of *counting* how many processes are at a given states. A way to do this is to use linear invariants, i.e., invariants that can be represented by sets of linear equations. Such invariants have already been studied for a wide range of models for concurrent systems, and in particular for Petri Nets [9]. The objective of this paper is to propose new methods for linear invariant generations in BIP. Our methods build on transitions of components that are abstracted by linear equations and then combined to form a system of equations. We show that each solution of such system is a linear invariant. Solving systems of linear equations can be done with classical techniques such as Gauss-Jordan elimination or LU-factorization. However, those approaches that are general do not exploit the structure of the system under consideration and may scale badly on large size systems. As a solution to this scalability problem, we propose an online algorithm that processes equations in the system in an iterative manner based on the structure of the underlying component-based system. As a second contribution, we proposed an incremental extension of the proposed approach. This approach uses the incremental framework from [5] with the additional difficulty of combining linear equations rather than Boolean ones.

Our new contributions have been implemented in DFINDER and evaluated on several case studies. The experiments show how our approaches outperform classical techniques on a wide range of models. Particularly, our method is as efficient as the one to compute boolean invariants, and it allows for finer state-space approximations (hence removing more spurious counter-examples). Finally, our results asses that DFINDER is faster than tools implementing classical mathematical approaches.

Related Work. The literature on generation of boolean invariants for BIP and comparison with other works is wide and partly covered in [3, 4, 5]. There exists an huge amount of literature on automatic generation of linear invariants for different categories of systems and/or programs. In fact, first results have been obtained in the context of hardware systems (see e.g. [10]). The main difference with our work is that we exploit a rich component-based design language that is clearly more expressive than classical Boolean circuits. Work on discovery of linear relations between variables of a sequential program dates back to early days of program verification [11]. Linear invariants have received particular attention for generation methods derived from abstract interpretation [12]. In the former, linear constraints are definitely amongst the most useful, expressive abstract domain for program analysis.

The work on algebraic methods for the generation of so called linear state-invariants for Petri net models is perhaps the most closest to ours. An introductory survey can be found in [13] while several extensions for invariant generation under particular constraints is available in [14]. These methods have been implemented since a long time and tools like CHARLIE [9] are widely known in the Petri net community. While being the most known, these techniques are however neither compositional nor incremental: the invariant generation problem is directly rephrased as linear algebra problem and solved using standard methods. Another method for generating linear invariants for Petri nets has been explored in [15]. This method relies on Farkas lemma as an effective mean for quantifier elimination. Invariant computation is carried transition by transition, and therefore avoid a global computation phase. Nonetheless, this method is not incremental and can be applied only once the system has been entirely constructed.

Structure of the paper. Section 2 recalls some basic definitions used throughout the rest of the paper. Section 3 introduces the component-based framework as well as the basic principles for compositional and incremental design. Section 4 defines linear invariants and Section 5 presents two novel methods for, respectively, global and incremental generation. Finally, section 6 review implementation and the experimental work done to validate the approach.

2 Preliminaries

We denote respectively by $\mathbb{Z}$ and $\mathbb{Q}$ the sets of integer and rational numbers. We consider *homogeneous linear systems* $\mathcal{S}$ of the form $\mathcal{S} \equiv \bigwedge_{i=1}^{m}(\sum_{j=1}^{n} a_{ij}x_j = 0)$ where x_j are integer unknowns and $a_{ij} \in \mathbb{Z}$ are integer coefficients, for all $1 \leq j \leq n$, $1 \leq i \leq m$. Such systems are compactly denoted as $\mathbf{Ax} = \mathbf{0}$ where $\mathbf{A} = (a_{ij})_{1\leq i \leq m, 1 \leq j \leq n} \in \mathbb{Z}^{m \times n}$ is the matrix of coefficients, $\mathbf{x} = (x_j)_{1 \leq j \leq n}$ is the vector of unknowns and $\mathbf{0}$ is the null vector in $\mathbb{Z}^m$. A vector of integers $\mathbf{u} \in \mathbb{Z}^n$ is a *solution* of the system if it satisfies $\mathbf{Au} = \mathbf{0}$. We denote with $Sol(\mathcal{S})$ the set of solutions of the system $\mathcal{S}$. Two systems $\mathcal{S}_1$ and $\mathcal{S}_2$ are called *equivalent* and denoted by $\mathcal{S}_1 \approx \mathcal{S}_2$ if they have the same set of solutions, that is, $Sol(\mathcal{S}_1) = Sol(\mathcal{S}_2)$. For any system $\mathcal{S}$, the set of solutions contains at least the trivial solution which is the null vector $\mathbf{0}_n$ in $\mathbb{Z}^n$. Moreover, if the set $Sol(\mathcal{S})$ contains non-trivial solutions, then it is infinite. In this latter case, we call *solution basis* any minimal (w.r.t. inclusion) set of solutions $\{\mathbf{u}_k\}_{k \in K} \subseteq Sol(\mathcal{S})$ that allows to generate $Sol(\mathcal{S})$ as linear combinations with rational coefficients, formally such that $Sol(\mathcal{S}) = \left\{\sum_{k\in K} \lambda_k \mathbf{u}_k \mid \lambda_k \in \mathbb{Q}\right\} \cap \mathbb{Z}^n$. We known from linear algebra that, for any system $\mathcal{S}$, a solution basis with at most n elements always exists. Such a basis can be effectively computed by using e.g., Gauss-Jordan elimination to transform the system (with an appropiate renaming of variables) into an equivalent *solved* (or *left-bound*) system $\mathcal{S}'$ of the form $\mathcal{S}' \equiv \bigwedge_{i=1}^{m'}(a'_{ii}x_i = \sum_{j=m'+1}^{n} a'_{ij}x_j)$ where $m' \leq m$, $a'_{ii} \neq 0$ for all $1 \leq i \leq m'$. A basis is obtained immediately from the solved form by selecting the set of solutions $\{\mathbf{u}_k\}_{m'+1 \leq k \leq n}$ such that $\mathbf{u}_{ki}$ is equal to (1) $a'_{ik}L/a'_{ii}$ for all $1 \leq i \leq m'$, (2) L, if $i = k$ and (3) 0, for all $m' + 1 \leq i \neq k \leq n$ and where $L = lcm\{a'_{ii} | 1 \leq i \leq m'\}$. For example, the system $2x_1 + 3x_3 - x_4 = 0 \wedge x_2 - 5x_3 + 2x_4 = 0 \wedge 4x_1 + x_2 + x_3 = 0$ can be transformed into the left bound form $2x_1 = -3x_3 + x_4 \wedge x_2 = 5x_3 - 2x_4$ which gives the basis $\{\mathbf{u}_3, \mathbf{u}_4\}$ where $\mathbf{u}_3 = [-3, 10, 2, 0]$ and $\mathbf{u}_4 = [1, -4, 0, 2]$.

3 Component-based Design

In this section, we introduce the underlying concepts for modeling and design of component-based systems.

Our component-based framework is a fragment of the BIP framework [7]. The BIP - *Behavior, Interaction, Priority* - framework allows description of complex, heterogeneous systems in a hierarchical and compositional manner. BIP supports a modeling methodology based on the assumption that components are obtained as the superposition of three layers, that is:

- *behavior*, specified as a set of automata extended with C data and functions,
- *interactions* between the automata, modeled as sets of *structured connectors*,
- *priorities* used to schedule among possible interactions.

In this paper, we restrict ourselves to a strict fragment of BIP, that is, without data and without priorities. In fact, we have previously shown in [3] how data can be taken into account for computing invariants through abstraction. Regarding priorities, we do not consider them, however, let us remark that priorities preserve invariant properties and deadlock-freedom [16].

In the rest of the section, we recall the most relevant concepts useful in this context, that is, *atomic components* and their *parallel composition* through *interactions*. Then, we recap a recent methodology proposed in [17] for *incremental design* of component-based systems with BIP.

3.1 Components and Interactions

In our setting, atomic components are labeled transition systems. Transitions' labels are called *ports* and are used to interact with other components.

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.*

Without loss of generality, we assume that, every port p labels exactly one transition $\tau_p \in T$. Given $\tau_p = (l, p, l') \in \mathcal{T}$, l and l' are the *source* and *destination* locations for τ. These locations are equally denoted respectively as ${}^\bullet\tau$ and $\tau^\bullet$.

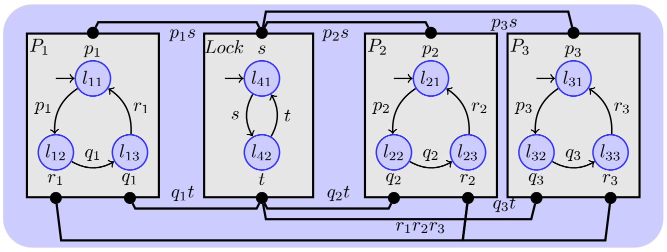


Figure 1: Running example: global composition

Example 1 *Figure 1 presents a simplified variant of the Reader-Writers problem with four atomic components P_1, P_2, P_3 and Lock. The ports of component P_1 are p_1, q_1, r_1. P_1 has three locations l_{11}, l_{12} and l_{13} and three transitions $\tau_1 = (l_{11}, p_1, l_{12})$, $\tau_2 = (l_{12}, q_1, l_{13})$ and $\tau_3 = (l_{13}, r_1, l_{11})$.*

Atomic components are running in parallel and communicate via *interactions*, i.e., by synchronization on ports. Formally, interactions and connectors are defined as follows.

Definition 2 (Interaction, Connector) *Let $\{B_i = (L_i, P_i, \mathcal{T}_i)\}_{i=1}^n$ be a set of atomic components with sets of locations and ports pairwise disjoint, that is, $L_i \cap L_j = \emptyset$ and $P_i \cap P_j = \emptyset$ for all $i \neq j$. An interaction a is a set of ports, that is, a subset of $\bigcup_{i=1}^n P_i$, such that $\forall i = 1, \ldots, n.\ |a \cap P_i| \leq 1$. A connector γ is a set of interactions $\{a_1, \ldots, a_m\}$.*

For the sake of simplicity, we write $p_1p_2\ \ldots\ p_k$ to denote the interaction $\{p_1, p_2, \ldots p_k\}$. We also write $a_1 \oplus \ldots \oplus a_m$ for the connector $\{a_1, \ldots, a_m\}$.

Example 2 *Graphically, interactions are represented by links between ports. The connector represented in Figure 1 consists of six binary and one ternary interactions, respectively* $p_1s \oplus p_2s \oplus p_3s \oplus q_1t \oplus q_2t \oplus q_3t \oplus r_1r_2r_3$.

We use parallel composition parameterized by a connector γ to build *composite components* from atomic components. Any global step of the composite component corresponds to an interaction a of γ. For any such interaction a, only those components that are involved in a can make a step. This is ensured by following a transition labelled by the port used in a. If a component does not participate to the interaction, then it remains in the same location.

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

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

We denote by $\ell \xrightarrow{a} \ell'$ transitions $(\ell, a, \ell') \in \mathcal{T}$. We say that a global state ℓ is reachable from an initial global state ℓ_0 if there exist a sequence of interactions $a_1, \cdots, a_k$ and global states $\ell_1, \cdots, \ell_k$ such that $\ell_0 \xrightarrow{a_1} \ell_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} \ell_k = \ell$.

Moreover, we extend the notation of source and destination to interactions and denote ${}^\bullet a = \{{}^\bullet\tau_p \mid p \in a\}$ and $a^\bullet = \{\tau_p^\bullet \mid p \in a\}$.

Example 3 *The example given in Figure 1 presents the composite component* $\gamma(P_1, P_2, P_3, Lock)$ *where* $\gamma = p_1s \oplus p_2s \oplus p_3s \oplus q_1t \oplus q_2t \oplus q_3t \oplus r_1r_2r_3$.

Let us observe also that any composite component $B = \gamma(B_1, \ldots, B_n)$ can be equivalently seen as a *1-safe*[1] *Petri net* whose set of places is $L = \bigcup_{i=1}^n L_i$, that is, the set of locations of B, and whose transition relation is given by $\mathcal{T}$.

3.2 Incremental Design

In component-based design, the construction of a composite system is both step-wise and hierarchical. Systems are usually obtained from atomic components by successive additions of new interactions also called *increments*. We have proposed in [17] a methodology to add new interactions to a composite component without breaking the existing synchronization. This way, properties enforced by synchronization at some step in the design flow are never lost in successive steps when increments are added.

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

Definition 4 (Increments) *Consider a connector* γ *over atomic components* $B_1, ..., B_n$ *and let* $\delta \subseteq 2^\gamma$ *be a set of interactions. We say* δ *is an increment over* γ *if for any interaction* $a \in \delta$ *there exists* disjoint *interactions* $b_1, \ldots, b_n \in \gamma$ *such that* $\bigcup_{i=1}^n b_i = a$.

In a dual manner, when increments are used, one has also to make sure that existing interactions in 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.

[1] the number of tokens in any place never exceeds one

Definition 5 (Layering) *Given a connector γ and an increment δ over γ, the set of interactions obtained by combining δ and γ, also called layering (or incremental modification of γ by δ), is given by the set $\delta\gamma = (\gamma \ominus \delta^f) \oplus \delta$ where $\delta^f = \{a \mid a \notin \delta \wedge \exists b \in \delta . a \subsetneq b\}$ is the set of interactions forbidden by δ.*

Example 4 *The connector γ illustrated in Figure 1 can be obtained by successive layering from $\gamma_\perp = p_1 \oplus q_1 \oplus r_1 \oplus p_2 \oplus q_2 \oplus r_2 \oplus p_3 \oplus q_3 \oplus r_3 \oplus s \oplus t$. That is, $\gamma = \delta_3\delta_2\delta_1\gamma_\perp$ where (i) increment $\delta_1 = \{p_1s, q_1t, s, t\}$ corresponds to synchronization of P_1 and Lock on p_1s and q_1t while leaving open s and t for further interactions (ii) increment $\delta_2 = \{r_2r_3\}$ corresponds to synchronization of P_2 and P_3 and (iii) finally, increment $\delta_3 = \{p_2s, p_3s, q_2t, q_3t, r_1r_2r_3\}$ enforces the remaining interactions between respectively P_2, P_3, Lock and P_1, P_2, P_3. This incremental construction is illustrated in Figure 2.*

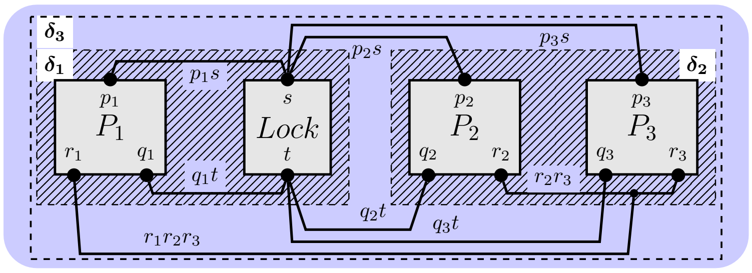


Figure 2: Running example: Incremental composition

4 Linear Invariants

Let $B = \gamma(B_1, \dots, B_n)$ be a composite component obtained by parallel composition using connector γ of atomic components $\{B_i = (L_i, P_i, \mathcal{T}_i)\}_{i=1}^n$. Let $(\mathcal{L}, \gamma, \mathcal{T})$ be the transition system associated to B, as defined by definition 3. Let ℓ_0 be an initial global state of B, fixed.

We consider the set At of *location variables* $\{at\_l \mid l \in L = \cup_{i=1}^n L_i\}$. At any global state $\ell = (l_1, l_2, ..., l_n) \in \mathcal{L}$, each location variable of At is assigned to a binary value through the valuation function $\sigma_\ell : At \to \{0,1\}$. This function characterizes the global state ℓ by mapping to 1 (resp. 0) variables corresponding to locations (resp. not) in ℓ, formally $\sigma_\ell(at\_l) = 1$ iff $\in \{l_1, l_2, ..., l_n\}$ and $\sigma_\ell(at\_l) = 0$ otherwise.

We consider linear equality constraints of the form $\sum_{l \in L} u_l \cdot at\_l = u_0$ built from location variables and with integer coefficients $u_0, u_l \in \mathbb{Z}$ for all $l \in L$. By abuse of notation, we interpret $(u_l)_{l\in L}$ and $(at\_l)_{l \in L}$ as vectors and we denote more compactly the constraints above as $\mathbf{u}^T \cdot At = u_0$. Similarly, we define the particular vector At_0 as $\sigma_{\ell_0}(At_0)$ which denotes the initial valuation of variables at ℓ_0.

Definition 6 (Linear Invariant) *A linear invariant is a linear equality constraint $\mathbf{u}^T \cdot At = u_0$ which hold in all reachable global states of the composite component, that is, for all ℓ reachable from ℓ_0 it holds $\sum_{l\in L} u_l \cdot \sigma_\ell(at\_l) = u_0$.*

Example 5 *In the example of Figure 1, the equality constraint $at\_l_{12} + at\_l_{22} + at\_l_{32} + at\_l_{41} = 1$ is a linear invariant for the composite component with initial global state $(l_{11}, l_{21}, l_{31}, l_{41})$. This linear invariant characterises a mutual exclusion property, that is, at most one process P_1, P_2, P_3 is in its critical location respectively l_{12}, l_{22}, l_{32} at any time.*

If not empty, the set of linear invariants is infinite. For instance, it can be easily checked that if $\mathbf{u}^T \cdot At = u_0$ is a linear invariant, so is $(\lambda \mathbf{u}^T) \cdot At = (\lambda u_0)$ for any integer coefficient $\lambda \in \mathbb{Z}$. In order to provide a finite representations of such sets, we introduce the notion of *basis of linear invariants*, as follows.

Definition 7 (Basis of Linear Invariants) *Let $\mathcal{I}$ be a set of linear invariants. A finite subset $\mathcal{I}_0 \subseteq \mathcal{I}$, $\mathcal{I}_0 = \{\mathbf{u}_k^T \cdot At = u_{0k}\}_{k \in K}$ is a basis for $\mathcal{I}$ if and only if for all invariant $\mathbf{u}^T \cdot At = u_0 \in \mathcal{I}$ there exists rational coefficients $(\lambda_k)_{k\in K} \in \mathbb{Q}$ such that $\mathbf{u} = \sum_{k \in K} \lambda_k \mathbf{u}_k$ and $u_0 = \sum_{k\in K} \lambda_k u_{0k}$.*

5 Automatic Generation of Linear Invariants

Consider a composite component $B = \gamma(B_1, \ldots, B_n)$ with associated transition system $(\mathcal{L}, \gamma, \mathcal{T})$ and initial state ℓ_0.

In Section 5.1, we introduce the global method to compute linear invariants from solutions of the homogeneous system of flow equations which characterizes B. While the theory is a classic, we introduce a new efficient algorithm to resolve such systems. The efficiency of the algorithm will be demonstrated in the experimental section. Section 5.2 proposes a new incremental approach that exploits the algorithm proposed in Section 5.1 together with the architecture of the design. We shall later see that the incremental approach outpeforms the global one.

5.1 Global Approach

We first introduce *characteristic System*, that is a system of linear equations representing the interactions within the BIP model. We then show that solutions of the characteristic systems are indeed linear invariants. This means that computing linear invariant reduces to solving a system of linear equations. Latter, we propose an efficient version of the Gauss-Jordan algorithm that exploits the structure of our specification language.

Definition 8 (Characteristic System) *For a finite set of atomic components $B_1, \ldots, B_n$ synchronized by a connector γ, the characteristic system $\mathcal{S}_G(\gamma, B_1, \ldots, B_n)$ is defined as the conjunction*

$$\mathcal{S}_G(\gamma, B_1, \ldots, B_n) \equiv \bigwedge_{a \in \gamma} \Big(\sum_{l \in a^\bullet} x_l - \sum_{l \in {}^\bullet a} x_l = 0 \Big)$$

The unknowns x_l correspond to locations $l \in L = \cup_{i=1}^{n} L_i$. The characteristic system introduces exactly one flow equation for each interaction a of the system.

Example 6 *The characteristic system for Example 1 following the enumeration of interactions $p_1 s$, $p_2 s$, $p_3 s$, $q_1 t$, $q_2 t$, $q_3 t$, $r_1 r_2 r_3$ is:*

$$\mathcal{S}_G \equiv \left\{ \begin{array}{ccccccccccc}
x_{l_{12}} & & & +x_{l_{42}} & -x_{l_{11}} & & & -x_{l_{41}} & = & 0 \\
& x_{l_{22}} & & +x_{l_{42}} & & -x_{l_{21}} & & -x_{l_{41}} & = & 0 \\
& & x_{l_{32}} & +x_{l_{42}} & & & -x_{l_{31}} & -x_{l_{41}} & = & 0 \\
x_{l_{13}} & & & +x_{l_{41}} & -x_{l_{12}} & & & -x_{l_{42}} & = & 0 \\
& x_{l_{23}} & & +x_{l_{41}} & & -x_{l_{22}} & & -x_{l_{42}} & = & 0 \\
& & x_{l_{33}} & +x_{l_{41}} & & & -x_{l_{32}} & -x_{l_{42}} & = & 0 \\
x_{l_{11}} & +x_{l_{21}} & +x_{l_{31}} & & -x_{l_{13}} & -x_{l_{23}} & -x_{l_{33}} & & = & 0
\end{array} \right.$$

We are now ready to show that solutions of the characteristic system are indeed linear invariants for the corresponding model.

Theorem 1 *Any solution $\mathbf{u}$ of $\mathcal{S}_G$ defines the linear invariant $\mathbf{u}^T \cdot At = \mathbf{u}^T \cdot At_0$ of the composite component B.*

PROOF. Regarding the composite component B as its equivalent Petri-Net PN, the characteristic system of $\mathcal{S}_G$ is equivalent to the equation $C^T x = 0$, where C is incidence matrix of PN. Each solution denotes an invariant of PN (*c.f.* [13]) and thus, an invariant of B □

Theorem 2 *Any set of invariants $\{\mathbf{u}_k^T \cdot At = \mathbf{u}_k^T \cdot At_0\}_{k \in K}$ constructed from a solution basis $(\mathbf{u}_k)_{k \in K}$ of $\mathcal{S}_G$ is a basis for the set of all linear invariants obtained from $\mathcal{S}_G$.*

PROOF. Using the solution basis $(\mathbf{u}_k)_{k \in K}$, all solutions $\mathbf{u}$ can be expressed as a linear combination such that we have the invariant $(\sum_{k \in K} \lambda_k \mathbf{u}_k^T) \cdot At = \sum_{k \in K} \lambda_k \mathbf{u}_k^T) \cdot At_0$. This invariant is trivially implied by the set of $\{\mathbf{u}_k^T \cdot At = \mathbf{u}_k^T \cdot At_0\}_{k \in K}$. □

The common techniques to solve homogeneous systems $\mathbf{Ax} = \mathbf{0}$ are the Gauss-Jordan elimination, Cholesky-, QR- or LU-factorization. These general well-known algorithms have low polynomial complexity and can be directly applied to solve the characteristic system $\mathcal{S}_G$. Nonetheless, naive implementations may badly scale to realistic systems, in particular, if they do not consider carefully the structure and the sparsity of the characteristic systems.

To ensure scalability, we developed an online resolution algorithm (Algorithm 1 below) that processes equations in the characteristic systems iteratively, one by one, while producing an equivalent left-bound system. It is essentially a variant of Gauss-Jordan that exploits the locality of unknowns as well as the particular form of equations. In addition to efficiency that will be demonstrated in Table 1, one of the major advantages of the new algorithm is that its structure can be exploited to derive an incremental version. This shall be the subject of the next section.

Algorithm 1 Online algorithm for direct resolution of $\mathcal{S}_G$

```
1: $LeftB \leftarrow \emptyset$                  ▷ $LeftB \equiv \bigwedge_{i\in I} (x_i = \overbrace{\sum_{j\in J} \lambda_{ij} x_j}^{def_i}),\ I \cap J = \emptyset$
2: while $\neg finished$ do
3:     $eq \leftarrow$ READEQUATION()
4:     $eq \leftarrow$ REWRITE($eq$, $LeftB$)      ▷ $eq$ has the form $\sum_{j\in J} k_j x_j = 0$
5:     if $\neg$TRIVIAL($eq$) then
6:         $(x = def) \leftarrow$ SOLVE($eq$)
7:         $LeftB \leftarrow$ PROPAGATE($LeftB$, $x = def$)
8:         $LeftB \leftarrow LeftB \wedge (x = def)$
9:     end if
10: end while
11: return $LeftB$
```

In the above algorithm, function REWRITE$(eq, LeftB)$ returns the equation eq in which all bounded unknowns x_i are substituted according to their definition def_i given by $(x_i = def_i)$ in $LeftB$. Function PROPAGATE is the dual of REWRITE. PROPAGATE$(LeftB, x = def)$ returns the system $LeftB$ where all occurences of the free (unbounded) unknowns x are substitued by def. When SOLVE(eq) is called, $eq \equiv \sum k_j x_j = 0$ contains only free unknowns. One of them is selected and the equation is rewritten into a solved form $x = def$. The choice is led by prefering the x_j with the smallest absolute value for k_j.

Our algorithm has been implemented in DFINDER. Experimental results and comparison with similar tools/methodologies are reported in section 6.

Example 7 *Using Algorithm 1, the characteristic system $\mathcal{S}_G$ given in Example 6 is transformed in left bound form shown below left. The solution basis extracted from the solved form generates $\mathcal{I}_0$ the basis of linear invariants.*

$$\mathcal{S}_G \equiv \left\{ \begin{array}{lcl} x_{l_{12}} & = & x_{l_{32}} + x_{l_{13}} - x_{l_{33}} \\ x_{l_{42}} & = & x_{l_{41}} - x_{l_{32}} + x_{l_{33}} \\ x_{l_{22}} & = & x_{l_{32}} + x_{l_{23}} - x_{l_{33}} \\ x_{l_{11}} & = & x_{l_{13}} \\ x_{l_{21}} & = & x_{l_{23}} \\ x_{l_{31}} & = & x_{l_{33}} \end{array} \right. \qquad \mathcal{I}_0 = \left\{ \begin{array}{rcl} at\_l_{13} + at\_l_{12} + at\_l_{11} & = & 1 \\ at\_l_{23} + at\_l_{22} + at\_l_{21} & = & 1 \\ at\_l_{33} + at\_l_{32} + at\_l_{31} & = & 1 \\ at\_l_{41} + at\_l_{42} & = & 1 \\ at\_l_{12} + at\_l_{22} + at\_l_{32} + at\_l_{41} & = & 1 \end{array} \right.$$

5.2 Incremental Approach

The incremental approach allows to organize the computation of linear invariants by following the incremental design process. Actually, incremental design provides a natural and meaningful manner to *split* the global characteristic system and to optimize its resolution.

The incremental approach relies on construction and manipulation of *incremental* characteristic systems. For a composite component, this characteristic system characterizes both (1) the existing interactions defined inside and (2) the still open possibilities for further interaction (inside or with extra components).

Definition 9 (Incremental Characteristic System) *For a finite set of atomic components $B_1, \dots, B_n$ synchronized by a connector γ, the incremental characteristic system $\mathcal{S}_I(\gamma, B_1, \dots, B_n)$ is defined as the conjunction*

$$\mathcal{S}_I(\gamma, B_1, \dots, B_n) \equiv \bigwedge_{a \in \gamma} \big(\sum_{l \in a^\bullet} x_l - \sum_{l \in {}^\bullet a} x_l - y_a = 0 \big)$$

The main difference with the global characteristic system is that, in addition to unknowns x_l associated to locations $l \in L = \cup_{i=1}^n L_i$, the incremental system uses unknowns y_a associated to interactions $a \in \gamma$. These unknowns are used to capture the (still) partial composition through γ. Every unknown y_a can be interpreted as denoting the partial flow realized on interaction a in the current composition by γ. Intuitively, any further extension of γ through layering will simply add extra constraints on the y_a unknowns, and preserve entirely the existing equations involving x_l unknowns.

When the parallel composition is completed, that is, no more interactions are added, the global characteristic system can be obtained from the incremental system simply by substituting with the constant 0 all the unknowns that correspond to the interactions. We define this operation as *freezing* of interaction constraints.

Theorem 3 (Freezing) *For every composite component $\gamma(B_1, \dots, B_n)$, the characteristic system $\mathcal{S}_G$ is obtained from the incremental characteristic system $\mathcal{S}_I$ as follows:*

$$\mathcal{S}_G \approx (\exists y_a)_{a \in \gamma} \big(\mathcal{S}_I \wedge \bigwedge_{a \in \gamma} y_a = 0 \big)$$

PROOF. The proof is trivial: the substitution of each unknown y_a by 0 in $\mathcal{S}_I$ gives syntactically the system $\mathcal{S}_G$. $\square$

This equivalence allows to establish that linear invariants are preserved through freezing. If $\mathbf{u}$ is a solution of the incremental characteristic system which assigns 0 to all y_a unknowns then, its restriction $\mathbf{u}_{|L}$ to x_l unknowns is a solution of global characteristic system. Such solutions $\mathbf{u}$ of incremental systems are called hereafter invariant-generating. By using observation above and theorem 1 it holds that $\mathbf{u}_{|L}$ defines a linear invariant for the composite component $\gamma(B_1, ..., B_n)$ for any invariant-generating solution $\mathbf{u}$ of $\mathcal{S}_I(\gamma, B_1, ..., B_n)$.

The main advantage of incremental systems is that they are easily transformed through layering. That is, there exist a strong relationship between the incremental systems, before and after layering, as stated by the following theorem.

Theorem 4 (Layering) *Given composite component $\gamma(B_1, \dots, B_n)$ and δ an increment of γ, it holds that*

$$\mathcal{S}_I(\delta\gamma, B_1, \dots, B_n) \approx (\exists y_b)_{b \in \gamma \cap \delta^f} \Big(\mathcal{S}_I(\gamma, B_1 \cdots B_n) \wedge \bigwedge_{a \in \delta} \big(y_a - \sum_{b_k \in \gamma, \sqcup_k b_k = a} y_{b_k} = 0 \big) \Big)$$

PROOF. By definition of layering, $\delta\gamma = (\gamma \ominus \delta^f) \oplus \delta$. The incremental characteristic system $\mathcal{S}(\delta\gamma, B_1, ..., B_n)$ is therefore equal to $\mathcal{S}_I((\gamma \ominus \delta^f) \oplus \delta, B_1, ..., B_n)$ and can be rewritten as:

$$\bigwedge_{a \in \gamma \ominus \delta^f} \big(\sum_{l \in a^\bullet} x_l - \sum_{l \in {}^\bullet a} x_l - y_a = 0 \big) \ \wedge \ \bigwedge_{a \in \delta} \big(\sum_{l \in a^\bullet} x_l - \sum_{l \in {}^\bullet a} x_l - y_a = 0 \big)$$

The first conjunction term can be obtained by applying existential quantification of unknowns $(y_b)_{b \in \gamma \cap \delta^f}$ on the conjunction over the set of interactions γ:

$$\bigwedge_{a \in \gamma \ominus \delta^f} \big(\sum_{l \in a^\bullet} x_l - \sum_{l \in {}^\bullet a} x_l - y_a = 0 \big) \equiv (\exists y_b)_{b \in \gamma \cap \delta^f} \Big(\bigwedge_{a \in \gamma} \big(\sum_{l \in a^\bullet} x_l - \sum_{l \in {}^\bullet a} x_l - y_a = 0 \big) \Big)$$

The existential quantification can be safely extended over both conjunction terms, as quantified unknowns do not occur (yet) in the second term. But now, regarding this second term, any interaction a of the increment δ can be written as a disjoint union $a = \sqcup_k b_k$ where interactions $b_k \in \gamma$, for all k. It follows that ${}^\bullet a = \sqcup_k {}^\bullet b_k$, $a^\bullet = \sqcup_k b_k^\bullet$ hence, we can rewrite for any $a \in \delta$ the sums

$$\sum_{l \in a^\bullet} x_l - \sum_{l \in {}^\bullet a} x_l = \sum_{b_k \in \gamma, \sqcup_k b_k = a} \Big(\sum_{l \in b_k^\bullet} x_l - \sum_{l \in {}^\bullet b_k} x_l \Big) = \sum_{b_k \in \gamma, \sqcup_k b_k = a} y_{b_k}$$

The above facts can be used together and prove the result. □

A direct consequence of the above theorem is that linear invariants are preserved by layering. That is, any invariant-generating solution $\mathbf{u}$ of $\mathcal{S}_I(\gamma, B_1, ..., B_n)$ can be *extended* to an invariant-generating solution $\mathbf{u}'$ of $\mathcal{S}_I(\delta\gamma, B_1, ..., B_n)$ such that $\mathbf{u}_{|L} = \mathbf{u}'_{|L}$. In fact, ones can easily check that, whenever y_a unknowns are set to 0, incremental systems put less constraints on x_l unknowns after layering than before, and hence, invariant-generating solutions are preserved. Consequently, linear invariants discovered at any step of the composition are preserved through layering operations.

Finally, the incremental system can also be split on disjoint union of components, as stated by the following proposition.

Proposition 1 (Disjoint Union) *Let $B_1 = \gamma_1(\{B_i\}_{i \in I_1})$, $B_2 = \gamma_2(\{B_i\}_{i \in I_2})$ be disjoint composite components, that is, $I_1 \cap I_2 = \emptyset$. Then, it holds:*

$$\mathcal{S}_I(\gamma_1 \oplus \gamma_2, \{B_i\}_{i \in I_1 \cup I_2}) \approx \mathcal{S}_I(\gamma_1, \{B_i\}_{i \in I_1}) \bigwedge \mathcal{S}_I(\gamma_2, \{B_i\}_{i \in I_2})$$

PROOF. Using Definition 9, we obtain the characteristic system of the component $(\gamma_1 \oplus \gamma_2)(\{B_i\}_{i \in I_1 \cup I_2})$. In this system, we split the main conjunction in the system by unfolding independently the two connectors γ_1 and γ_2:

$$\mathcal{S}_I(\gamma_1 \oplus \gamma_2, \{B_i\}_{i \in I_1 \cup I_2}) \approx$$
$$\bigwedge_{a \in \gamma_1} \Big(\sum_{l \in a^\bullet} x_l - \sum_{l \in {}^\bullet a} x_l - y_a = 0 \Big) \wedge \bigwedge_{a \in \gamma_2} \Big(\sum_{l \in a^\bullet} x_l - \sum_{l \in {}^\bullet a} x_l - y_a = 0 \Big)$$

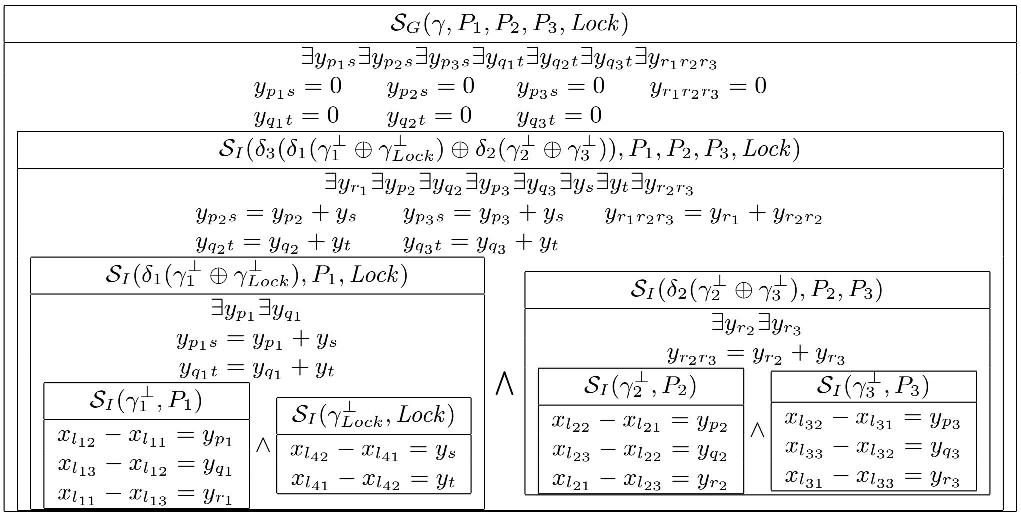
Using Definition 9, we rewrite each subterm to obtain the equivalence $\mathcal{S}_I(\gamma_1 \oplus \gamma_2, \{B_i\}_{i \in I_1 \cup I_2}) \approx \mathcal{S}_I(\gamma_1, \{B_i\}_{i \in I_1 \cup I_2}) \wedge \mathcal{S}_I(\gamma_2, \{B_i\}_{i \in I_1 \cup I_2})$. The interactions in γ_1 are only defined over the component set $\{B_i\}_{i \in I_1}$. For any interaction $a \in \gamma_1$, each unknown x_l in the sets ${}^\bullet a$ or $a^\bullet$ corresponds to the location l. This location belongs to a component of $\{B_i\}_{i \in I_1}$ that is separated from $\{B_i\}_{i \in I_1}$: the characteristic systems $\mathcal{S}_I(\gamma_1, \{B_i\}_{i \in I_1 \cup I_2})$ and $\mathcal{S}_I(\gamma_1, \{B_i\}_{i \in I_1})$ are equivalent. We similarly deduce that $\mathcal{S}_I(\gamma_2, \{B_i\}_{i \in I_1 \cup I_2}) \approx \mathcal{S}_I(\gamma_2, \{B_i\}_{i \in I_2})$. After rewritting terms using equivalence relation, the conclusion is immediate. □

This proposition allows to infer that invariant-generating solutions are preserved by disjoint union, and consequently, any linear invariant discovered locally for $\gamma_1(\{B_i\}_{i \in I_1})$ and $\gamma_2(\{B_i\}_{i \in I_2})$ is also an invariant for the composite $(\gamma_1 \oplus \gamma_2)(\{B_i\}_{i \in I_1 \cup I_2})$.

Example 8 *Following the incremental composition used for the example illustrated in Figure 2, the incremental characteristic systems constructed at different steps of the design are given in the table below. For each increment (a subdivision of the table) we discover some linear invariants. The computation steps associated to the increments δ_1 and δ_2 gives an invariant $at\_l_{i1} + at\_l_{i2} + at\_l_{i3} = 1$ for each component P_i and the invariant $at\_l_{41} + at\_l_{42} = 1$ for the component Lock. The next step corresponds to the disjoint union: we merge the two characteristic systems, and we collect the invariants obtained form each one. For the last increment δ_3, we obtain the invariant $at\_l_{12} + at\_l_{22} + at\_l_{32} + at\_l_{41} = 1$. This invariant ensures the mutual exclusion property in the system. When Lock is activated $at\_l_{42} = 1$ and hence $at\_l_{41} = 0$, the invariant ensures that exactly one of the P_i reached its location at_l_{i2}.*

BIP Model				Matrix $\mathbf{A}$	
Name	$\lvert\gamma\rvert$	$\lvert L\rvert$	$avg(\gamma)$	Matrix Size	Fill factor
`Voting Srv`	18	29	2	522	17%
`Philo(`n`)`	$5n$	$6n$	2.2	$30n^2$	$4/(5n)$
`Smokers(`n`)`	$12n$	$9n$	2.25	$108n^2$	$1/(2n)$
`ReadWrite(`n`)`	$33n$	$23n$	2	$759n^2$	$1/(16n)$
`ATM(`n`)`	$39n$	$36n$	0.6	$1404n^2$	$1/(3n)$
`Gas Station(`n`)`	$40n$	$43n$	2.5	$1720n^2$	$1/(16n)$

Table 1: Matrix sparsity for the characteristic systems $\mathcal{S}_G$



$$\mathcal{S}_G(\gamma, P_1, P_2, P_3, Lock)$$

$$\exists y_{p_1 s}\exists y_{p_2 s}\exists y_{p_3 s}\exists y_{q_1 t}\exists y_{q_2 t}\exists y_{q_3 t}\exists y_{r_1 r_2 r_3}$$
$$y_{p_1 s} = 0 \quad y_{p_2 s} = 0 \quad y_{p_3 s} = 0 \quad y_{r_1 r_2 r_3} = 0$$
$$y_{q_1 t} = 0 \quad y_{q_2 t} = 0 \quad y_{q_3 t} = 0$$

$$\mathcal{S}_I(\delta_3(\delta_1(\gamma_1^\perp \oplus \gamma_{Lock}^\perp) \oplus \delta_2(\gamma_2^\perp \oplus \gamma_3^\perp)), P_1, P_2, P_3, Lock)$$

$$\exists y_{r_1}\exists y_{p_2}\exists y_{q_2}\exists y_{p_3}\exists y_{q_3}\exists y_s\exists y_t\exists y_{r_2 r_3}$$
$$y_{p_2 s} = y_{p_2} + y_s \quad y_{p_3 s} = y_{p_3} + y_s \quad y_{r_1 r_2 r_3} = y_{r_1} + y_{r_2 r_2}$$
$$y_{q_2 t} = y_{q_2} + y_t \quad y_{q_3 t} = y_{q_3} + y_t$$

$$\mathcal{S}_I(\delta_1(\gamma_1^\perp \oplus \gamma_{Lock}^\perp), P_1, Lock)$$

$$\exists y_{p_1}\exists y_{q_1}$$
$$y_{p_1 s} = y_{p_1} + y_s$$
$$y_{q_1 t} = y_{q_1} + y_t$$

$$\mathcal{S}_I(\gamma_1^\perp, P_1)$$
$$x_{l_{12}} - x_{l_{11}} = y_{p_1}$$
$$x_{l_{13}} - x_{l_{12}} = y_{q_1}$$
$$x_{l_{11}} - x_{l_{13}} = y_{r_1}$$

$$\wedge$$

$$\mathcal{S}_I(\gamma_{Lock}^\perp, Lock)$$
$$x_{l_{42}} - x_{l_{41}} = y_s$$
$$x_{l_{41}} - x_{l_{42}} = y_t$$

$$\bigwedge$$

$$\mathcal{S}_I(\delta_2(\gamma_2^\perp \oplus \gamma_3^\perp), P_2, P_3)$$

$$\exists y_{r_2}\exists y_{r_3}$$
$$y_{r_2 r_3} = y_{r_2} + y_{r_3}$$

$$\mathcal{S}_I(\gamma_2^\perp, P_2)$$
$$x_{l_{22}} - x_{l_{21}} = y_{p_2}$$
$$x_{l_{23}} - x_{l_{22}} = y_{q_2}$$
$$x_{l_{21}} - x_{l_{23}} = y_{r_2}$$

$$\wedge$$

$$\mathcal{S}_I(\gamma_3^\perp, P_3)$$
$$x_{l_{32}} - x_{l_{31}} = y_{p_3}$$
$$x_{l_{33}} - x_{l_{32}} = y_{q_3}$$
$$x_{l_{31}} - x_{l_{33}} = y_{r_3}$$

6 Implementation, Experiments and Results

We split the section in two parts. First we show the power of algorithm 1; second we demonstrate the efficiency of our incremental approach.

6.1 On Algorithm 1

As we have seen in previous sections, linear invariant generation relies on methods to compute the set of solutions of a given homogeneous system of linear equations. As we observed in Section 5, complexity of standard algorithms for solving such systems is $O(mn^2)$, for systems of size $m \times n$. Most of classical algorithms such as Gauss-Jordan elimination may reach this complexity. This is especially the case when considering *dense* systems. However, in the context of our work, we observed that characteristic systems are usually *sparse*. The reason is that interactions synchronize few components, and therefore the associated equations involve few locations. In many cases, bigger is the composition (which implies a large number of components and locations), lower is the fill factor of the characteristic system. Given a composition with $|\gamma|$ interactions of atomic components totalizing $|L|$ locations, the matrix $\mathbf{A}$ for $\mathcal{S}_G$ has size of $|\gamma| \times |L|$. If $avg(\gamma)$ denotes the average number of components used per interaction, the fill factor of $\mathbf{A}$ is $2 \cdot avg(\gamma)/|L|$. Table 1 illustrates the fill factor for some common BIP examples. This particular structure is exploited by the global online Algorithm 1.

6.2 On Computing Linear Invariants

DFINDER. We have implemented the techniques proposed in this paper as an extension of DFINDER, a tool capable of checking deadlocks of programm written in the BIP language. DFINDER originally implements efficient symbolic techniques for computing Boolean invariants ψ of the interactions between components [6]. As shown in Figure 3, ψ can then be combined with the invariant ϕ_i of each constituent component to deduce a global invariant for the complete system (see [17] for a proof). At the same time, the tool also computes all the potential deadlock states denoted by DIS. If the formula $\wedge_i \phi_i \wedge \Psi \wedge DIS$ is unsatifiable, then the system is deadlock free. In the other case, the solutions denote some suspicious counter examples that can be reused by the tool to refine automatically the analysis. For the purpose of this work, we have implemented new techniques based on linear invariants in order to compute ψ.

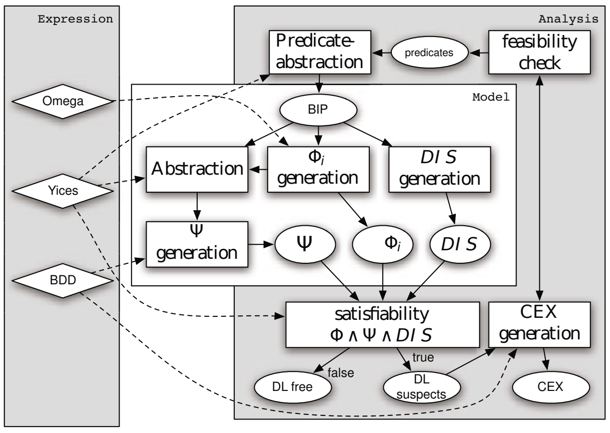
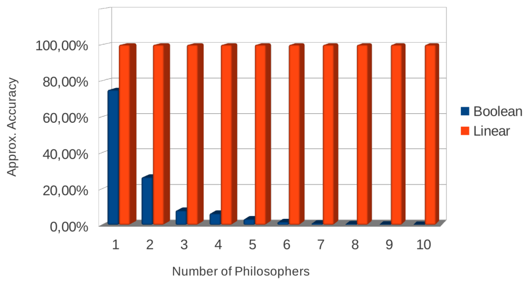


Figure 3: Structure of the D-Finder tool

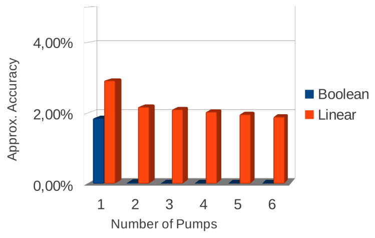
Experiments. Table 2 represents a set of experiments. All of the experiments have been conducted with incremental approach as we observed that it clearly outperforms the global one. All our experiments were done with a 2.4GHz Core 2 Duo CPU with 8GB of RAM (a laptop running Mac OS X 10.6). We generated linear invariants for various case studies, including the Gas Station [18], a derived version of the Smoker [19], the Automatic Teller Machine (ATM) [20] and the classical Dinning Philosopher problem. 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). Each example is parametrized by *s*cale, which denotes its "size"; *l*ocation denotes the total number of control locations $|L|$; *i*nteraction is for the total number of interactions $|\gamma|$. The computation time is given in minutes and the memory usage is given in kilo- or MegaBytes. The timeout, i.e., "-" is one hour. We implemented the methods described in Section 5 within DFINDER. Alternatively we implemented GAUSS a standard Gauss-Jordan elimination and we used CHARLIE, a general Petri-net analyzer [9]. We observe that the approach based on Algorithm 1 is always faster, and the consumed memory by DFINDER is negligible compared to the other approaches. We also observe that CHARLIE fails to analyze the Petri-nets generated from the BIP models. It generates a particular set of invariants so-called semi-linear positive invariants that require an important complexity. They allow to check several kinds of properties (structural, coverability, reachability, ...), but for the reachability analysis they are however equivalent to the linear invariants.

Preciseness. We also observe that our technique generates invariants that are coarser than Booleans ones, which decreases the risk of introducing counter examples. Figures 4(a), 4(b) and 4(c) give the accuracy of the generated invariants (for both the Boolean and the linear one) for the Dining Philosopers, the Gas Station and the ATM, with for each system with different sizes. On these figures, the value 60% means that the reachable states of the system are 60% of all the states characterized by the invariants. It dualy means that these same invariants catch 40% of unreachable states. Notice that an accuracy of 0% (*i.e.* no reachable state contained) is never reached since the generated invariants are sound and always denote all the reachable states. But for some of the Boolean invariants, the approximation is so imprecise that the result is really close to 0% in the figures.

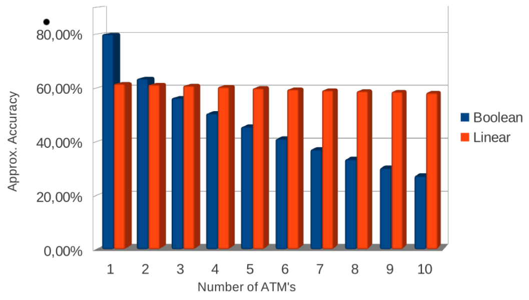
The above examples have differents types of interactions between the consituent componenents, and this has an impact on the preciseness. In the Dining Philosophers, one can see that all the interactions



(a) Dining Philosophers



(b) Gas Station



(c) ATM

Figure 4: Preciseness for some examples

are there in order to introduce mutual exclusion mecanisms. As explained below, the linear invariants are really adequate to express such properties as they can be encoded by linear equations. For such an example, the result is of clear interest. Indeed, the generated linear invariants exactly denotes the set of reachable states. For the same reason, we also obtain an excellent precision (90%) with the linear invariants for Readers/Writers example.

On the contrary, the approximation for the Gas Station example is coarser. Indeed, the relation between the consumers and the pumps is quite well-suited (e.g. ressemble a mutual exclusion principle), but the overall behavior of the station is guaranteed by an operator that relies on global self-loops. Such interactions are more expressive than linear equation. This means that they can only be approximated by such equations. Additionally, each new pump added to the system is connected to the operator with interactions over the self-loops that deteriorates the precision of the approximation.

In Figure 4(c), the ATM example contains also some interactions defined over self-loops. But there are used to define some timers in some of the compenents. As such, they do not define strong synchronisations between the compenents. This means that there impact is smaller than for the Gas Station. Consequently, this justifies that for each ATM added in the system, the 60% of accuracy does not decrease so much.

Globally, we clearly observe that the linear invariants drastically increase the accuracy of the verification compared to the boolean invariants. But as explained in [6], Boolean invariants are sufficient to prove the deadlock freeness of a system. Moreover, if the linear invariants are more accurate than the boolean invariants, the approximated states of the linear invariants are not always a subset of those of the boolean invariants: the conjunction of the linear and boolean invariants increase the precision of the analysis for the cases with self-loops like in the Gas Station example.

Component information			Time (m′ ss)			Memory (Bytes)		
scale	locations	interactions	CHARLIE	GAUSS	DFINDER	CHARLIE	GAUSS	DFINDER
DINING PHILOSOPHERS								
500 philos	3000	2500	8’40	0’03	>0’01	143M	120M	0.9M
1000 philos	6000	5000	74’42	0’13	0’01	468M	596M	1.0M
2000 philos	12000	10000	-	0’73	0’04	-	2.4G	1.2M
6000 philos	36000	30000	-	-	1’40	-	-	1.8M
9000 philos	54000	45000	-	-	9’15	-	-	2.0M
ATM								
50 machines	1812	1656	-	0’14	>0’01	-	73M	1.6M
100 machines	3612	3306	-	1’38	0’01	-	238M	2.8M
200 machines	7212	6606	-	12’41	0’03	-	940M	4.0M
400 machines	14412	13206	-	-	0’13	-	3.6G	6.4M
500 machines	18012	16506	-	-	0’31	-	-	7.2M
GAS STATION								
50 pumps	2152	2000	-	1’17	0’01	-	69M	2.5M
100 pumps	4302	4000	-	14’58	0’04	-	271M	3.3M
200 pumps	8602	8000	-	-	0’14	-	-	4.7M
500 pumps	21502	20000	-	-	2’30	-	-	8.7M
700 pumps	30102	28000	-	-	3’40	-	-	11.4M
READERS - WRITERS								
50 writers	1152	1650	3’15	1’06	>0’01	150M	54M	2.2M
100 writers	2322	3300	19’50	8’12	0’02	937M	212M	2.6M
200 writers	4642	6600	-	65’43	0’06	-	847M	3.2M
500 writers	11502	16500	-	-	0’37	-	-	5.0M
1000 writers	23002	33000	-	-	3’22	-	-	7.5M
2000 writers	46002	66000	-	-	17’40	-	-	9.7M
SMOKERS								
300 smokers	906	901	0’17	0’30	0’01	90M	14M	1.4M
600 smokers	1806	1801	1’31	3’11	0’01	229M	52M	2.3M
1500 smokers	4506	4501	-	55’00	0’06	395M	319M	3.1M
6000 smokers	18006	18001	-	-	1’51	-	-	6.8M
9000 smokers	27006	27001	-	-	4’37	-	-	9.3M

Table 2: Execution time for some examples

References

[1] J-P.Queille and J.Sifakis. Specification and verification of concurrent systems in CESAR. In *Symposium on Programming*, LNCS. Springer, 1982. 1

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

[3] S. Bensalem, M. Bozga, T.-H. Nguyen, and J. Sifakis. Compositional verification for component-based systems and application. In *ATVA*, 2008. 1, 1, 3

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

[5] S. Bensalem, M. Bozga, A. Legay, Th.H. Nguyen, J. Sifakis, and R. Yan. Incremental component-based construction and verification using invariants. In *FMCAD’10*, 2010. 1, 1

[6] S. Bensalem, A. Griesmayer, A. Legay, T.H. Nguyen, J. Sifakis, and R. Yan. D-finder 2: Towards efficient correctness of incremental design. In *NASA Formal Methods*, LNCS. Springer, 2011. 1, 6.2, 6.2

[7] BIP – incremental component-based construction of real-time systems. `http://www-verimag.imag.fr/~async/bip.php`. 1, 3

[8] S. Bensalem, L. Silva, A. Griesmayer, F. Ingrand, A. Legay, and R. Yan. A formal approach for incremental construction with an application to autonomous robotic systems. In *SC’11*, LNCS. Springer, 2011. 1

[9] University of Technology in Cottbus. Charlie: a tool to analyse Petri nets. `http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Charlie`. 1, 1, 6.2

[10] Kenneth L. McMillan. *Symbolic model checking*. Kluwer, 1993. 1

[11] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *POPL*. ACM Press, 1978. 1

[12] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Symposium on Principles of Programming Languages*. ACM Press, 1977. 1

[13] T. Murata. Petri nets: Properties, analysis and applications. *IEEE*, 1989. 1, 5.1

[14] F. Krückeberg and M. Jaxy. Mathematical methods for calculating invariants in petri nets. In *ICAPTN*, LNCS. Springer, 1986. 1

[15] S. Sankaranarayanan, H. Sipma, and Z. Manna. Petri net analysis using invariant generation. In *Verification: Theory and Practice*, LNCS. Springer, 2004. 1

[16] Gregor Gößler and Joseph Sifakis. Priority systems. In *FMCO*, pages 314–329, 2003. 3

[17] S. Bensalem, A. Legay, T.H. Nguyen, J. Sifakis, and R. Yan. Incremental invariant generation for compositional design. In *TASE*, 2010. 3, 3.2, 6.2

[18] D. Heimbold and D. Luckham. Debugging Ada tasking programs. *IEEE*, 1985. 6.2

[19] S. S. Patil. *Limitations and Capabilities of Dijkstra's Semaphore Primitives for Coordination among Processes*. Computation Structures Group Memo 57, 1971. 6.2

[20] M.R.V. Chaudron, E.M. Eskenazi, A.V. Fioukov, and D.K. Hammer. A framework for formal component-based software architecting. In *OOPSLA*, 2001. 6.2