Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2014, Article ID 703098, 12 pages
http://dx.doi.org/10.1155/2014/703098

Research Article

Compositional Abstraction Refinement for Component-Based Systems

Lianyi Zhang, Qingdi Meng, and Kueiming Lo

School of Software, Tsinghua University, Beijing 100084, China

Correspondence should be addressed to Lianyi Zhang; lyzhang117@gmail.com

Received 14 March 2014; Accepted 9 May 2014; Published 11 June 2014

Academic Editor: Xiaoyu Song

The efficiency of the compositional verification of invariants depends on the abstraction, which may lead to verification incompleteness. The invariant strengthening and state partitioning techniques are proposed in this paper. The former could refine the overapproximation by removing the unreachable states, and the latter is a variant of counterexample-guided abstraction refinement. Integrated with these two refinement techniques, a unified compositional verification framework is presented to strengthen the abstraction and find counterexamples. Some examples are included to show that the verification of the safety properties in component-based systems has been achieved by our framework.

1. Introduction

A component-based system is made up of several components using a set of glue (interactions) that describes the coordination between components [1]. Some components, like program processes, contain local data and control information to determine their own behaviors, and their interactions represent the communication between components. Component-based system has the advantages that the development can be simplified and the time-to-market would be reduced. However, the tremendously high (or even infinite) number of states of the components makes the verification of properties of these systems intractable. In this paper, we discuss the compositional verification using abstraction refinement techniques on component-based systems.

Compositional verification would alleviate the problem of state explosion in concurrent model checking, especially in conjunction with abstraction. The work in [2, 3] proposed an automated compositional abstraction framework which differs in the communication method. The abstraction procedure in [2] is guided by the data (interaction through shared variables) of the model while it is guided by the action (interaction through message-passing event) in [3]. Thread modular reasoning is also a compositional abstraction framework proposed in [4], which automatically generates the environment for each thread by analyzing its interaction. Compositional verification using abstraction refinement techniques on component-based systems has been extensively studied. Malkis et al. combined CEGAR and thread modular reasoning in their proposed technique [5], called thread-modular CEGAR. The technique computed reachable states with Cartesian abstraction. They directly compute the reachable states for all processes of the compositional system in an explicit way. Then refinement was applied to eliminate unreachable states by removing them from the Cartesian product. A technique closely related to compositional verification is assume-guarantee reasoning. Henzinger et al. [6, 7] proposed a component-based verification algorithm which is complete for verifying safety properties. They first initialize each component as true (the most abstract model which essentially accepts any behaviors) and then iteratively refine them by adding new auxiliary predicates, whereas they initialize the guarantee of the components as false (the most abstract model which essentially rejects any behaviors) and refine them successively by considering abstraction of current component and guarantees of other components.

Another interesting branch for compositional verification is the inductive invariant based method [8–10]. The properties established from the verification are usually invariants.

Given a property on component-based system states, compositional verification of invariant is to verify whether the given property can be inferred from the invariants of their components. Divide-and-conquer approaches can be applied in compositional verification to infer global invariants from local invariants. A compositional verification method based on the inductive invariant is presented in [11, 12]. In this approach, the property of the component-based system can be inferred by the compositional verification rule which is referred to as the component invariant and the interaction invariant. This approach is incomplete. Given a property (predicate), if the conjunction of the component invariants and the interaction invariants is able to establish the property, one concludes the verification. Otherwise, one reports that the verification is inconclusive.

The inconclusive problem is due to two reasons. One is the *abstraction*, which transforms the original system (infinite) to an abstract system (finite). The other is the *interaction invariant*, which is the overapproximation of the reachable abstract states. Thus, the conjunction of component invariants and the interaction invariant is such an overapproximation that unreachable states in the original system may be computed as reachable in the abstract system, which may include unreachable error states. We propose two refinement techniques, counterexample-guided invariant strengthening and state partitioning to address the inconclusive problem.

A unified compositional abstraction refinement algorithm for the invariant checking problem on component-based systems is given, which contains two phases. At the verification phase, we apply compositional verification to the invariant checking problem. If it suffices to conclude the verification, we are done. Otherwise, the verification algorithm moves to the refinement phases. In the other phase, we first strengthen the interaction invariant and remove the spurious counterexamples. After the invariant is strengthened, we analyze the remaining counterexamples and partition abstract states. When our algorithm returns to the verification phase, added states will induce a refined abstract model.

The rest is organized as follows. The component-based system, component invariant, and component abstraction are introduced in Section 2. In Section 3, we introduce the compositional verification of invariant for component-based systems, which is the overapproximation of the system states. We give counterexample-guided invariant strength and partition refinement methods in Section 4. A compositional verification framework integrated with iterative refinement and its correctness proof are presented in Section 5. We show the effectiveness of our proposed method in Section 6 and then conclude in Section 7.

2. Preliminaries

In this section, we present concepts and definitions that are used in this paper. We start with the overview of component-based systems, which includes the concepts of atomic component, interaction, and compound component. In the rest of this section, we introduce the concepts for the component invariant and abstraction.

2.1. Component-Based System. Component-based system is a basic model of wide-ranging concurrent systems. It is proven in [13] that such a hierarchical model can be converted to semantically equivalent flat model. Therefore, instead of analyzing complex hierarchical structure, we concentrate on two-level structure in this paper. The lower level of the component-based system is atomic component, and the higher level is compound component which is composed of several atomic components with a set of interactions.

An *atomic component* is a transition system, denoted by a tuple $B = (L, P, T, X, \{g_t\}_{t\in T}, \{f_t\}_{t\in T})$, where (L, P, T) is a transition system; that is, $L = \{l_1, l_2, \dots, l_k\}$ is a set of control locations, P is a set of ports, and $T \subseteq L \times P \times L$ is a set of transitions. $X = \{x_1, \dots, x_n\}$ is a set of variables and for each $t \in T$, respectively, $g_t(\mathbf{x})$ is a guard, a predicate on $\mathbf{x}$, and $f_t(\mathbf{x}, \mathbf{x}')$ is an update relation, a formula on $\mathbf{x}$(current) and $\mathbf{x}'$(next) state variables. Given a transition $t = (l, p, l') \in T$, l and l' are, respectively, the source and target location denoted by $\bullet t$ and $t\bullet$.

Given a set of components $B_1, B_2, \dots, B_n$ where $B_i = (L_i, P_i, T_i, X_i, \{g_t\}_{t\in T_i}, \{f_t\}_{t\in T_i})$, an interaction a is a set of ports, subset of $\bigcup_{i=1}^{n} P_i$, such that, for all $i = 1, \dots, n$ s.t. $|a \cap P_i| \leq 1$. The trans(a) is a set of transitions $\{t_i = (l_i, p_i, l_i') \in T_i\}_{i\in I}$, for arbitrary $I \subseteq \{1, \dots, n\}$, which have to execute synchronously.

A compound component $\gamma(B_1, \dots, B_n)$ is also a transition system, denoted by $B = (L, \gamma, T, X, \{g_t\}_{t\in T}, \{f_t\}_{t\in T})$, where

(1) (L, γ, T) is the transition system such that

  (i) $L = L_1 \times L_2 \times \cdots \times L_n$ is a set of control locations,

  (ii) $T \subseteq L \times \gamma \times L$ contains transitions $t = ((l_1, \dots, l_n), a, (l_1', \dots, l_n'))$ obtained by synchronization of sets of transitions $\{t_i = (l_i, p_i, l_i') \in T_i\}_{i\in I}$ such that $\{p_i\}_{i\in I} = a \in \gamma$ and $l_j' = l_j$ if $j \notin I$, for arbitrary $I \subseteq \{1, \dots, n\}$;

(2) $X = \bigcup_{i=1}^{n} X_i$ and for $a \in \gamma$, the transition trans(a) resulting from the synchronization of a set of transitions $\{t_i\}_{i\in I}$, the associated guard and update predicate are, respectively, $g_t = \bigwedge_{i\in I} g_{t_i}$ and $f_t = \bigwedge_{i\in I} f_{t_i} \wedge \bigwedge_{i\notin I}(X_i' = X_i)$.

A typical transition system (e.g., a component B) contains control states, usually the program counter which takes values from the set of control locations. Control states are defined by formulas of the form (at_l_i), where $l_i \in L$. Additionally, we assume that for each transition $t = (l_i, p, l_j) \in T$, we get a transition formula written as at_$l_i \wedge g_t(\mathbf{x}) \mapsto f_t(\mathbf{x}, \mathbf{x}')$; at_$l_j$ where $\mathbf{x} \in X$.

Definition 1 (component system). A component system is denoted by $\mathcal{S} = (B, \text{Init})$, where B is a component (atomic or compound component) and Init is the initial predicate of the system.

2.2. Component Invariant. Most properties established during verification are either invariants or depend on invariants. To prove that a predicate ϕ is an invariant of some system $\mathcal{S}$, we need to find such a predicate ψ that ψ is stronger than ϕ and ψ is inductive. In general, for a system S, ψ is an invariant of S if every initial state of system S satisfies ψ and ψ is preserved under all transitions of the system.

Definition 2 (component invariant). Given a component $B = (L, P, T, X, \{g_t\}_{t\in T}, \{f_t\}_{t\in T})$, for a global state predicate $\Phi = \bigvee_{l\in L}\text{at\_}l \wedge \varphi_l$, the generated formula by post predicate is $\text{post}(\Phi) = \bigvee_{l\in L}(\bigvee_{t=(l,p,l')}\text{at\_ } l' \wedge \text{post}_t(\varphi_l))$, where $\text{post}_t(\varphi(X)) = \exists X' \cdot (g_t(X') \wedge f_t(X', X) \wedge \varphi(X'))$. A fixed point of the post transformer is a component invariant (formula Φ) of the component B; that is, $\text{post}(\Phi) \Leftrightarrow \Phi$.

The component invariant is computed by the post predicate transformer. A more detailed technique, as well as other related properties for component invariants, is given in [8, 14] and will not be presented.

Consider a system $\mathcal{S} = (B, \text{Init})$, where B is a component and Init is a state predicate satisfied by the initial states of B. Φ is an invariant of S, if it is a component invariant of B and $\text{Init} \Rightarrow \Phi$.

2.3. Component Abstraction. When original system has real or large range discrete variables, enumeration of concrete states by valuation is impossible. Before computing interaction invariants of the compound system, we need first to generate a finite state abstraction for the component-based system.

In our methodology, given a component-based system $\mathcal{S} = \langle B, \text{Init}\rangle$, component invariant $\Phi = \bigvee_{l\in L}\text{at\_}l \wedge (\bigvee_{m\in M_l}\varphi_{lm})$ is automatically generated from a transition system (a component) that describes its behavior, which is represented by the disjunction of atomic formulas, such as $\text{at\_}l \wedge \varphi_{lm}$. However, instead of using the atomic formulas to generate an abstract system, we use them to construct an *abstraction function*.

The abstraction function maps an atomic formula to an abstract value of that formula, which preserves the relation among the atomic formulas instead of treating them as independent propositions. The abstraction technique used here is based on the approach proposed by Bensalem et al. in [2, 8]. An abstraction function α maps the concrete representation of the atomic predicates $\text{at\_}l \wedge \varphi_{lm}$ to the abstract presentation ϕ, called abstract state. We use Φ^α to represent a set of abstract states.

Definition 3 (abstract component). Given a component B, a component invariant Φ of B, and the associated abstraction function α, the abstract component B^α is denoted by $B^\alpha = (\Phi^\alpha, P, \rightsquigarrow)$, where

(i) Φ^α is the abstract state, where Φ is the component invariant and α is abstraction function,

(ii) P is a set of ports of B,

(iii) $\rightsquigarrow$ is the transition of B. For any pair of abstract states $\phi = \alpha(\text{at\_}l \wedge \varphi)$ and $\phi' = \alpha(\text{at\_}l' \wedge \varphi')$, there exists the transition from ϕ to ϕ' through p (denoted by $\phi \overset{p}{\rightsquigarrow} \phi'$) if and only if $\exists t = (l, p, l') \in T$, $\text{post}_t(\varphi) \wedge \varphi' \neq \text{false}$.

3. Compositional Verification

After the overview of component-based system, the component invariant, and the component abstraction, we first present the compositional verification rule of component-based systems, then we give out the concepts of the verification rules, and last we do the analysis on the overapproximation of the compositional abstraction.

3.1. Compositional Verification Rule. In the approach presented in [11, 12], the property Φ of $\gamma(B_1, \ldots, B_n)$ can be inferred by the following compositional verification rule:

$$\frac{\{B_i\langle\Phi_i\rangle\}_i^n, \Psi \in \text{II}\left(\{B_1^{\alpha_1}, \ldots, B_n^{\alpha_n}\}, \gamma\right), \left(\bigwedge_i^n \Phi_i\right) \wedge \Psi \Rightarrow \Phi}{\gamma(B_1, \ldots, B_n)\langle\Phi\rangle}, \tag{1}$$

where $B_i\langle\Phi_i\rangle$ means that Φ_i is an invariant of component B_i and Ψ belongs to the set of interaction invariants II. For component B_i, a predicate Φ_i on the component states is a component invariant of B_i. Before generating the interaction invariants, one needs to compute the abstraction for the components. The abstraction for B_i, denoted by $B_i^{\alpha_i}$, is computed from Φ_i by the abstraction function α_i. In the abstract system $\gamma(\{B_1^{\alpha_1}, \ldots, B_n^{\alpha_n}\})$, we generate the interaction invariant Ψ under the global behavior constraint, which is the overapproximation of the reachable abstract states. Given a property (predicate) Φ, if the conjunction of the invariant $\bigwedge_i^n\Phi_i$ and the interaction invariants Ψ are able to establish the property, one concludes the verification. Otherwise, one reports that the verification is inconclusive.

3.2. Related Concepts. Consider a compound system $\mathcal{S} = \langle\gamma(B_1, \ldots, B_n), \text{Init}\rangle$ and a set of component invariants $\Phi_1, \ldots, \Phi_n$ associated with the atomic components. If $B_i^{\alpha_i}$ is an abstraction of B_i with respect to an invariant Φ_i and its abstraction function α_i for $i = 1, \ldots, n$, then $B^\alpha = \gamma(B_1^{\alpha_1}, \ldots, B_n^{\alpha_n})$ is an abstraction of $B = \gamma(B_1, \ldots, B_n)$ with respect to $\bigwedge_{i=1}^n \Phi_i$ and an abstraction function α obtained as the composition of the α_i.

Definition 4 (abstract system). Given a component system $\mathcal{S} = (B, \text{Init})$, the abstract system is denoted by $\mathcal{S}^\alpha = \langle B^\alpha, \text{Init}^\alpha\rangle$, where B^α is the abstraction for a component and $\text{Init}^\alpha = \bigvee_{\phi\in\Phi_0^\alpha}\text{at\_}\phi$, where $\Phi_0^\alpha = \{\phi \in \Phi^\alpha \mid \alpha^{-1}(\phi) \wedge \text{Init} \neq \text{false}\}$ is the set of the initial abstract states.

The following lemma in [11] says that $\gamma(B_1^{\alpha_1}, \ldots, B_n^{\alpha_n})$ is an abstraction of $B = \gamma(B_1, \ldots, B_n)$ and that invariants of the abstract system are also invariants of the concrete system.

Lemma 5. *If B^α is an abstraction of B with respect to an invariant Φ and α its abstraction function, then B^α simulates*

B. Moreover, if Φ^{α} is an invariant of $\mathcal{S}^{\alpha}$, then $\alpha^{-1}(\Phi^{\alpha})$ is an invariant of $\mathcal{S}$.

Given an abstract system, interaction invariants characterize constraints on the global abstract state space induced by synchronization among components. Interaction invariants are based on the concept of traps in Petri net, which are computed on finite transition system $\gamma(B_1^{\alpha_1}, \ldots, B_n^{\alpha_n})$ without variables but abstract locations ϕ.

Consider a compound system $\gamma(B_1^{\alpha_1}, \ldots, B_n^{\alpha_n})$, where $B_i^{\alpha_i} = (\Phi_i^{\alpha_i}, P_i, \rightsquigarrow_i)$ is a transition system. For a set of abstract states $\Phi^{\alpha} \subseteq \bigcup_{i=1}^{n} \Phi_i^{\alpha_i}$, the forward interaction set is denoted by $\Phi^{\alpha}{\scriptstyle\blacksquare} = \bigcup_{\phi \in \Phi^{\alpha}} \phi{\scriptstyle\blacksquare}$ where $\phi{\scriptstyle\blacksquare} = \{\{\tau_i\}_{i \in I} \mid \forall i \cdot \tau_i \in \rightsquigarrow_i \wedge \exists i \cdot {\scriptstyle\blacksquare}\tau_i = \phi \wedge \{\text{ port } (\tau_i)\}_{i \in I} \in \gamma\}$ is related to some interactions. In each interaction, there exists a transition τ_i from ϕ participate in. That is, $\phi{\scriptstyle\blacksquare}$ is composed of sets of transitions involved in some interaction of γ in which a transition τ_i from ϕ can participate.

Given a parallel composition $\gamma(B_1^{\alpha_1}, \ldots, B_n^{\alpha_n})$, where $B_i^{\alpha_i} = (\Phi_i^{\alpha_i}, P_i, \rightsquigarrow_i)$, a trap [15] is a set Φ^{α} of abstract locations $\Phi^{\alpha} \subseteq \bigcup_{i=1}^{n} \Phi_i^{\alpha_i}$ such that $\Phi^{\alpha}{\scriptstyle\blacksquare} \subseteq {\scriptstyle\blacksquare}\Phi^{\alpha}$.

Definition 6 (interaction invariant). Given an abstract system $\mathcal{S}^{\alpha} = \langle \gamma(B_1^{\alpha_1}, \ldots, B_n^{\alpha_n}), \text{Init}^{\alpha} \rangle$, if the set of locations $\Phi^{\alpha} \subseteq \bigcup_{i=1}^{n} \Phi_i^{\alpha_i}$ is a trap containing the initial states of some components, then $\bigvee_{\phi \in \Phi^{\alpha}} \text{at\_}\phi$ is the interaction invariant of $\mathcal{S}^{\alpha}$.

The characterization of traps in [12] allows one to compute the set of traps [16] using different approaches, including methods of positive mapping or fix-pointed computation.

3.3. Overapproximation. In the compositional verification rule present in Section 1, the conjunction of component invariants and the interaction invariant is an *overapproximation* of reachable system states. The overapproximation of the compositional verification mainly results from two kinds of abstractions. One is due to the *abstraction function*, and the other is caused by interaction interference. The conjunction of *interaction invariants* overapproximates the set of reachable states, which may cause some unreachable error states to be included.

The abstract system B^{α} for a component B is generated by elimination in a conservative way. To check whether $\phi \rightsquigarrow \phi'$, where $\phi = \alpha(\text{at\_}l \wedge \varphi)$ and $\phi' = \alpha(\text{at\_}l' \wedge \varphi')$, we check that for all transitions $t = (l, p, l')$ we have $\text{post}_t(\varphi) \wedge \varphi' = \text{false}$. When we generate abstract systems, the behaviors of the abstract system are more than the original system's. As the compositional verification performance on abstract components, the verification is incompleteness.

The conjunction of interaction invariant (a trap) characterizes constraints on the states induced by synchronization among components. States satisfying interaction invariants provide enabled execution information, a local projection of global states on part of components. However, these global states may not be reachable from initial states. Moreover, different interactions may include common ports. The interactions including the same port must exclusively execute, that is, just only one interaction can execute. Interaction invariant cannot characterize this dynamic information.

From the above analysis, we can conclude that invariant-based compositional verification is incomplete. We propose two refinement techniques in conjunction with invariant-based compositional verification rule to get a verification framework. The framework is an iterative scheme which starts from the abstraction until a concrete counterexample is found or until the safety property holds on abstract systems.

4. Refinement Approaches

We give counterexample-guided invariant strengthening and partition refinement methods in this section.

4.1. Invariant Strengthening. In the first part of this section, we present the invariant strengthening approach. As interaction invariant computed by the greatest fixed point is the overapproximation, counterexamples (represented by the conjunction of abstract states) may be spurious. For this case, we check whether counterexamples can be reachable from the initial states. If not, we generate a fixed point backward from the spurious counterexample and strengthen the invariant by the fixed point.

Given an abstract component system, $\mathcal{S}^{\alpha} = \langle B^{\alpha}, \text{Init}^{\alpha} \rangle$, where $B^{\alpha} = \gamma(B_1^{\alpha_1}, \ldots, B_n^{\alpha_n})$ and $B_i^{\alpha_i} = (\Phi_i^{\alpha_i}, P_i, \rightsquigarrow_i)$. Formally, $\mathcal{S}^{\alpha} = (\Phi, \mathcal{T}, \text{Init}^{\alpha})$ is a finite state machine (FSM) in which

(i) Φ is a set of global states; for the abstract state $\phi = (\phi_1, \ldots, \phi_n)_{\phi_i \in \Phi_i^{\alpha_i}}$, we can represent it as conjunction (a vector) of local abstract state; that is, $\phi = \bigwedge_{\phi_i \in \Phi_i^{\alpha_i}} \phi_i \in \Phi$;

(ii) $\mathcal{T}$ is a predicate on global states Φ and Φ'; for $((\phi_1, \ldots, \phi_n), a, (\phi_1', \ldots, \phi_n'))$, where $a \in \gamma$, there exists $\mathcal{T}(\phi, \phi')$, where $\phi = \bigwedge_{\phi_i \in \Phi_i^{\alpha_i}} \phi_i$ and $\phi' = \bigwedge_{\phi_i' \in \Phi_i^{\alpha_i}} \phi_i'$;

(iii) Init^{α} is the initial predicate of the abstract system.

A formula φ is interpreted as the set $|[\varphi]|$ of all the global states; $\phi \in \Phi$ satisfy φ. We define the set $\text{Reach}_{\mathcal{T}}(\text{Init}^{\alpha})$ of the global states *reachable* from the states $|[\text{Init}^{\alpha}]|$ via the transition as the smallest set such that $|[\text{Init}^{\alpha}]| \subseteq \text{Reach}_{\mathcal{T}}(\text{Init}^{\alpha})$; and $\{\phi' \mid \exists \phi \cdot \mathcal{T}(\phi, \phi')\} \subseteq \text{Reach}_{\mathcal{T}}(\text{Init}^{\alpha})$ if $\phi \subseteq \text{Reach}_{\mathcal{T}}(\text{Init}^{\alpha})$. Consider an abstract component system $\mathcal{S}^{\alpha} = (\Phi, \mathcal{T}, \text{Init}^{\alpha})$, where Φ is a set of the global states, $\mathcal{T}$ is the transition predicate, and Init^{α} is the initial predicate. The converse transition system $(\Phi, \mathcal{T}^{-1}, \text{Init}^{\alpha})$ is defined by $\mathcal{T}^{-1}(\phi', \phi) = \mathcal{T}(\phi, \phi')$. For a formula φ, we compute all the successor states which are generated from the states hold φ by $\text{sp}_{\mathcal{T}}(\varphi)$. $\text{sp}_{\mathcal{T}}(\varphi) = \{\phi' \mid \exists \phi \cdot (\mathcal{T}(\phi, \phi') \wedge \varphi)\}$, where $\phi, \phi' \in \Phi$ and ϕ holds φ. Likewise, we can define $\text{sp}_{\mathcal{T}^{-1}}(\varphi) = \{\phi' \mid \exists \phi \cdot (\mathcal{T}(\phi', \phi) \wedge \varphi)\}$, where $\phi, \phi' \in \Phi$ and ϕ holds φ.

The following lemma [17] says that if none of the states represented by ϕ is backward reachable from the initial states, then $\neg\phi$ is an invariant.

Lemma 7. *Let $\mathcal{S}^{\alpha} = \langle \Phi, \mathcal{T}, \text{Init}^{\alpha} \rangle$ be a transition system and ϕ an arbitrary formula (or states). If φ is such that $(sp_{\mathcal{T}^{-1}}(\varphi) \vee$*

$\phi) \Rightarrow \varphi$ and the formula $Init \wedge \varphi$ is unsatisfiable, then $\neg\varphi$ is an inductive invariant of $\mathcal{S}^{\alpha}$.

Corollary 8. *If $Reach_{\mathcal{T}^{-1}}(\varphi) \cap Init = \emptyset$, then the formula corresponding to the complement of the set of $Reach_{\mathcal{T}^{-1}}(\varphi)$ is an inductive invariant.*

Theorem 9. *Assume that vilo is a counterexample and vilo $\wedge\ \alpha^{-1}(\phi) \neq$ false, where ϕ is an abstract state of $\mathcal{S}^{\alpha}$. If $Reach_{\mathcal{T}^{-1}}(\phi) \cap Init^{\alpha} = \emptyset$, then vilo is the spurious counterexample and the complement of the set of $Reach_{\mathcal{T}^{-1}}(\phi)$ is used to strengthen the invariants of S^{α}.*

Inspired by the above theorem, we propose an approach, called the invariant strengthening, to strengthen the invariants from the compositional verification rule.

We first choose a counterexample and generate a fixed point from the selected counterexample using backward propagation. An abstract state including a counterexample is the conjunction formula of abstract states like $\phi = \bigwedge_{\phi_i \in \Phi_i^{\alpha_i}} \phi_i$, where $\phi_i = \alpha_i(\mathrm{at\_}l \wedge \varphi)$, which is the product of abstract states of each component. On the abstract system $\mathcal{S}^{\alpha}$, we do the backward propagation by $\mathrm{sp}_{\mathcal{T}^{-1}}(\phi) = \{\phi' \mid \exists\phi \cdot (\mathcal{T}(\phi', \phi) \wedge \phi)\}$. Then from a counterexample, we can compute all the global abstract system states using backward propagation. If the intersection of the generated fixed point and the initial states is empty, the counterexample and all of the backward generated fixed points are not reachable from initial states. For this case, we say the selected counterexample is spurious. The invariant strengthening approach removes the spurious counterexample and eliminates the unreachable states generated from the spurious counterexample to strengthen the invariant.

For invariant strengthening, time consumes at counterexample reachable global states set computation. As this set computation is based on fixed-pointed computation and is monotonic, so time complexity for Algorithm 1 is $\mathcal{O}(\Phi)$, where Φ is number of global states of abstract system and $\Phi = \prod_{i=1}^{n} \Phi_i$ and $\Phi_i = |\Phi_i^{\alpha}|$ the number of abstract states from ith component.

4.2. Partition Refinement. In this part, we proposed a more effective counterexample-guided method, called partition refinement, which can accelerate abstraction refinement mechanism. Consider the abstraction function which transforms the original infinite system to the finite abstract system. An abstract state is an aggregation of a number of original system states. If a concrete state Φ_c is part of an abstract state Φ, we cannot make decisions (1) whether Φ is a counterexample, if Φ_c is a counterexample in concrete system; (2) whether Φ_c is reachable in concrete system, although Φ is reachable in abstract system. We propose our refinement approach to give the solution of the above question (Figure 1).

In the above approach, we analyze whether the remaining counterexamples are the spurious counterexamples which are caused by inaccurate characterization. To eliminate these spurious counterexamples, we refine the abstract component states by the proposed state partition approach. Consider an abstract system state $\phi^{\alpha} = \mathrm{at\_}l \wedge \varphi$ and a counterexample $\phi_e = \mathrm{at\_}l \wedge \varphi_e$ such that $\varphi_e \Rightarrow \varphi$ (Dec$_0$). We can partition ϕ^{α} into two states ϕ_e and ϕ', $\phi_e = \mathrm{at\_}l \wedge \varphi_e$ and $\phi' = \mathrm{at\_}l \wedge \varphi'$, where $\varphi' = \varphi - \varphi_e$; here we denote $\phi_e \vee \phi' = \phi^{\alpha}$ and $\phi_e \wedge \phi' =$ false. Assume that there exist ϕ_1^{α} and ϕ_2^{α}, from which ϕ^{α} is reachable such that $\mathrm{post}_{\tau_1}(\alpha^{-1}(\phi_1^{\alpha}) \cdot \varphi) \wedge \alpha^{-1}(\phi_e) \cdot \varphi \neq$ false and $\mathrm{post}_{\tau_2}(\alpha^{-1}(\phi_2^{\alpha}) \cdot \varphi) \wedge \alpha^{-1}(\phi') \cdot \varphi \neq$ false (Dec$_1$) hold. We remove ϕ^{α} from abstract state space Φ^{α} and add ϕ_e and ϕ' into Φ^{α}.

From the above, we assume that ϕ_1^{α} reach the error-part ϕ_e of ϕ^{α} and ϕ_2^{α} reach the other ϕ'. In this case, we should add the transitions into the system. If $\exists\phi_i^{\alpha}, \exists\tau_i : (\phi_i^{\alpha}, p_i, \phi^{\alpha}) \in \rightsquigarrow$ such that $\mathrm{post}_{\tau_i}(\alpha^{-1}(\phi_i^{\alpha}) \cdot \varphi) \wedge \alpha^{-1}(\phi) \cdot \varphi \neq$ false (Dec$_2$), we add a transition from ϕ_i^{α} to ϕ (here, ϕ stands for both ϕ_e and ϕ'); likewise, if $\exists\phi_i^{\alpha}$, and $\exists\tau_i : (\phi^{\alpha}, p_i, \phi_i^{\alpha}) \in \rightsquigarrow$ such that $\mathrm{post}_{\tau}(\alpha^{-1}(\phi) \cdot \varphi) \wedge \alpha^{-1}(\phi_i^{\alpha}) \cdot \varphi \neq$ false (Dec$_3$), we add a transition from ϕ to ϕ_i^{α}.

Theorem 10. *If $\mathcal{S}_{ref}^{\alpha}$ is a refined system returned by Algorithm 2, then B_{ref}^{α} simulates B. Moreover, if Φ^{α} is an invariant of B_{ref}^{α}, then $\Phi = \alpha^{-1}(\Phi^{\alpha})$ is an invariant of B.*

Proof. By Lemma 5, we show that the Algorithm 2 only makes states partition and then has no effect on the simulation relation. If $(l_1, x) \xrightarrow{p} (l, x')$ and $\alpha^{-1}(\phi_1^{\alpha}) = \mathrm{at\_}l_1 \wedge \varphi_1$, then there exists ϕ^{α} such that $\alpha^{-1}(\phi^{\alpha}) = \mathrm{at\_}l \wedge \varphi$, $(l, x')R\phi^{\alpha}$, and $\phi_1^{\alpha} \overset{p}{\rightsquigarrow} \phi^{\alpha}$. If $\varphi_e \Rightarrow \varphi$ does not hold, then the abstract state ϕ^{α} cannot be partitioned by φ_e. Otherwise, we split abstract state ϕ^{α} with disjoint two parts $\phi = \alpha(\mathrm{at\_}l \wedge \varphi_e)$ and $\phi' = \alpha(\mathrm{at\_}l \wedge \varphi')$, and we have $\varphi_e \wedge \varphi' =$ false and $\varphi_e \vee \varphi' = \varphi$, so we have either $\phi_1^{\alpha} \overset{p}{\rightsquigarrow} \phi$ or $\phi_1^{\alpha} \overset{p}{\rightsquigarrow} \phi'$. Then, the refined abstract system $B_{\mathrm{ref}}^{\alpha}$ simulates B. If Φ^{α} is an invariant of $B_{\mathrm{ref}}^{\alpha}$, then $\alpha^{-1}(\Phi^{\alpha})$ is an invariant of B. □

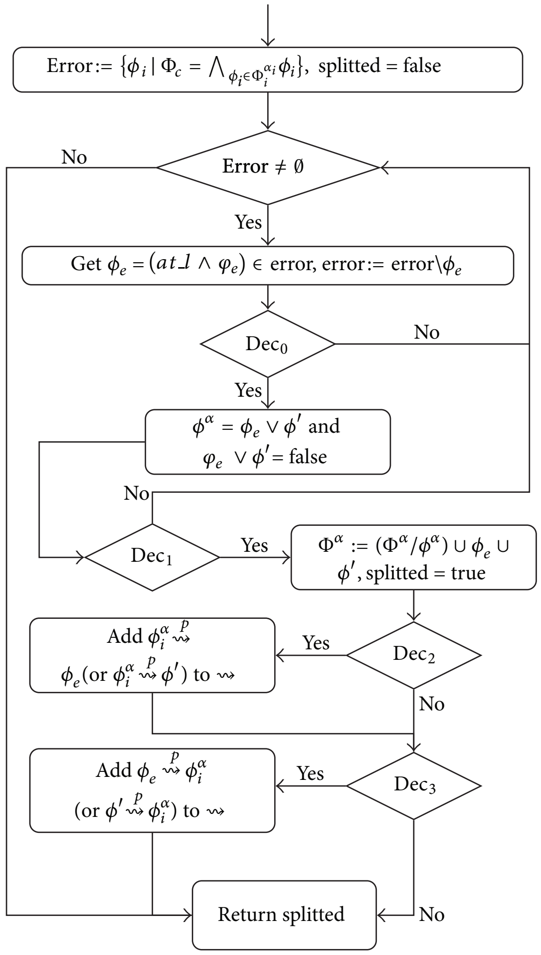
For partition refinement method, given a counterexample $\Phi_c = \bigwedge_{\phi_i \in \Phi_i^{\alpha_i}} \phi_i$, state partition in ith component, based on ϕ_i, only affects one abstract state to be parted. And a state is parted only once by ϕ_i. As exists quantifier for states needs examines all possible abstract state pairs and so does the quantifier of transitions, state partition needs time complexity as $\mathcal{O}(\sum_{i=1}^{n}(|\Phi_i^{\alpha}| \cdot |\rightsquigarrow_i|^2))$.

The above refinement approaches are applied on the parallel compositional system. However, extended with the decidable theories and symbolic representation, our approaches can be applied on a more complex system like the hierarchical component system.

5. Iteration Verification Framework

In this section, we present the unified verification framework composed of compositional abstraction and our proposed refinement techniques.

Our verification framework for component-based systems is an iterative scheme presented by Algorithm 1. In our framework, there exists the method of abstraction refinement



Error:= $\{\phi_i \mid \Phi_c = \bigwedge_{\phi_i \in \Phi_i^{\alpha_i}} \phi_i\}$, splitted = false
Error ≠ ∅
No
Yes
Get $\phi_e = (at\_l \wedge \varphi_e) \in$ error, error:= error\\$\phi_e$
Dec$_0$
No
Yes
$\phi^\alpha = \phi_e \vee \phi'$ and $\varphi_e \vee \phi'$ = false
No
Dec$_1$
Yes
$\Phi^\alpha := (\Phi^\alpha/\phi^\alpha) \cup \phi_e \cup \phi'$, splitted = true
Add $\phi_i^\alpha \overset{p}{\rightsquigarrow} \phi_e$(or $\phi_i^\alpha \overset{p}{\rightsquigarrow} \phi'$) to $\rightsquigarrow$
Yes
Dec$_2$
No
Add $\phi_e \overset{p}{\rightsquigarrow} \phi_i^\alpha$ (or $\phi' \overset{p}{\rightsquigarrow} \phi_i^\alpha$) to $\rightsquigarrow$
Yes
Dec$_3$
Return splitted
No

Figure 1

presented by Algorithm 2, which includes the counterexample guided invariant strengthening and abstract state partitioning inspired by CEGAR mechanism. In the Algorithm 1, the unified compositional abstraction refinement for the invariant checking problem on component-based systems is given, which contains two phases.

At the verification phase, we apply compositional verification to the invariant checking problem. If properties can be proven under the current abstraction (see line 12 in Algorithm 1), the verification succeeds immediately. Otherwise, the refinement is performed (see line 16 in Algorithm 1). If the abstraction cannot be further refined by the current reachable counterexample (see line 17 in Algorithm 1), then current counterexample is a genuine counterexample. If current counterexamples are all spurious counterexamples (see line 20 in Algorithm 1), we conclude that verification succeeds. Otherwise, we go to line 23; we get refined abstract system and compute the new more concrete interaction invariant with conjunction of strengthened inductive invariant computed from backward propagation of spurious counterexamples and go back to counterexample guiding abstraction refinement, portion of verification basis, which call for less efforts than previous iteration processes in [11, 12].

At the refinement phase, we first strengthen the interaction invariant and remove the spurious counterexamples. After the invariant is strengthened, we analyze the remaining counterexamples and partition abstract states. When our algorithm returns to the verification phase, added states will induce a refined abstract model. Our refinement method (see Algorithm 2) reconsiders information of counterexamples, which may include $\mathrm{Err}_{\mathrm{I}}$ and $\mathrm{Err}_{\mathrm{II}}$, where $\mathrm{Err}_{\mathrm{I}}$ is unreachable error state introduced by interaction interference and $\mathrm{Err}_{\mathrm{II}}$ is inaccurate local configuration caused by abstraction function. We use the invariant strengthening technique to remove $\mathrm{Err}_{\mathrm{I}}$ from the counterexamples and strengthen the invariant use of the counterexample in $\mathrm{Err}_{\mathrm{I}}$. We use the counterexample in $\mathrm{Err}_{\mathrm{II}}$ to do the state partition refinement which refines the abstraction until a genuine counterexample is found or the safety property holds on abstract systems.

Theorem 11. *Let $\mathcal{S} = \langle \gamma(B_1, \ldots, B_n), Init \rangle$ be a compound component-based system, and let Φ be a specific property predicate.*

(1) *If $IterationVerify(\mathcal{S}, \Phi)$ returns true, then $\mathcal{S} \models \Phi$;*

(2) *if $IterationVerify(\mathcal{S}, \Phi)$ returns false, then $\mathcal{S} \not\models \Phi$.*

```
     Input: Component system $\mathcal{S} = (\gamma(B_1, \ldots, B_n), Init)$, property $\Phi$
     Output: $true$ or $false$
(1)  $\Phi_i = true$ for each $i = 1, \ldots, n$, $Refinable := true$
(2)  for $i \leftarrow 1$ to $n$ do
(3)      compute the component invariant $\Phi_i'$ based on $B_i$
(4)      $\Phi_i := \Phi_i \wedge \Phi_i'$
(5)      compute the corresponding abstraction $B_i^{\alpha_i}$ based on $B_i$ and $\Phi_i$
(6)  end
(7)  from $\gamma(B_1^{\alpha_1}, \ldots, B_n^{\alpha_n})$, compute $L_1, L_2, \ldots, L_m$
(8)  for $k \leftarrow 1$ to $m$ do
(9)      $\Psi_k = \bigvee_{\phi \in L_k} \alpha^{-1}(\phi)$
(10) end
(11) $\Psi := \bigwedge_{k=1}^{m} \Psi_k$
(12) if $\neg\Phi \wedge \Psi \wedge (\bigwedge_{i=1}^{n} \Phi_i) = false$ then
(13)     $\Phi$ is an invariant of $\mathcal{S}$
(14)     return $true$
(15) else
(16)     compute counterexample set $Err := \{\Phi_c\}$
         $Refinable := AbsRefinement(\mathcal{S}^{\alpha}, Err, \Psi)$
(17)     if $Refinable = false \&\& Err \neq \emptyset$ then
(18)         return $false$
(19)     end
(20)     if $Err = \emptyset$ then
(21)         return $true$
(22)     else
(23)         goto 12
(24)     end
(25) end
```

Algorithm 1: Iteration verification.

Proof. (1) If Algorithm 1 returns true from line 14, with the precondition $\neg\Phi \wedge \Psi \wedge (\bigwedge_{i=1}^{n} \Psi_i) =$ false, we get the conclusion that the property satisfied immediately. Otherwise, if all current counterexamples are spurious counterexamples, the verification stops with the property satisfied at line 21. (2) If Algorithm 1 returns false from line 18 with Refinable $=$ false and Err $\neq \emptyset$, we get that the abstract system is unable to be refined any more by the current counterexample, which means the counterexample is a real violated behavior truly reachable from the initial configuration and cannot be eliminated by a more concrete system. Otherwise, abstract system should be refined by the next refinement using the current counterexamples. □

Theorem 12. *Let $\mathcal{S} = \langle \gamma(B_1, \ldots, B_n), Init \rangle$ be a compound component-based system, and let Φ be a specific property predicate. Algorithm $IterationVerify(\mathcal{S}, \Phi)$ always terminates.*

Proof. In each refinement iteration, either some new strengthening invariant is added or the abstract system state is split into new states. Note that the state space of abstract system is finite and that the number of possible split states is also finite. In the worst case, the algorithm finally terminated with the refined system is just the original system. Then the algorithm can terminate finally. □

Theorem 13. *Let $\mathcal{S} = \langle \gamma(B_1, \ldots, B_n), Init \rangle$ be a compound component-based system, and let Φ be a specific property predicate.*

(1) *If $\mathcal{S} \models \Phi$, then $IterationVerify(\mathcal{S}, \Phi)$ returns true;*

(2) *if $\mathcal{S} \not\models \Phi$, then $IterationVerify(\mathcal{S}, \Phi)$ returns false.*

Proof. (1) According to the second statements of Theorem 11, if $\mathcal{S} \models \Phi$, Algorithm 1 cannot return with false. According to Theorem 12, Algorithm 1 always terminates. Thus, if $\mathcal{S} \models \Phi$, then algorithm can only terminate with true.

(2) Similarly, according to the first statement of Theorems 11 and 12, if $\mathcal{S} \not\models \Phi$, the algorithm can only terminate with false. □

6. Examples and Experiments

We will provide certain examples to illustrate the effectiveness of our proposed verification framework.

6.1. Train Gate Controller.

Consider the example of the train gate controller [18], which consists of a *controller*, a *gate*, and a number of *trains*. The model presented in Figure 2 describes only one train interacting with the controller and the gate. The controller operates the gate up and down when a train enters and exits, respectively.

The *Controller* has four locations $\{c_0, c_1, c_2, c_3\}$, one variable z, five ports {approach, raise, exit, lower, tick}, and eight guarded transitions. The *Train* has three locations {far, in, near}, a variable x, four ports {approach, exit, t, tick}, and six guarded transitions. The *Gate* has four locations $\{g_0, g_1, g_2, g_3\}$, a variable

```
     Input: Abstract system $\mathcal{S}^{\alpha} = (\gamma(B_1^{\alpha_1}, \ldots, B_n^{\alpha_n}), Init^{\alpha})$,
            set of counterexamples $Err$, interaction invariant $\Psi$
     Output: $true$ or $false$
(1)  $Refined := false$, $Err_I := \emptyset$, $Err_{II} := \emptyset$
(2)  foreach $\Phi_c \in Err$ do
(3)      $Err := Err \setminus \Phi_c$
(4)      $InvStr_{\Phi_c} := InvarStrengthen(\mathcal{S}^{\alpha}, \Phi_c)$
(5)      if $InvStr_{\Phi_c} = true$ then
(6)          $Err_{II} := Err_{II} \cup \Phi_c$
(7)      else
(8)          $Err_I := Err_I \cup \Phi_c$
(9)      end
(10) end
(11) $Err := Err_{II}$
(12) if $Err_{II} = \emptyset$ then
(13)     return $false$
(14) else
(15)     foreach $\Phi_c \in Err_{II}$ do
(16)         if $SplitAbstraction(\mathcal{S}^{\alpha}, \Phi_c) = true$ then
(17)             $Refined := true$
(18)         end
(19)     end
(20)     from split refinement abstract system $\mathcal{S}^{\alpha}_{ref}$, compute
         new compute interaction invariant $\Psi$
(21)     foreach $\Phi_c \in Err_I$ do
(22)         $\Psi := \Psi \wedge InvStr_{\Phi_c}$
(23)     end
(24)     return $Refined$
(25) end
```

Algorithm 2: Abstraction refinement.

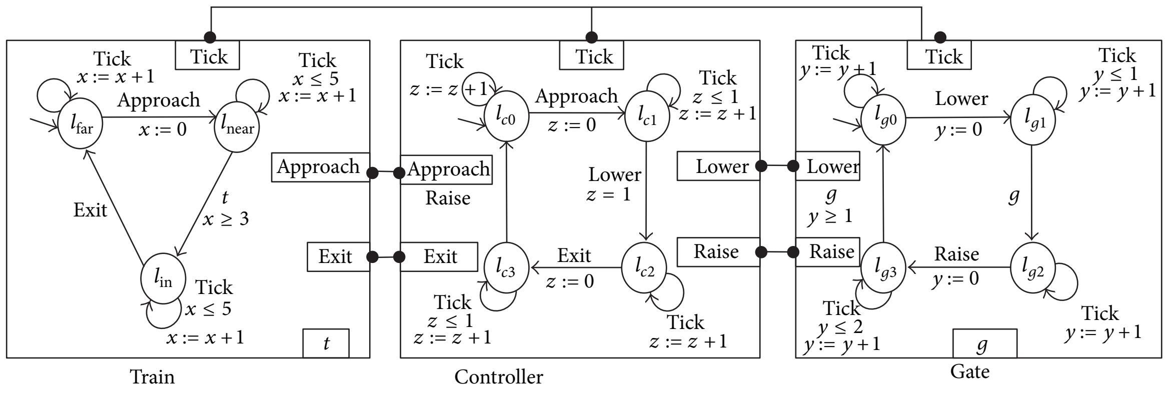


Figure 2: The component-based model for train gate controller (TGC).

y, four ports {lower, raise, g, tick}, and eight guarded transitions. Three atomic components Train, Gate, and Controller are composed with a set γ of interactions: {C.tick, T.tick, G.tick}, {T.approach, C.approach}, {T.exit, C.exit}, {C.lower, G.lower}, and {C.raise, G.raise}. In this example, we aim to verify the TGC system by the initial condition Init $= l_{\text{far}} \wedge (x = 3) \wedge l_{c_0} \wedge (z = 1) \wedge l_{g_0} \wedge (y = 1)$. For the above three atomic components, we generate the following predicates $\Phi_{\text{Controller}} = (l_{c_0} \wedge z \ge 0) \vee (l_{c_1} \wedge 0 \le z \le 1) \vee (l_{c_2} \wedge z \ge 1) \vee (l_{c_3} \wedge 0 \le z \le 1)$, $\Phi_{\text{Train}} = (l_{\text{far}} \wedge x \ge 3) \vee (l_{\text{near}} \wedge 0 \le x \le 5) \vee (l_{\text{in}} \wedge 3 \le x \le 5)$, and $\Phi_{\text{Gate}} = (l_{g_0} \wedge y \ge 1) \vee (l_{g_1} \wedge 0 \le y \le 1) \vee (l_{g_2} \wedge y \ge 0) \vee (l_{g_3} \wedge 0 \le y \le 2)$, which are, respectively, the component invariants of Train, Gate, and Controller. Since this system has an infinite number of reachable states. With the application of abstraction function α, we transform the original components (infinite) to the computed abstract components (finite) presented in Figure 3. The computed abstract states are $\{\phi_{n_1}, \phi_{n_2}, \phi_{n_3}, \phi_{\text{in}}, \phi_{\text{far}}, \phi_{c_{11}}, \phi_{c_{12}}, \phi_{c_2}, \phi_{c_{31}}, \phi_{c_{32}}, \phi_{c_0}, \phi_{g_{11}}, \phi_{g_{12}}, \phi_{g_2}, \phi_{g_{31}}, \phi_{g_{32}}, \phi_{g_0}\}$ (see Table 1).

Table 1

Train^{α}	$\text{Controller}^{\alpha}$	Gate^{α}
$\phi_{n_1} = l_{\text{near}} \wedge x = 0$	$\phi_{c_{11}} = l_{c_1} \wedge z = 0$	$\phi_{g_{11}} = l_{g_1} \wedge y = 0$
$\phi_{n_2} = l_{\text{near}} \wedge 1 \leq x \leq 2$	$\phi_{c_{12}} = l_{c_1} \wedge z = 1$	$\phi_{g_{12}} = l_{g_1} \wedge y = 1$
$\phi_{n_3} = l_{\text{near}} \wedge 3 \leq x \leq 5$	$\phi_{c_2} = l_{c_2} \wedge z \geq 1$	$\phi_{g_2} = l_{g_2} \wedge y \geq 0$
$\phi_{\text{in}} = l_{\text{in}} \wedge 3 \leq x \leq 5$	$\phi_{c_{31}} = l_{c_3} \wedge z = 0$	$\phi_{g_{31}} = l_{g_3} \wedge y = 0$
$\phi_{\text{far}} = l_{\text{far}} \wedge x \geq 3$	$\phi_{c_{32}} = l_{c_3} \wedge z = 1$	$\phi_{g_{32}} = l_{g_3} \wedge 1 \leq y$
	$\phi_{c_0} = l_{c_0} \wedge z \geq 0$	$\phi_{g_0} = l_{g_0} \wedge y \geq 1$

The interested safety property is that when the train is at the location l_{in}, the gate is at l_{g_2}. We apply the compositional verification rule to check the property. In this model, the verification rule can infer this property, so we get the conclusion of the properties satisfied.

6.2. Temperature Control System. Consider the example of the temperature control system [11, 12] presented in Figure 4. The model consists of three atomic components: Controller, Rod_1, and Rod_2.

The Controller has two locations $\{l_5, l_6\}$, a variable θ, three ports $\{\text{tick}, \text{cool}, \text{heat}\}$, and four guarded transitions. The Rod_1 has two locations $\{l_1, l_2\}$, a variable t_1, three ports $\{\text{tick}_1, \text{cool}_1, \text{rest}_1\}$, and four guarded transitions. Likewise, the Rod_2 has two locations $\{l_3, l_4\}$, a variable t_2, three ports $\{\text{tick}_2, \text{cool}_2, \text{rest}_2\}$, and four guarded transitions. Three atomic components Controller, Rod_1, and Rod_2 are composed with a set γ of interactions: $\{\text{tick}, \text{tick}_1, \text{tick}_2\}$, $\{\text{cool}, \text{cool}_1\}$, $\{\text{cool}, \text{cool}_2\}$, $\{\text{heat}, \text{rest}_1\}$, and $\{\text{heat}, \text{rest}_2\}$. In this example, we aim to verify deadlock-freedom of the temperature control system by the initial condition $\text{Init} = l_5 \wedge (\theta = 100) \wedge l_1 \wedge (t_1 = 3600) \wedge l_3 \wedge (t_2 = 3600)$.

For the above three atomic components, we generate the following predicates $\Phi_1 = (l_1 \wedge t_1 \geq 0) \vee (l_2 \wedge t_1 \geq 3600)$, $\Phi_2 = (l_3 \wedge t_2 \geq 0) \vee (l_4 \wedge t_1 \geq 3600)$, and $\Phi_3 = (l_5 \wedge 100 \leq \theta \leq 1000) \vee (l_6 \wedge 100 \leq \theta \leq 1000)$, which are, respectively, the component invariants of $\text{Rod}_1, \text{Rod}_2$, and Controller. Since this system has an infinite number of reachable states, with the application of abstraction function α, we transform the original components (infinite) to the computed abstract components (finite) in Figure 5. The computed abstract states are $\{\phi_{11}, \phi_{12}, \phi_{21}, \phi_{22}, \phi_{31}, \phi_{32}, \phi_{41}, \phi_{42}, \phi_{51}, \phi_{52}, \phi_{61}, \phi_{62}\}$.

The interaction invariant of the component system is generated from the concept of the trap. The sets of traps for the abstract system are $L_1 = \{\phi_{21}, \phi_{41}, \phi_{51}, \phi_{52}\}, L_2 = \{\phi_{11}, \phi_{12}, \phi_{21}, \phi_{31}, \phi_{32}, \phi_{41}\}$, $L_3 = \{\phi_{32}, \phi_{41}, \phi_{42}, \phi_{51}\}$, $L_4 = \{\phi_{11}, \phi_{12}, \phi_{31}, \phi_{32}, \phi_{61}, \phi_{62}\}$, and $L_5 = \{\phi_{12}, \phi_{21}, \phi_{22}, \phi_{51}\}$. After the computation of traps, we generate interaction invariants $\Psi_i = \bigvee_{\phi \in L_i} \alpha^{-1}(\phi)$ $(i = 1, 5)$ and then get $\Psi := \bigwedge_{i=1}^{5} \Psi_i$.

To verify the deadlock-freedom of the temperature control system, we define a predicate *DIS* which characterizes the set of system states from which all interactions in γ are disabled. In this example, $\text{DIS} = (\neg(l_5 \wedge \theta < 1000)) \bigwedge (\neg(l_6 \wedge \theta = 100) \vee \neg l_2) \bigwedge (\neg(l_6 \wedge \theta > 100)) \bigwedge (\neg(l_5 \wedge \theta = 1000) \vee \neg(l_3 \wedge t_2 \geq 3600)) \bigwedge (\neg(l_5 \wedge \theta = 1000) \vee \neg(l_1 \wedge t_1 \geq 3600)) \bigwedge (\neg(_6 \wedge \theta = 100) \vee \neg l_4)$. If the predicate $\neg\text{DIS}$ is an invariant of the temperature control system, then it is deadlock-free. To check that $\neg\text{DIS}$ is an invariant, we need a stronger invariant Φ such that $\Phi \Rightarrow \neg\text{DIS}$ (equivalently, $\Phi \wedge \text{DIS} = \text{false}$). Based on the previous compositional verification rule, the computed invariant is $\Phi = (\Phi_1 \wedge \Phi_2 \wedge \Phi_3) \wedge \Psi$. To verify the deadlock-freedom of the system, we computed the predicate $\Phi \wedge \text{Init} \wedge \text{DIS}$, which is reduced to

(1) $\Phi_{c1} = (l_1 \wedge 1 \leq t_1 < 3600) \wedge (l_3 \wedge 1 \leq t_2 < 3600) \wedge (l_5 \wedge \theta = 1000)$;

(2) $\Phi_{c2} = (l_1 \wedge 1 \leq t_1 < 3600) \wedge (l_4 \wedge t_2 \geq 3600) \wedge (l_5 \wedge \theta = 1000)$;

(3) $\Phi_{c3} = (l_2 \wedge t_1 \geq 3600) \wedge (l_3 \wedge 1 \leq t_2 < 3600) \wedge (l_5 \wedge \theta = 1000)$.

As the invariant $\Phi = (\Phi_1 \wedge \Phi_2 \wedge \Phi_3) \wedge \Psi$ is the *overapproximate* of the reachable states, some spurious counterexamples may be included. We can strengthen invariants and refine the abstract system by our proposed refinement approaches. At first, we analyze the generated counterexamples Φ_{c1}, Φ_{c2}, and Φ_{c3} and check whether the counterexamples are spurious or not by our counterexample-guided invariant strengthening. The abstract states $\Phi_2 = \phi_{12} \wedge \phi_{42} \wedge \phi_{52}$ and $\Phi_3 = \phi_{21} \wedge \phi_{32} \wedge \phi_{52}$, which include Φ_{c2} and Φ_{c3}, are unreachable from the initial abstract states. Since Φ_2 and Φ_3 are unreachable from the initial abstract states, we can conclude that Φ_{c2} and Φ_{c3} are spurious counterexamples. However, we cannot make decisions whether Φ_{c1} is reachable from the initial states, although Φ_1 is reachable from the initial abstract states.

After the application of invariant strengthening on the abstract system, we eliminate Φ_{c2} and Φ_{c3} from counterexamples and strengthen invariants by the generated infeasible states. Now, we refine the abstract system and check whether Φ_{c1} is genuine counterexample or not by state partition technique. After the application of state partition technique on the abstract system, we get the refined abstract system presented in Figure 6. We split the state $\phi_{12} = l_1 \wedge t_1 \geq 1$ into $\phi_{121} = l_1 \wedge 1 \leq t_1 < 3600$ and $\phi_{122} = l_1 \wedge t_1 \geq 3600$; state $\phi_{32} = l_3 \wedge t_2 \geq 1$ into $\phi_{321} = l_3 \wedge 1 \leq t_2 < 3600$ and $\phi_{322} = l_3 \wedge t_2 \geq 3600$; state $\phi_{52} = l_5 \wedge 101 \leq \theta \leq 1000$ into $\phi_{521} = l_5 \wedge 101 \leq \theta < 1000$ and $\phi_{522} = l_5 \wedge \theta = 1000$. By the state partition technique, finally we find that Φ_{c1} is a counterexample of the system.

The example is implemented as BIP models and has been translated into timed automata using the tool BIP2UPPAAL [19]. To illustrate effectiveness of our approach, we verify models against property $\text{EF}\Phi_{c1}$, $\text{EF}\Phi_{c2}$, and $\text{EF}\Phi_{c3}$, which means "there exists a run that eventually reaches deadlock states Φ_{ci}." As a comparison, we first check both original system and abstract system. The checking results are shown in Table 2. As $\text{EF}\Phi_{c2}$ and $\text{EF}\Phi_{c3}$ are not satisfied by the abstract model, we conclude that Φ_{c2} and Φ_{c3} are spurious counterexamples immediately. Then we make partition refinement with Φ_{c2}. During state partition refinement, states $P12$, $P52$, and $P32$ (refer to ϕ_{12}, ϕ_{52}, and ϕ_{32}, resp.) have been partitioned. We verified refined abstract models against property. As $\text{EF}\Phi_{c1}$ is ultimately satisfied and we cannot use Φ_{c1} to partition models further, we conclude that Φ_{c1} is a genuine counterexample.

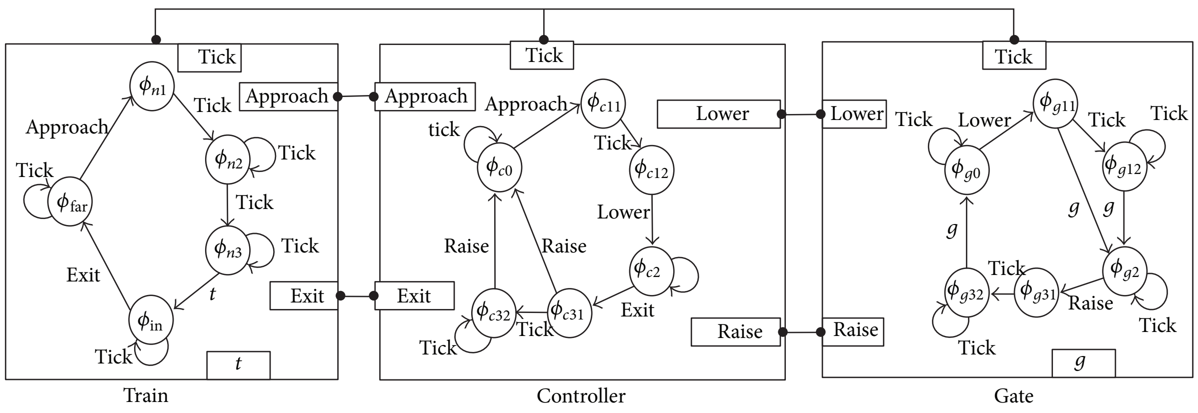


FIGURE 3: The abstraction for the TGC.

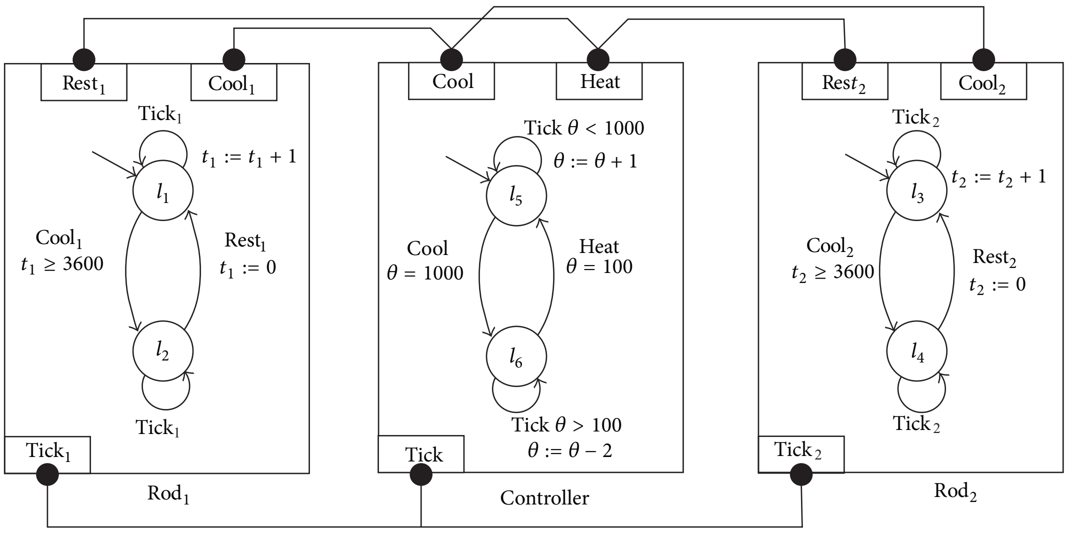


FIGURE 4: The component-based model for temperature control system.

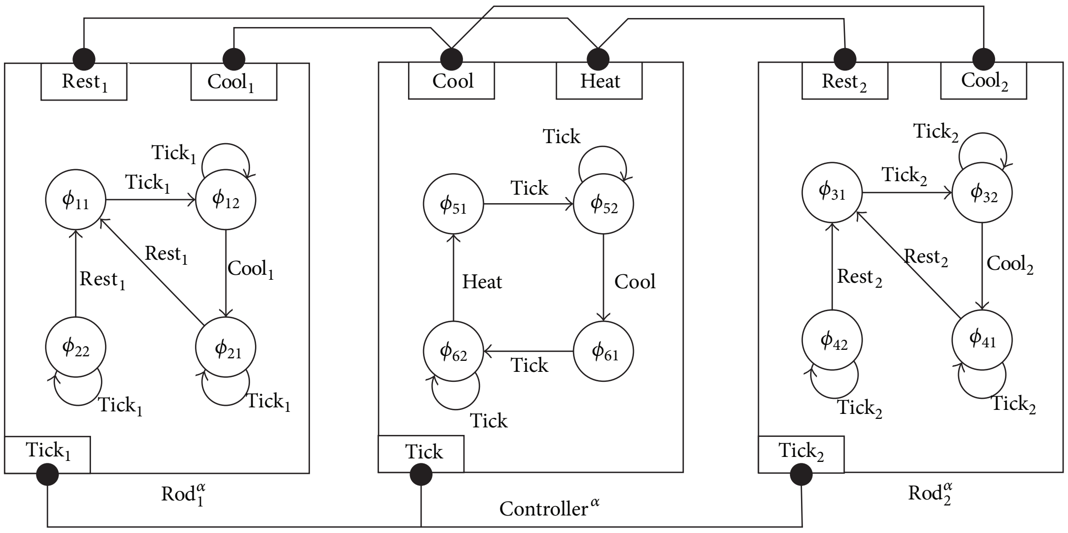


FIGURE 5: The abstraction for the component-based model.

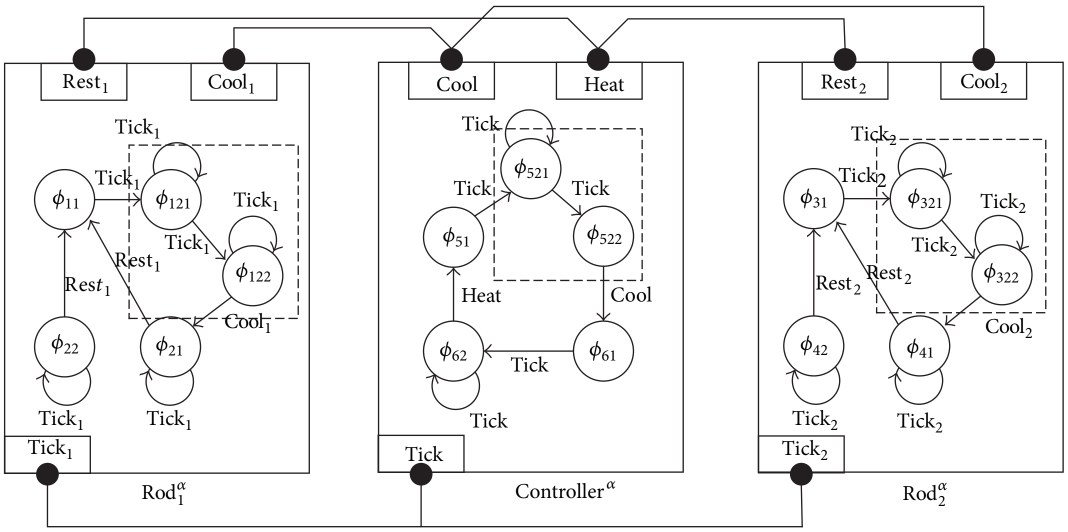


Figure 6: The refined abstract model.

Table 2: Simple table.

	Original	Abstract	Refined
Property_1		EFΦ_{c1}	
Result	True	True	True
Time (s)	0.046	0.001	0.001
Memory (KB)	8,056	7,084	7,084
Property_2		EFΦ_{c2}	
Result	False	False	False
Time (s)	0.044	0.001	0.001
Memory (KB)	8,060	7,088	7,072
Property_3		EFΦ_{c3}	
Result	False	False	False
Time (s)	0.045	0.001	0.001
Memory (KB)	8,064	7,084	7,072

From the above analysis, we have shown the effectiveness of our proposed verification framework. Using our proposed refinement techniques, we reconsidered the counterexamples of the system. We finally identify which are spurious counterexamples and refine the abstract systems.

7. Conclusions

We have proposed a unified framework with iterative refinements for compositional verification of component-based systems. This framework extends invariant-based compositional verification rule with the counterexample-guided invariant strength and state partition techniques. The former removes the spurious counterexample and uses the fixed point generated backward from the spurious to strengthen the system invariant. The latter partitions the abstract states according to the rest counterexamples to refine the abstract system. Both contribute to the unified verification framework which is proved to be sound and complete.

Compared with the invariant-based compositional verification which is incomplete, our verification framework with the iterative refinements can get more precise results with the balance of the verification complexity.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by the Fund NSFC61171121 and the Science Foundation of Chinese Ministry of Education-China Mobile 2012.

References

[1] K. M. Chandy, *Parallel Program Design*, Springer, 1989.

[2] S. Bensalem, Y. Lakhnech, and S. Owre, "Computing abstractions of infinite state systems compositionally and automatically," in *Computer Aided Verification*, pp. 319–331, Springer, 1998.

[3] S. Chaki, J. Ouaknine, K. Yorav, and E. Clarke, "Automated compositional abstraction refinement for concurrent C programs: a two-level approach," *Electronic Notes in Theoretical Computer Science*, vol. 89, no. 3, pp. 417–432, 2003.

[4] C. Flanagan and S. Qadeer, "Thread-modular model checking," in *Model Checking Software*, pp. 213–224, Springer, 2003.

[5] A. Malkis, A. Podelski, and A. Rybalchenko, "Thread-modular counterexample-guided abstraction refinement," in *Static Analysis*, pp. 356–372, Springer, 2011.

[6] T. A. Henzinger, R. Jhala, R. Majumdar, and S. Qadeer, "Thread-modular abstraction refinement," in *Computer Aided Verification*, pp. 262–274, Springer, Berlin, 2003.

[7] T. A. Henzinger, R. Jhala, and R. Majumdar, "Race checking by context inference," *ACM SIGPLAN Notices*, vol. 39, no. 6, pp. 1–13, 2004.

[8] S. Bensalem, Y. Lakhnech, and S. Owre, "Invest: a tool for the verification of invariants," in *Computer Aided Verification*, pp. 505–510, Springer, 1998.

[9] A. Pnueli, S. Ruah, and L. Zuck, "Automatic deductive verification with invisible invariants," in *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 82–97, Springer, 2001.

[10] T. Arons, A. Pnueli, S. Ruah, Y. Xu, and L. Zuck, "Parameterized verification with automatically computed inductive assertions," in *Computer Aided Verification*, pp. 221–234, Springer, Berlin, 2001.

[11] S. Bensalem, M. Bozga, T.-H. Nguyen et al., "Compositional verification for component-based systems and application," in *Proceedings of the Automated Technology for Verification and Analysis 6th International Symposium (ATVA '08)*, vol. 5311, pp. 64–79, Seoul, Republic of Korea, 2008.

[12] S. Bensalem, M. Bozga, T.-H. Nguyen, and J. Sifakis, "Compositional verification for component-based systems and application," *IET Software*, vol. 4, no. 3, pp. 181–193, 2010.

[13] M. Bozga, M. Jaber, and J. Sifakis, "Source-to-source architecture transformation for performance optimization in BIP," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 4, pp. 708–718, 2010.

[14] S. Bensalem and Y. Lakhnech, "Automatic generation of invariants," *Formal Methods in System Design*, vol. 15, no. 1, pp. 75–92, 1999.

[15] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice Hall, Englewood Cliffs, NJ, USA, 1981.

[16] K. Barkaoui and M. Minoux, "A polynomial-time graph algorithm to decide liveness of some basic classes of bounded Petri nets," in *Application and Theory of Petri Nets*, vol. 616, pp. 62–75, Springer, Berlin, 1992.

[17] A. Tiwari, H. Rueß, H. Saïdi, and N. Shankar, "A technique for invariant generation," in *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 113–127, Springer, 2001.

[18] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.

[19] C. Su, M. Zhou, L. Yin, H. Wan, and M. Gu, "Modeling and verification of component-based systems with data passing using bip," in *Proceedings of the 18th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS '13)*, pp. 4–13, 2013.