Verification of Component-Based Systems via Predicate Abstraction and Simultaneous Set Reduction

Wang Qiang(✉) and Simon Bliudze

École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland
qiang.wang@epfl.ch

Abstract. This paper presents a novel safety property verification approach for component-based systems modelled in BIP (Behaviour, Interaction and Priority), encompassing multiparty synchronisation with data transfer and priority. Our contributions consist of: (1) an on-the-fly lazy predicate abstraction technique for BIP; (2) a novel explicit state reduction technique, called simultaneous set reduction, that can be combined with lazy predicate abstraction to prune the search space of abstract reachability analysis; (3) a prototype tool implementing all the proposed techniques. We also conduct thorough experimental evaluation, which demonstrates the effectiveness of our proposed approach.

1 Introduction

BIP [2] is a component-based rigorous system design framework, that advocates the methodology of correctness-by-construction. Rigorous system design can be understood as a formal, accountable and coherent process for deriving trustworthy implementations from high-level system models, which aims at guaranteeing the essential properties of a design at the earliest possible design phase, and then automatically generating correct implementations by a sequence of property preserving model transformations progressively refining the models with details specific to the target platforms [22].

BIP supports the rigorous design flow with the well-defined BIP modelling language and an associated tool-set. To model complex systems, the BIP language advocates the principle of separation of concerns (i.e. computation and coordination), and provides a three-layered mechanism for this purpose, i.e. Behaviour, Interaction, and Priority. Behaviour is characterised by a set of atomic components, defined as automata extended with linear arithmetic. Interaction represents the multiparty synchronisation of atomic components, among which data transfer may take place. Priority can be used to schedule the interactions or resolve conflicts when several interactions are enabled simultaneously.

This work was carried out within the D-MILS project, which is partially funded under the European Commission's Seventh Framework Programme (FP7).

P. Ganty and M. Loreti (Eds.): TGC 2015, LNCS 9533, pp. 147–162, 2016.
DOI: 10.1007/978-3-319-28766-9_10

In the BIP framework, DFinder [4] is the dedicated tool for automatic invariant generation and safety properties verification. DFinder computes an invariant in a compositional manner: it first computes a component invariant for each component over-approximating its behaviour and then computes the interaction invariant characterising the coordination constraint of all components. The invariant of the global system is then the conjunction of component invariants and the interaction invariant. However, DFinder does not handle system models with data transfer. This limitation hampers the practical application of DFinder and of the BIP framework, since data transfer is necessary and common in the design of real-life systems. Besides, it is not clear in DFinder how to refine the abstraction automatically when the inferred invariant fails to justify the property.

Some other works on automatic verification of BIP models exist, but they all suffer from certain limitations. The VCS [14] tool translates a BIP model into a symbolic transition system and then performs the bounded model checking. It handles data transfer among components, but only deals with finite domain variables. In [23], a timed BIP model is translated into Timed Automata and then verified with Uppaal [3]. The translation handles data transfers, but it is limited to BIP models with finite domain data variables. In [18], the authors show an encoding of a BIP model into Horn Clauses, which are verified with Eldarica [17], but they do not handle data transfers on interactions.

In [5], the authors instantiate the ESST (Explicit Scheduler Symbolic Thread) framework [8] for BIP, where a dedicated BIP scheduler is developed to orchestrate the abstract reachability analysis, and partial order reduction techniques [11] are applied to further boost the analysis. Although being closely related, our approach is tailored for BIP and leverages its operational semantics to define the necessary minimal notion of abstract state, as opposed to that of ESST, where additional component status information and primitive functions have to be stored to account for the BIP scheduler.

Our approach is inspired by the idea of separation of computation and coordination, advocated by BIP three-layered modelling mechanism. In brief, we propose to decompose the verification of component-based systems into two levels by taking advantage of the structure features of such systems. Thus, we handle the computation of components and the coordination among components separately. On the computation level, we exploit the state-of-the-art counterexample guided abstraction refinement technique (e.g. lazy abstraction [15,16]) to analyse the behaviour of components; while on the coordination level, we deal with the redundant interleavings by a novel explicit state reduction technique, called simultaneous set reduction. The basic idea is that when two concurrent actions are enabled at the same time, instead of taking into account all the possible interleavings, we may consider executing them simultaneously. To this end, we make the following contributions in this paper: (1) we propose an on-the-fly lazy predicate abstraction technique for the verification of BIP models; (2) we propose a novel explicit state reduction technique (i.e. simultaneous set reduction) to reduce the search space when performing the abstract reachability analysis; (3) we have implemented the proposed techniques in our prototype tool and conducted thorough experimental evaluation, which shows the proposed techniques are promising for verifying generic BIP models.

2 BIP Framework

In this section we introduce the syntax and semantics of a subset of the BIP language, which encompasses the multiparty synchronisation and data transfer.

2.1 BIP Modelling Language

We use symbol Var to denote a finite set of variables with both finite and infinite domains. A guard (or predicate) is a boolean expression over Var. An operation is either an assignment or a sequence of assignments of the form $x := exp$, where $x \in Var$ and exp is an expression in linear arithmetic over Var. We denote by $Guard$ and Op the set of guards and operations over Var respectively, and Op includes a special operation $skip$, which has no effect on variables in Var. Symbols can be indexed to refer to a specific component.

Definition 1 (Atomic Component). *An atomic component is a tuple* $B_i = (Var_i, Loc_i, Port_i, Trans_i, l_{0_i})$*, where:*

1. Var_i *is a finite set of variables;*
2. Loc_i *is a finite set of control locations;*
3. $Port_i$ *is a finite set of ports, which are labels on the transitions;*
4. $Trans_i \subseteq Loc_i \times Guard_i \times Port_i \times Op_i \times Loc_i$ *is a set of transitions with guards and operations over* Var_i*.*
5. $l_{0_i} \in Loc_i$ *is the initial control location.*

The values of atomic component variables can be transfered to other components upon interaction (see Definition 2 below). However, they cannot be modified by the receiving components.

Transitions are labelled by ports, which form the interface of atomic components, and are used for defining the interactions. A port is enabled iff the transition labelled by this port is enabled.

Given a set of atomic components $\{B_i\}_{i=1}^n$, we denote $Port = \bigcup_{i=1}^n Port_i$ the set of all the ports and $Var = \bigcup_{i=1}^n Var_i$ the set of all the variables belonging to the components $\{B_i\}_{i=1}^n$. Notice that we assume that all $Port_i$

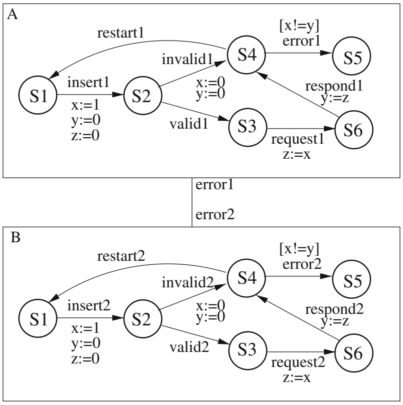

Fig. 1. An example BIP model

and all Var_i, for $i = 1, ..., n$, are pairwise disjoint. Thus, in particular, the scope of a variable can be considered to be the component, to which it belongs. The model in Fig. 1 has six variables: x, y and z in each of the two components A and B.

Composition of a set of atomic components is then specified by a set of interactions.

Definition 2 (Interaction). *An interaction γ is a tuple (g, u, op), where $u \subseteq Port$ such that $|u \cap Port_i| \leq 1$, $\forall i \in [1, n]$, and $g \in Guard$, $op \in Op$.*

Intuitively, an interaction γ specifies a guarded synchronisation among the participating components: the synchronisation and the corresponding operation (i.e. data transfer) op can take place only when the guard g is satisfied, and all the ports in u are enabled. When an interaction is taken, the transitions labelled by these ports are taken synchronously, i.e. the execution of all the operations associated to the interaction and the involved transitions constitutes a single atomic operation. When several interactions are enabled at the same time, priority can be used to schedule the ones to be executed.

Definition 3 (Priority Model). *Given a set of interactions Γ, a priority model π is a strict partial order on Γ. For $\gamma, \gamma' \in \Gamma$, we write $\gamma < \gamma'$ if and only if $(\gamma, \gamma') \in \pi$, which means that interaction γ' has a higher priority than γ.*

Given a set of atomic components $\{B_i\}_{i=1}^n$, a set of interactions $\Gamma = \{\gamma_i\}_{i=1}^m$, and a priority model π, we denote by $\Gamma_\pi(B_1, ..., B_n)$ the system model constructed by composing atomic components with Γ and π.

Example 1. To give an intuitive understanding of the BIP modelling language, we show a simple BIP model with two components A and B in Fig. 1. Each component has three integer variables and may enter a deadlock state $S5$ by taking transition $error1$ or $error2$ when the guard $x \neq y$ is true. There is one binary interaction $\gamma = (true, \{error1, error2\}, skip)$ synchronising the two transitions labelled by ports $error1$ and $error2$, and all the other transitions form singleton interactions (e.g. $(true, \{invalid1\}, x := 0; y := 0)$). No data transfer or priority is defined in this model.

2.2 Operational Semantics of BIP

To define the operational semantics of BIP, we first introduce the notion of configuration.

Definition 4 (Configuration of a BIP Model). *Given a BIP model $\Gamma_\pi(B_1, ..., B_n)$, a configuration is a tuple $c \triangleq \big((l_1, \boldsymbol{x}_1), ..., (l_n, \boldsymbol{x}_n)\big)$, where each l_i is a control location of component B_i, and $\boldsymbol{x}_i$ is a valuation of variables in Var_i of B_i.*

An interaction $\gamma = (g, u, op)$ is enabled in a configuration $c = \big((l_1, \mathbf{x}_1), ..., (l_n, \mathbf{x}_n)\big)$, if the following two conditions are satisfied: (1) the guard g is satisfied by $(\mathbf{x}_i)_{i=1}^n$; and (2) for each component B_i such that $u \cap Port_i = \{p_i\}$, there is a transition $(l_i, g_i, p_i, op_i, l_i') \in Trans_i$ starting from l_i and labelled by p_i, such that guard g_i is satisfied by $\mathbf{x}_i$.

Definition 5 (Operational Semantics of BIP). *Given a BIP model $\Gamma_\pi(B_1, ..., B_n)$, there is a transition from $c = \big((l_1, \boldsymbol{x}_1), ..., (l_n, \boldsymbol{x}_n)\big)$ to $c' = \big((l'_1, \boldsymbol{x}'_1), ..., (l'_n, \boldsymbol{x}'_n)\big)$ if there is an interaction $\gamma = (g, u, op)$, such that*

1. *γ is enabled in c;*
2. *for each component B_i such that $u \cap Port_i = \{p_i\}$, there is a transition $(l_i, g_i, p_i, op_i, l'_i) \in Trans_i$ and $\boldsymbol{x}'_i = op_i\big(op(\boldsymbol{x}_i)\big)$;*
3. *for each component B_j such that $u \cap Port_j = \emptyset$, we have $(l'_j, \boldsymbol{x}'_j) = (l_j, \boldsymbol{x}_j)$;*
4. *there does not exist an interaction γ', such that γ' is enabled in c and $\gamma' > \gamma$.*

Whenever there is a transition from configuration c to c', we use the notation $c \xrightarrow{\gamma} c'$ to indicate that this transition is triggered by the interaction γ. Notation $op(\mathbf{x})$ denotes the application of operation op to the expression $\mathbf{x}$. When op is an assignment of form $x := exp$, its semantics can be given by substitution $\mathbf{x}[exp/x]$ denoting the valuation of variables, where the valuation of x is substituted by exp.

We say that configuration $c_0 = ((l_1, \mathbf{x}_1), \ldots, (l_n, \mathbf{x}_n))$ is an *initial configuration* if $l_i = l_{0_i}$, for all $1 \leq i \leq n$. A trace is then a sequence of transitions $c_0 \xrightarrow{\gamma_1} c_1 \xrightarrow{\gamma_2} \cdots \xrightarrow{\gamma_k} c_k$. A configuration c is *reachable* if and only if there exists a trace that starts from the initial configuration and ends in c.

To encode a safety property, we identify a set of *error locations* (which are also deadlock locations, e.g. location $S5$ in Fig. 1), such that a BIP model is safe if and only if no error locations are reachable. Notice that every safety property verification problem can be encoded into a reachability problem with additional transitions, interactions and error locations in the BIP model.

3 On-the-fly Lazy Predicate Abstraction of BIP

In this section, we present our key verification algorithm for BIP which is based on lazy abstraction [15,16] and features an on-the-fly exploration of the abstract reachable states.

3.1 Verification Algorithm

The main function of our verification algorithm is shown in Algorithm 1. The algorithm takes a BIP model with the encoding of safety property as input, and explores its reachable state space by constructing an abstract reachability tree (ART). The verification procedure is sound and complete: the lazy abstraction approach consists in verifying the most abstract model sufficient to establish a definite result (safe or unsafe). Abstraction is refined every time a spurious counterexample is found.

Our algorithm constructs the ART by expanding the ART nodes progressively, starting from the initial one. Whenever an error node is encountered, it generates a counterexample (line 8) and checks if the counterexample is real (line 9). If the counterexample is real, the algorithm stops and reports the model is unsafe and a counterexample is found (line 10). Otherwise, the algorithm will refine the abstraction and restart the exploration (line 12). An ART node is expanded when it cannot be covered by another one and all its children will be pushed into the worklist (lines 16 and 17). When a node is covered, the algorithm stops the expansion from this node by marking it as covered (line 14).

Algorithm 1. Main function

```
Input: a BIP model $B = \Gamma_\pi(B_1, ..., B_n)$ with encoding of safety property
Output: Either $B$ is safe, or $B$ is unsafe with a counterexample $cex$
 1: create an ART node $node_0$ from the initial state
 2: create an ART $art$ with $node_0$ being the root
 3: create a worklist $wl$ of ART nodes
 4: push $node_0$ into $wl$
 5: while $wl \neq \emptyset$ do
 6:     $node \leftarrow$ pop($wl$)
 7:     if $node$ is an error node then
 8:         $cex \leftarrow$ CounterExample($node$)
 9:         if $cex$ is real then
10:             return $B$ is unsafe with a real counterexample $cex$
11:         else
12:             Refine($art$, $cex$)
13:     else if $node$ is covered then
14:         mark $node$ as covered
15:     else
16:         Expand($node$)
17:         push all children of $node$ into $wl$
18: return $B$ is safe
```

Definition 6 (ART Node). *Given a BIP model $B = \Gamma_\pi(B_1, ..., B_n)$, an ART node is a tuple $\big((l_1, \phi_1), ..., (l_n, \phi_n), \phi\big)$, where (l_i, ϕ_i) is the local region consisting of the control location l_i and the abstract data region ϕ_i of component B_i, and ϕ is the global data region.*

A data region is a formula that over-approximates the concrete valuations of variables. We maintain a global data region ϕ to keep track of all the variables that are used in data transfer. An ART node is an error node if at least one of the control location l_i is an error location and the data regions are consistent, i.e. $\phi \wedge \bigwedge_{i=1}^{n} \phi_i$ is satisfiable.

Definition 7 (Node Covering). *An ART node $\big((l_1, \phi_1), ..., (l_n, \phi_n), \phi\big)$ is covered by another node $\big((l'_1, \phi'_1), ..., (l'_n, \phi'_n), \phi'\big)$ if $l_i = l'_i$ and the implication $\phi_i \Rightarrow \phi'_i$ is valid for all $i \in [1, n]$, and $\phi \Rightarrow \phi'$ is valid.*

We say that an ART is safe when all the nodes are either fully expanded or covered, and there are no error nodes.

Node Expansion. The node expansion procedure is shown in Algorithm 2. The procedure first computes the set of enabled interactions on this node (function EnabledInteraction in line 2). We say that an interaction $\gamma = (u, g, op)$ is *enabled* on an ART node $\big((l_1, \phi_1), ..., (l_n, \phi_n), \phi\big)$ if for each component B_i such that $u \cap Port_i = \{p_i\}$, there is a transition $(l_i, g_i, p_i, op_i, l'_i) \in Trans_i$ starting from l_i and labelled by p_i. Notice that the interaction enabledness on an ART node is different from the one on a BIP configuration. We do not check the satisfiability of the guards on the ART node, since we are doing lazy abstraction: if an interaction is disabled on the ART node, the successor node will be inconsistent.

For each enabled interaction γ, the procedure creates a new successor ART node with dummy elements, which will be updated accordingly (line 4). To update the abstract data region of B_i, that participates in γ (line 7), the procedure calls ExtractTransition($Trans_i, l_i, p_i$) in line 8 to extract the participating transition starting from l_i and labelled by port p_i from the set of transitions $Trans_i$, and then builds a sequential composition (denoted by symbol $\bullet$) of the guard and operation of this transition (line 11). The new abstract data region ϕ'_i is then obtained by applying the abstract strongest post-condition $SP^{\pi_{l'_i}}_{\hat{op}_i}(\phi_i)$ to the previous data region ϕ_i (line 12). Our algorithm maintains precisions for both control location (e.g. l'_i) and global region, denoted by $\pi_{l'_i}$ and π respectively. A precision is a set of predicates, over which the predicate abstraction is performed. We refer to [16] for more details. For other components, which do not participate in this interaction, their local regions and control locations will stay the same (line 15 and 16).

To update the global region, we need to consider all the participating transitions, since they may also modify component variables. For this purpose, the procedure creates two temporary variables g' and op' (line 5). Variable g' is the conjunction of interaction guard and all the participating transition guards (line 9), and op' is the sequential composition of the data transfer and all the participating transitions (line 10). Notice that, since the operations associated to the transitions modify only variables local to the respective components, the order of composition is irrelevant. The new global region ϕ is then updated by applying the abstract strongest post-condition $SP^{\pi}_{\hat{op}}(\phi)$ to the previous global region ϕ (line 18), where $\hat{op}$ is the guarded operation composed of g' and op'. If all abstract strongest post-condition computations succeed, the new ART node is inserted as the child of *node* and the edge is labelled by interaction γ (function AddChild in line 21). Otherwise, this new successor node does not represent any concrete reachable configurations, thus will be ignored.

Counterexample Analysis and Abstraction Refinement. If an error node is encountered during the exploration of abstract state space, we check if this error is reachable or not in the concrete state space in two steps. First, our algorithm constructs a counterexample by backtracking the ART from the error node to the root (function CounterExample in Algorithm 1). In BIP, we denote a counterexample *cex* by a sequence of interactions, labelling the path from the root to the error node. Then, our algorithm builds a sequential execution tr_{cex}

Algorithm 2. Node expansion procedure

```
 1: procedure EXPAND($node = ((l_1, \phi_1), ..., (l_n, \phi_n), \phi)$ )
 2:     $interactions \leftarrow$ EnabledInteraction($node$)
 3:     for $\gamma = (g, u, op) \in interactions$ do
 4:         $node' \leftarrow ((l''_1, \phi'_1), ..., (l''_n, \phi'_n), \phi')$
 5:         $g' \leftarrow g$; $op' \leftarrow op$
 6:         for $B_i \in B = \Gamma_\pi(B_1, ..., B_n)$ do
 7:             if $Port_i \cap u = \{p_i\}$ then
 8:                 $(l_i, g_i, p_i, op_i, l'_i) \leftarrow$ ExtractTransition($Trans_i$, $l_i$, $p_i$)
 9:                 $g' \leftarrow g' \wedge g_i$
10:                 $op' \leftarrow op' \bullet op_i$
11:                 $\hat{op}_i \leftarrow g_i \bullet op_i$
12:                 $\phi'_i = SP^{\pi_{l'_i}}_{\hat{op}_i}(\phi_i)$; $l''_i = l'_i$
13:                 if $\phi'_i$ is false then
14:                     goto 3
15:             else if $Port_i \cap u = \emptyset$ then
16:                 $l''_i = l_i$; $\phi'_i = \phi_i$
17:         $\hat{op} \leftarrow g' \bullet op'$
18:         $\phi' = SP^{\pi}_{\hat{op}}(\phi)$
19:         if $\phi'$ is false then
20:             goto 3
21:         AddChild($\gamma$, $node'$)
```

of the counterexample *cex*, such that the counterexample *cex* is real if and only if $SP_{tr_{cex}}(true)$ is satisfiable.

Formally, given a counterexample $cex = \gamma_1 \gamma_2 \ldots \gamma_k$, where for each $i \in [1, k]$, interaction $\gamma_i = (u_i, g_i, op_i)$, $u_i = \{p^i_1, \ldots, p^i_t\}$, our algorithm constructs a sequence tr_{γ_i} of transitions $g_i \bullet op_i \bullet op^i_{j_1} \bullet ... \bullet op^i_{j_t}$, where the sequence of indices $j_1, \ldots, j_t$ is an arbitrary permutation of $\{1, \ldots, t\}$, and $op^i_{j_1}$ is the operation of transition labelled by port $p^i_{j_1}$. Then the sequential execution of counterexample *cex* is the sequential composition of all tr_{γ_i}, i.e. $tr_{cex} = tr_{\gamma_1} \bullet ... \bullet tr_{\gamma_k}$.

If the analysis reveals that the encountered error location is unreachable in the concrete state space, the precisions of the abstract analysis must be refined to eliminate the spurious counterexample by adding new predicates (function Refine in Algorithm 1). Our algorithm discovers new predicates from the interpolants of trace formula of tr_{cex}. If a predicate involves only variables that are not used in the data transfer, it is added to the precisions associated to the corresponding control locations. A predicate involving variables that are used in the data transfer is added to the global precision.

Once the precisions are refined, our algorithm will remove the sub-tree that contains the spurious counterexample, and then restart the expansion using the refined precisions. We refer to [15] for more details and the correctness of this abstraction refinement approach.

3.2 Correctness Proof

To prove the correctness of Algorithm 1, we need to relate the construction of ART with BIP operational semantics. We first show that the node expansion procedure creates successor nodes that cover (or over-approximate) the corresponding reachable configurations.

Let $B = \Gamma_\pi(B_1, ..., B_n)$ be a BIP model, and $c = \big((l_1, \mathbf{x}_1), ..., (l_n, \mathbf{x}_n)\big)$ be a configuration of B. Let $node = \big((l'_1, \phi_1), ..., (l'_n, \phi_n), \phi\big)$ be an ART node. We say that configuration c satisfies ART node $node$ (or $node$ covers c), denoted by $c \models node$, if and only if, for all $i \in [1, n]$, we have $l_i = l'_i$ and $\mathbf{x}_i \models \phi_i$, and $(\mathbf{x}_i)_{i=1}^n \models \phi$.

Lemma 1. *Let node be an ART node for a BIP model $B = \Gamma_\pi(B_1, ..., B_n)$ and $node'$ be its successor. Let c be a configuration such that $c \models node$. If $node'$ is obtained by performing interaction γ, then for any configuration c' such that $c \xrightarrow{\gamma} c'$, we have $c' \models node'$.*

Proof. Suppose $c = \big((l_1, \mathbf{x}_1), ..., (l_n, \mathbf{x}_n)\big)$, and $node = \big((l_1, \phi_1), ..., (l_n, \phi_n), \phi\big)$, where $\mathbf{x}_i \models \phi_i$, for each $i \in [1, n]$, and $(\mathbf{x}_i)_{i=1}^n \models \phi$, since $c \models node$. Suppose the successor configuration following $\gamma = (g, u, op)$ is $c' = \big((l'_1, \mathbf{x}'_1), ..., (l'_n, \mathbf{x}'_n)\big)$, and the successor node is $node' = \big((l''_1, \phi'_1), ..., (l''_n, \phi'_n), \phi'\big)$. To prove $c' \models n'$, we have to show that $l'_i = l''_i$ and $\mathbf{x}'_i \models \phi'_i$, for all $i \in [1, n]$, and $(\mathbf{x}'_i)_{i=1}^n \models \phi'$.

Consider a component B_i, such that $u \cap Port_i = \{p_i\}$, and let the corresponding transition in $Trans_i$ be $(l_i, g_i, p_i, op_i, l'_i)$. Then we have $\mathbf{x}_i \models g_i$ and $\mathbf{x}'_i = op_i(op(\mathbf{x}_i))$. According to Algorithm 2, we have $l''_i = l'_i$ and $\phi'_i = SP_{\hat{op}_i}(\phi_i)$, where $\hat{op}_i$ denotes $g_i \bullet op_i$. Based on the semantics of strongest post-condition, the fact that $\mathbf{x}_i \models \phi_i$ and $\phi_i \wedge g_i$ is satisfiable, we have $\mathbf{x}'_i \models \phi'_i$. Following a similar argument, we can prove $(\mathbf{x}'_i)_{i=1}^n \models \phi'$.

For each component B_i such that $u \cap Port_i = \emptyset$, since it does not participate the interaction, its state is unchanged. Thus, the satisfaction relation trivially holds.

Theorem 1 (Correctness of On-the-fly Lazy Predicate Abstraction of BIP). *Given a BIP model B, and for every terminating execution of Algorithm 1, we have the following properties:*

1. *if Algorithm 1 returns a real counterexample path cex, then there is a concrete execution $c \xrightarrow{cex} c'$ from an initial configuration c and an error configuration c' in B;*
2. *if Algorithm 1 returns a safe ART, then for every reachable configuration c of B, there is an ART node that covers this configuration.*

Proof. (Sketch) In the safe case, the conclusion follows from Lemma 9 and an induction proof on the execution path to the reachable configuration c. In the unsafe case, the conclusion holds because the counterexample analysis boils down to a symbolic simulation. □

4 Simultaneous Set Reduction for BIP

In this section, we present a novel reduction technique, which can be combined with on-the-fly lazy predicate abstraction to reduce the search space of reachability analysis. The idea is based on the observation that in component-based systems, when two concurrent interactions are enabled at the same time (e.g. interactions $\{insert1\}$ and $\{insert2\}$ in Fig. 1), we may consider executing them simultaneously instead of taking into account all the possible interleavings in the reachability analysis. First of all, we have to formalise the constraints imposed on the set of interactions, which can be executed simultaneously, in order to make sure no error location is missed during the reachability analysis.

4.1 Simultaneous Set Constraints

Two interactions can be executed simultaneously only when they are independent.

Definition 8 (Independent Interactions). *Two interactions γ_1 and γ_2 are independent if for every configuration c, the following conditions hold:*

1. if γ_1 is enabled in c, then γ_2 is enabled in c iff γ_2 is enabled in c', where $c \xrightarrow{\gamma_1} c'$.
2. if γ_1 and γ_2 are both enabled in c, then $c'_1 = c'_2$, where $c \xrightarrow{\gamma_1;\gamma_2} c'_1$, and $c \xrightarrow{\gamma_2;\gamma_1} c'_2$.

Since independence relation is a global property, in the sequel we will instead use the valid dependence relation.

Definition 9 (Valid Dependency Relation). *A valid dependence relation D over a set of interactions Γ is a symmetric, reflexive relation such that for every $(\gamma_1, \gamma_2) \notin D$, the interactions γ_1 and γ_2 are independent interactions.*

In BIP context, we can compute a valid dependency relation statically from the specifications: two interactions are *dependent* if they share one common component. It is worthy to notice that our independency and dependency relations also work for abstract analysis.

However, independency is not enough. For instance, in the example BIP model in Fig. 1, suppose we want to expand the node $\big((S_3, \phi_A), (S_4, \phi_B), \phi\big)$, where component A is in control location S_3 and component B is in control location S_4. The set of enabled interactions is $\{\{request1\}, \{restart2\}\}$. Notice that interaction $\{error1, error2\}$ is disabled since port $error1$ is disabled. The two interactions $\{request1\}$ and $\{restart2\}$ are independent, however, if we execute them simultaneously we will miss the following (fragment) counterexample from this node: $\{request1\}, \{respond1\}, \{error1, error2\}$. This observation tells us to take into account the future executions when firing interactions simultaneously.

Definition 10 (Simultaneous Set). *A set of interactions SSet on configuration c is called a simultaneous set if the following two constraints are satisfied:*

1. *all the interactions in SSet are independent;*
2. *for each $\alpha \in SSet$, let $c \xrightarrow{\alpha} c_1 \xrightarrow{\beta_1} ... \xrightarrow{\beta_n} c_{n+1}$ be a finite execution fragment starting with α, then for each $\alpha' \in SSet$, such that $\alpha' \neq \alpha$, all β_i are independent of α'.*

Intuitively, the second constraint means that whatever one does from the simultaneous set should still be independent from the others in the set. We remark that simultaneous set is different from the ample set [10] in that members in ample set are interdependent, and interleavings should be taken into account.

We use notation A_G to represent the full reachable state space, and A_R to represent reduced reachable state space. A transition in A_R is denoted by $c \xrightarrow{SSet(c)} c'$, where $SSet(c)$ is a simultaneous set on c. A trace in A_R is then labelled by a sequence of simultaneous sets, e.g. $c_0 \xrightarrow{SSet(c_0)} c_1 \xrightarrow{SSet(c_1)} \ldots \xrightarrow{SSet(c_{k-1})} c_k$. Similarly, we say that a configuration c is *reachable* in A_R if and only if there exists a trace that starts from the initial configuration and ends up with c. However, a trace in A_R is not a trace of A_G, but a representation of several equivalent traces.

Definition 11 (Semantics of Simultaneous Set). *Given a configuration c, a transition $c \xrightarrow{SSet(c)} c'$ in A_R denotes a set of transition sequences $\{c \xrightarrow{\gamma_1} ... \xrightarrow{\gamma_k} c' | \forall i \in [1,k], \gamma_i \in SSet(c)$ and $|SSet(c)| = k\}$ in A_G.*

Each transition sequence $c \xrightarrow{\gamma_1} ... \xrightarrow{\gamma_k} c'$ is a representation of $c \xrightarrow{SSet(c)} c'$. Inductively, we can also define the representation of a trace in A_R. Based on the definition of simultaneous set, it is easy to see that each representation of a trace in A_R is a trace in A_G.

The correctness of simultaneous set reduction for deadlock state reachability analysis is stated in the following theorem.

Theorem 2 (Correctness of Simultaneous Set Reduction). *Let e be an error configuration. If there is a trace ρ_g leading to e in A_G, then there is also a trace ρ_r leading to e in A_R.*

Proof. Assume $\rho_g = c_0 \xrightarrow{\gamma_0} \cdots \xrightarrow{\gamma_{n-2}} c_{n-1}$, where $c_{n-1} = e$. The proof proceeds by using complete induction on the number of configurations in ρ_g. For the base case $|\rho_g| = 1$, the result trivially holds since the initial configuration is also the error one. Assume the theorem holds for all the cases $|\rho_g| <= n$, where $n >= 1$, then we prove it also holds for $|\rho_g| = n + 1$.

Assume $\rho_g = c_0 \xrightarrow{\gamma_0} c_1 \xrightarrow{\gamma_1} \cdots \xrightarrow{\gamma_{n-2}} c_{n-1} \xrightarrow{\gamma_{n-1}} c_n$, where $c_n = e$, and the simultaneous set on configuration c_0 that contains interaction γ_0 is $SSet(c_0)$. If $SSet(c_0)$ is a singleton set, then ρ_r is ρ_g. If $SSet(c_0) = \{\beta_i | i \in [1,k]\} \cup \{\gamma_0\}$, according to the definition of simultaneous set, β_i is independent of γ_j, for all $i \in [1,k]$, and $j \in [1, n-1]$, then β_i should be enabled on configuration c_n, which contradicts with the fact that c_n is a deadlock state. Thus, all β_i should be executed, i.e. for each β_i there must exist a γ_j such that $\beta_i = \gamma_j$. Then by permuting independent interactions, we obtain an equivalent trace $\rho'_g = c_0 \xrightarrow{\gamma_0} c_1 \xrightarrow{\beta_1} \cdots \xrightarrow{\beta_k}\xrightarrow{\gamma_{k+1}} \cdots \xrightarrow{\gamma_{n-1}} c_n$. The sequence of interactions $\xrightarrow{\gamma_0}\xrightarrow{\beta_1} \cdots \xrightarrow{\beta_k}$ is a representation of the simultaneous set $SSet(c_0)$, while based on the induction hypothesis the rest is a representation of some trace in A_R. They all together prove our theorem. □

4.2 Combining Simultaneous Set Reduction with Lazy Predicate Abstraction

To combine the simultaneous set reduction with lazy predicate abstraction of BIP, we modify the node expansion procedure in Algorithm 2 by replacing the function EnabledInteraction in line 2 with Algorithm 3, such that instead of creating a new successor node for each possible interaction (line 3), we create a new successor node for each simultaneous set. Notice that since a simultaneous set is a set of interactions, the successor computation (the loop in line 3) should also be slightly adjusted.

Algorithm 3 computes the set of simultaneous sets on an ART node. It uses two additional functions EnabledInteraction and DisabledInteraction. Function DisabledInteraction computes the set of disabled interactions on an ART node, which is simply the complement of the set of enabled interactions.

Algorithm 3. Simultaneous set computation

```
Input: an ART node $node = ((l_1, \phi_1), ..., (l_n, \phi_n), \phi)$
Output: a set of simultaneous sets $SSets$
$enabled\_interactions$ ← EnabledInteraction($node$)
$disabled\_interactions$ ← DisabledInteraction($node$)
create a worklist of interaction sets $wl$
push $enabled\_interactions$ into $wl$
while $wl \neq \emptyset$ do
    $current\_set$ ← pop($wl$)
    if exists $\gamma_1, \gamma_2 \in current\_set$, s.t. $\gamma_1, \gamma_2$ are dependent then
        $copy_1$ ← $current\_set - \{\gamma_1\}$
        $copy_2$ ← $current\_set - \{\gamma_2\}$
        push $copy_1, copy_2$ into $wl$
    else if exists $\gamma_1, \gamma_2 \in current\_set$, $\gamma_3 \in disabled\_interactions$,
                s.t. $\gamma_3, \gamma_1$ are dependent, and $\gamma_3, \gamma_2$ are dependent then
        $copy_1$ ← $current\_set - \{\gamma_1\}$
        $copy_2$ ← $current\_set - \{\gamma_2\}$
        push $copy_1, copy_2$ into $wl$
    else
        if $SSets$ does not contain $current\_set$ then
            push $current\_set$ into $SSets$
```

The basic idea is that starting from the set of enabled interactions, the algorithm progressively refines this set by splitting it into two sets. If two interactions from the set are dependent (line 7), or they are independent of each other, but dependent with a disabled interaction (line 11), then this set is split into two, each of which is obtained by removing one of the interactions (lines 8, 9 and 12, 13). Otherwise, if all interactions are independent of each other and with the disabled interactions, then the set is a simultaneous set and is added into the result set *SSets*.

Assume that, given two interactions γ_1 and γ_2, it takes $\mathcal{O}(1)$ time for the dependence check with precomputed dependence relation on lines 7 and 11. The while loop (line 5) executes at most $|enabled\_interactions|$ times, where $|enabled\_interactions|$ denotes the number of enabled interactions on the input ART node, since in each loop execution at most two interactions will be split and one simultaneous set will be added into the worklist *wl*. In the worst case, $|enabled\_interactions|^2 * |disabled\_interactions|$ checks need to be performs to find the two interactions to be split. Thus, the worst case time complexity of Algorithm 3 is $\mathcal{O}(|enabled\_interactions|^3 * |disabled\_interactions|)$.

The correctness of Algorithm 3 is straightforward, according to the simultaneous set constraints in Definition 10.

Theorem 3 (Correctness of Lazy Predicate Abstraction with Simultaneous Set Reduction). *Given a BIP model, and for every terminating execution of the combination of Algorithms 1 and 3, the two properties of Theorem 10 still hold.*

Proof (Sketch). Algorithm 3 computes the set of simultaneous sets on an ART node. A simultaneous set on an ART node is a simultaneous set on the configurations that are covered by this ART node. Therefore, the theorem follows from Theorem 15.

5 Related Work

Although there are plenty of works on safety property verification in literature, we review the most related ones in two aspects. With respect to combining abstraction techniques with explicit state reduction techniques, the works most related to ours are [8,9,24]. In [8,9] the authors propose two ESST-based verification techniques for multi-threaded programs with a preemptive and stateful scheduler (e.g. SystemC [20] and FairThreads [7]). The work in [24] combines classical lazy abstraction and partial order reduction [11] for the verification of generic multi-threaded programs with pointers. The difference between these works and ours is that they combine the abstraction techniques with classical partial order reduction techniques, (e.g. persistent set approach [11] and ample set approach [10]) in which one reduces the interleavings of concurrent transitions by exploring only a representative subset of all enabled transitions. In our approach, we leverage the BIP operational semantics to tackle this issue by executing concurrent interactions simultaneously.

With respect to the compositional verification, the most related ones are [6,13,21]. In [6] the authors presents an assume-guarantee abstraction refinement technique for compositional verification of component-based systems. However, the target system model is finite state and without data transfer. In [13] the authors propose a compositional verification technique for multi-threaded programs based on abstract interpretation framework. This algorithm relies on solving recursion-free Horn clauses to refine the abstraction. Later the work in [21] combines this method with a reduction technique based on Lipton's theory of reduction [19]. The programming model is quite different from ours. They handle shared variable concurrent programs, whereas BIP does not provide communication through shared variables, but only multiparty synchronisation and data transfer.

6 Experimental Evaluation

We implemented the proposed techniques in our prototype tool BIPChecker, based on the symbolic model checker nuXmv and the SMT solver MathSAT. In the experimental evaluation, we took a set of benchmarks from the literature, including the untimed temperature and railway control system [18], the ATM transaction model [4], the leader election algorithm [1], and the Quorum consensus algorithm [12]. We modelled them in the BIP framework and verified different safe and unsafe invariant properties. All these benchmarks (1) are scalable in terms of the number of components; (2) are infinite-state, using potentially unbounded integer variables and (3) feature data transfer on interactions.

All the experiments have been performed on a 64-bit Linux PC with a 2.8 GHz Intel i7-2640M CPU, with a memory limit of 4 Gb and a time limit of 300 s per benchmark. We refer to our website[1] for all the benchmarks and the tool.

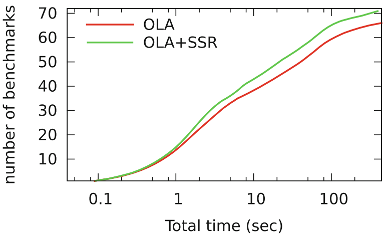

Fig. 2. Cumulative plot of time for solving all benchmarks

We run two configurations of BIPChecker: OLA and OLA+SSR, where OLA stands for on-the-fly lazy abstraction, and SSR stands for the simultaneous set reduction. We do not compare the performance of our tool with DFinder [4] and VCS [14], since they do not handle data transfer and infinite-state models respectively. The comparison of OLA and OLA+SSR on the full set of benchmarks is shown in Figs. 2 and 3.

In Fig. 2, we plot the cumulative time (x-axis) to solve an increasing number of benchmarks (y-axis), and in Fig. 3, we show the scatter plot of time for solving

[1] http://risd.epfl.ch/bipchecker.

each benchmark.[2] The plots show that simultaneous set reduction can improve the performance in general when it is combined with the on-the-fly lazy abstraction. In particular, from Fig. 3 we find that for safe models, OLA is comparable to OLA+SSR, while for unsafe models, OLA+SSR is always more efficient than OLA. In other words, OLA+SSR is more efficient to find counterexamples. This phenomenon can be explained because with simultaneous set reduction, some independent interactions are executed simultaneously, thus reducing the length of execution steps and being faster to detect counterexamples.

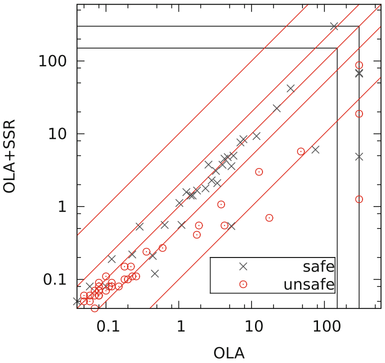

Fig. 3. Scatter plot of time for solving each benchmark

7 Conclusion

In this paper we proposed a generic approach to safety property verification of BIP models, which combines on-the-fly lazy abstraction and simultaneous set reduction technique. We also implemented our techniques in the BIPChecker tool. The experimental evaluation demonstrates the efficiency of the proposed approach. As future work we will investigate more efficient reduction techniques for component-based systems, that can boost the abstract reachability analysis, such as property guided reduction.

Acknowledgements. We want to thank Alessandro Cimatti, Marco Roveri and Sergio Mover for the instructive guidance during our collaboration that enabled this work and for their help with the nuXmv model checker and the MathSAT SMT solver, and all the anonymous reviewers for their careful reading of the paper.

References

1. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press, Cambridge (2008)
2. Basu, A., Bensalem, S., Bozga, M., Combaz, J., Jaber, M., Nguyen, T.H., Sifakis, J.: Rigorous component-based system design using the BIP framework. Softw. IEEE **28**, 41–48 (2011)
3. Behrmann, G., David, A., Larsen, K.G., Håkansson, J., Pettersson, P., Yi, W., Hendriks, M.: UPPAAL 4.0. In: QEST (2006)
4. Bensalem, S., Bozga, M., Nguyen, T.-H., Sifakis, J.: D-Finder: a tool for compositional deadlock detection and verification. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 614–619. Springer, Heidelberg (2009)

[2] Red diagonal guides provide a reference for comparison, each indicating shift of one order of magnitude.

5. Bliudze, S., Cimatti, A., Jaber, M., Mover, S., Roveri, M., Saab, W., Wang, Q.: Formal verification of infinite-state BIP models. In: Finkbeiner, B., et al. (eds.) ATVA 2015. LNCS, vol. 9364, pp. 326–343. Springer, Heidelberg (2015). doi:10.1007/978-3-319-24953-7_25
6. Gheorghiu Bobaru, M., Păsăreanu, C.S., Giannakopoulou, D.: Automated assume-guarantee reasoning by abstraction refinement. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 135–148. Springer, Heidelberg (2008)
7. Boussinot, F.: FairThreads: mixing cooperative and preemptive threads in C. Concur. Comput. Pract. Exp. **18**, 445–469 (2006)
8. Cimatti, A., Narasamdya, I., Roveri, M.: Software model checking with explicit scheduler and symbolic threads. Log. Methods Comput. Sci. **8**, 1–42 (2012)
9. Cimatti, A., Narasamdya, I., Roveri, M.: Verification of parametric system designs. In: FMCAD (2012)
10. Clarke Jr., E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
11. Godefroid, P.: Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem. Lecture Notes in Computer Science, vol. 1032. Springer, Heidelberg (1996)
12. Guerraoui, R., Kuncak, V., Losa, G.: Speculative linearizability. In: PLDI (2012)
13. Gupta, A., Popeea, C., Rybalchenko, A.: Predicate abstraction and refinement for verifying multi-threaded programs. In: POPL (2011)
14. He, F., Yin, L., Wang, B.-Y., Zhang, L., Mu, G., Meng, W.: VCS: a verifier for component-based systems. In: Van Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 478–481. Springer, Heidelberg (2013)
15. Henzinger, T.A., Jhala, R., Majumdar, R., McMillan, K.L.: Abstractions from proofs. In: ACM SIGPLAN Notices. ACM (2004)
16. Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G.: Lazy abstraction. In: POPL (2002)
17. Hojjat, H., Konecný, F., Garnier, F., Iosif, R., Kuncak, V., Rümmer, P.: A verification toolkit for numerical transition systems - tool paper. In: FM (2012)
18. Hojjat, H., Rümmer, P., Subotic, P., Yi, W.: Horn clauses for communicating timed systems. In: HCVS (2014)
19. Lipton, R.J.: Reduction: a method of proving properties of parallel programs. Commun. ACM **18**, 717–721 (1975)
20. IEEE 1666: SystemC language Reference Manual (2005)
21. Popeea, C., Rybalchenko, A., Wilhelm, A.: Reduction for compositional verification of multi-threaded programs. In: FMCAD (2014)
22. Sifakis, J.: Rigorous system design. In: Foundations and Trends in Electronic Design Automation (2013)
23. Su, C., Zhou, M., Yin, L., Wan, H., Gu, M.: Modeling and verification of component-based systems with data passing using BIP. In: ICECCS (2013)
24. Wachter, B., Kroening, D., Ouaknine, J.: Verifying multi-threaded software with Impact. In: FMCAD (2013)