11-2015

Interpolation guided compositional verification

Shang-Wei LIN

Jun SUN
Singapore Management University, junsun@smu.edu.sg

Truong Khanh NGUYEN

Yang LIU

Jin Song DONG

2015 30th IEEE/ACM International Conference on Automated Software Engineering

Interpolation Guided Compositional Verification

Shang-Wei Lin*, Jun Sun†, Truong Khanh Nguyen†, Yang Liu*, and Jin Song Dong‡

*School of Computer Engineering, Nanyang Technological University
†Singapore University of Technology and Design
‡School of Computing, National University of Singapore

***Abstract*—Model checking suffers from the state space explosion problem. Compositional verification techniques such as assume-guarantee reasoning (AGR) have been proposed to alleviate the problem. However, there are at least three challenges in applying AGR. Firstly, given a system $M_1 \parallel M_2$, how do we automatically construct and refine (in the presence of spurious counterexamples) an assumption $\mathcal{A}_2$, which must be an abstraction of M_2? Previous approaches suggest to incrementally learn and modify the assumption through multiple invocations of a model checker, which could be often time consuming. Secondly, how do we keep the state space small when checking $M_1 \parallel \mathcal{A}_2 \models \varphi$ if multiple refinements of $\mathcal{A}_2$ are necessary? Lastly, in the presence of multiple parallel components, how do we partition the components? In this work, we propose interpolation-guided compositional verification. The idea is to tackle three challenges by using interpolations to generate and refine the abstraction of M_2, to abstract M_1 at the same time (so that the state space is reduced even if $\mathcal{A}_2$ is refined all the way to M_2), and to find good partitions. Experimental results show that the proposed approach outperforms existing approaches consistently.**

***Keywords*—model checking; automatic compositional verification; satisfiability; interpolation;**

I. Introduction

Model checking [14], [35] is a successful formal verification technique, which can automatically check whether a system model M satisfies a property φ, denoted by $M \models \varphi$. However, it suffers from the infamous *state space explosion* problem [14], [35]. To alleviate the problem, *assume-guarantee reasoning* (AGR) [20], [15], [34], a well-known compositional technique, has been proposed and applied on model checking. The most common rule used in AGR is the following assume-guarantee non-circular (AG-NC) rule:

$$\frac{M_1 \parallel \mathcal{A}_2 \models \varphi \quad \text{and} \quad M_2 \preceq \mathcal{A}_2}{M_1 \parallel M_2 \models \varphi} \tag{1}$$

Given a system with two components modeled by M_1 and M_2 and a property φ, the AG-NC proof rule says that if M_1 can satisfy a property φ under an assumption $\mathcal{A}_2$ and $\mathcal{A}_2$ is an abstraction of M_2 (i.e., M_2 can be simulated by $\mathcal{A}_2$, denoted by $M_2 \preceq \mathcal{A}_2$ as formulated in Section III-A), then we can conclude that $M_1 \parallel M_2$ satisfies φ. However, the challenge of applying AGR is at least threefold. The first is how to automatically construct and refine (in the presence of spurious counterexamples) the assumption $\mathcal{A}_2$. In general, the assumption should be kept as small as possible, i.e., containing only sufficient details to prove or disprove $M_1 \parallel \mathcal{A}_2 \models \varphi$. Besides relying on human creativity to create $\mathcal{A}_2$ manually, there is a line of works on applying learning techniques (e.g., [19], [4], [12], [26], [32]) to learn the assumption. The idea is to construct a candidate assumption through learning and then verify that the candidate is indeed an abstraction of M_2. Otherwise, the assumption must be modified (sometimes multiple times) until it becomes an abstraction of M_2. Such a process requires multiple invocations of a model checker and therefore could be time consuming. Secondly, the worst case scenario for AGR is that every detail of M_2 is needed in order to prove or disprove $M_1 \parallel M_2 \models \varphi$ and thus $\mathcal{A}_2$ is refined all the way to M_2. As a result, all the effort on finding the assumptions and checking $M_1 \parallel \mathcal{A}_2 \models \varphi$, often multiple times, is wasted. The question is then: is it possible to make use of the intermediate checking results so as to keep the state space reduced even in the worst case scenario? The last challenge is: in the presence of multiple parallel components, how do we partition components to apply AGR? It has been reported in [17] that without a good partition strategy, model checking based on AGR might be even worse than the traditional monolithic model checking.

In this work, we propose an approach to complement existing AGR-based compositional verification techniques by tackling the three challenges above. Central to our approach is the idea of learning from bounded model checking (BMC) results. In the following, we briefly present our approach, and Fig. 1 shows its workflow. A model in our work is a parallel composition of multiple components, which communicate through shared variables[1]. At the beginning, the components are partitioned into two groups, either randomly or based on simple heuristics. Let us assume that the model is $G_1 \parallel G_2$ where G_i where $i \in \{1, 2\}$ itself is a parallel composition of multiple components. In our method, we change the partition based on intermediate verification results. In addition, we would construct not only an abstraction $\mathcal{A}_2$ for G_2 but also an abstraction $\mathcal{A}_1$ for G_1. Initially, we set the transition relation of $\mathcal{A}_2$ to be `TRUE`, which is the weakest over-approximation, and $\mathcal{A}_1$ to be G_1. We then model check $\mathcal{A}_1 \parallel \mathcal{A}_2$. If $\mathcal{A}_1 \parallel \mathcal{A}_2$ satisfies the property φ, we prove the system satisfies φ. Otherwise, we check whether the counterexample is spurious or not. This is done by bounded model checking $G_1 \parallel G_2$ up to the length of the counterexample. If the counterexample is not spurious, we find a counterexample. Otherwise, we obtain

The corresponding author, Shang-Wei Lin, can be contacted via the following e-mail address: shang-wei.lin@ntu.edu.sg.

This research is supported (in part) by the National Research Foundation, Prime Ministers Office, Singapore under its National Cybersecurity R&D Program (Award No. NRF2014NCR-NCR001-30) and administered by the National Cybersecurity R&D Directorate.

[1]Our work can be extended to support messaging or barrier synchronization.

978-1-5090-0025-8/15 $31.00 © 2015 IEEE
DOI 10.1109/ASE.2015.33

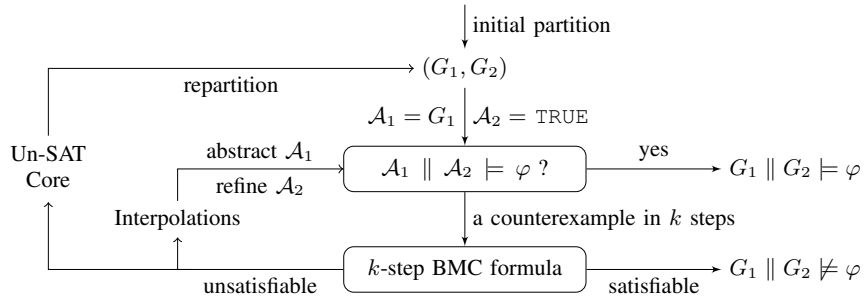

Fig. 1. Overall Flow

the unsatisfiability (unsat) core from the BMC formula. We re-partition the components such that those relevant to the unsat core are grouped into G_1 (since intuitively details of those components matter, at least in avoiding the spurious counterexample). If the partition (G_1, G_2) cannot be improved by unsat cores anymore, we refine $\mathcal{A}_2$ (the abstraction of the new G_2) based on the interpolants [22], [23] from the unsatisfiable BMC formula. Lastly, we use the interpolants to construct the abstraction $\mathcal{A}_1$ of G_1 (so as to avoid details of the processes which are irreverent at least to the proof of unsatisfiable BMC formula). The above process continues until a verification result can be concluded, i.e., the property is proved or a real counterexample is found.

Our interpolation-guided approach tackles the above-mentioned three problems as follows. Firstly, the assumptions are generated and refined automatically based on interpolations. Different from existing approaches on learning assumptions [19], [4], [12], [26], [32], the assumptions in our approach are abstractions of G_2 by construction. Secondly, unlike in existing AGR-based approaches where the component G_1 is never changed, we actively abstract the transition relation of G_1 based on interpolations. As a result, we would not explore $G_1 \parallel G_2$ even if $\mathcal{A}_2$ has to be refined all the way to G_2. Lastly, we use unsat cores to guide the partition of components. We have implemented the approach in the PAT model checker [38], and experiments show the benefits of our approach.

The rest of this paper is organized as follows. Section II illustrates our approach with a simple example. Section III reviews some preliminary backgrounds and recalls the transition over-approximation based on interpolants. In Section IV, we show how we construct and refine $\mathcal{A}_1$ and $\mathcal{A}_2$ by using interpolations. Experiment results are presented in Section V to show the effectiveness of our approach. Section VI summarizes related works. Section VII concludes this work.

II. A Simple Example

We illustrate how our approach works using a simple example. We first show abstracting M_1 whilst refining M_2 could be beneficial to a system with two components. Next, we generalize the system to n components and then show how a good partition is found. A two-bit counter is modeled by two components, $\texttt{cell}_1$ and $\texttt{cell}_2$ in Fig. 2. Each component $\texttt{cell}_i$ for $i \in \{1, 2\}$ has three Boolean variables as follows. The $\texttt{in}_i$ variable indicates whether the carry-in value of $\texttt{cell}_i$ is asserted. The $\texttt{bit}_i$ variable stores the current bit value of $\texttt{cell}_i$. It is initialized as FALSE, and its next value depends on the exclusive-or of its current value and its carry-in value. The $\texttt{out}_i$ indicates whether the current bit value of $\texttt{cell}_i$ should be carried out. If the bit value of $\texttt{cell}_1$ is carried out, then the carry-in value of $\texttt{cell}_2$ should be asserted. The initial condition I_i and transition relation T_i of the two components are encoded as follows, respectively.

- I_1: $\neg\texttt{bit}_1 \wedge \texttt{in}_1$
- I_2: $\neg\texttt{bit}_2$
- T_1: $(\texttt{bit}'_1 \Leftrightarrow \texttt{bit}_1 \oplus \texttt{in}_1) \wedge (\texttt{out}'_1 \Leftrightarrow \texttt{bit}_1 \wedge \texttt{in}_1)$
- T_2: $(\texttt{in}'_2 \Leftrightarrow \texttt{out}_1) \wedge (\texttt{bit}'_2 \Leftrightarrow \texttt{bit}_2 \oplus \texttt{in}_2) \wedge (\texttt{out}'_2 \Leftrightarrow \texttt{bit}_2 \wedge \texttt{in}_2)$

Suppose we want to verify the property φ requiring that $\texttt{out}_2$, $\texttt{bit}_2$, and $\texttt{in}_2$ do not hold simultaneously, i.e., G $\neg(\texttt{out}_2 \wedge \texttt{bit}_2 \wedge \texttt{in}_2)$. Let $\texttt{cell}_1$ be M_1 and $\texttt{cell}_2$ be M_2, respectively. We use $\hat{T}_i^l$ to denote the over-approximation of T_i after l-th iteration. Initially in our approach, $\hat{T}_2^0$ is set to the weakest transition relation $\top$, and $\hat{T}_1^0$ is kept as T_1. Let $\mathcal{A}_i^l$ be the component encoded by the initial condition I_i and the abstract transition relation $\hat{T}_i^l$.

In the first iteration, a counterexample is found in one step when model checking $\mathcal{A}_1^0 \parallel \mathcal{A}_2^0 \models \varphi$. To check whether there is any one-step counterexample in the concrete system, a bounded model checking (BMC) of length one based on T_1 and T_2 is performed. However, the BMC formula is not satisfiable meaning that the counterexample is spurious, and $\hat{T}_2^0$ should be strengthened. From the proof of unsatisfiability, we obtain the symmetric interpolant $\Omega_1 = \top$ for T_1 and $\Omega_2 = \texttt{bit}_2 \vee \neg\texttt{out}'_2$ for T_2, respectively (c.f. Section III-B for details). We use the obtained interpolant to weaken $\hat{T}_1^0$ and strengthen $\hat{T}_2^0$ as follows: $\hat{T}_1^1 = \Omega_1 = \top$ and $\hat{T}_2^1 = \hat{T}_2^0 \wedge \Omega_2 = (\texttt{bit}_2 \vee \neg\texttt{out}'_2)$. By the characteristics of interpolants, $\hat{T}_1^1$ and $\hat{T}_2^1$ are over-approximations of T_1 and T_2, respectively. In addition, $I_1 \wedge \hat{T}_1^1 \wedge I_2 \wedge \hat{T}_2^1$ does not admit any one-step counterexamples.

In the second iteration, $\mathcal{A}_1^1 \parallel \mathcal{A}_2^1 \models \varphi$ are verified again, and a counterexample in three steps is found. To check the feasibility of any three-step counterexamples, a BMC of length three based on the concrete transition relations, T_1 and T_2, is performed. However, the BMC formula is not satisfiable, and $\hat{T}_2^1$ still needs to be strengthened. We obtain the interpolants Ω'_1 and Ω'_2 from the unsatisfiability proof to refine $\hat{T}_1^1$ and $\hat{T}_2^1$,

```
MODULE cell1
    var bit1: bool;
    var in1: bool;
    var out1: bool;

    init(bit1) := FALSE;
    init(in1) := TRUE;

    next(bit1) := bit1 xor in1;
    next(out1) := bit1 & in1;
END MODULE
```

```
MODULE cell2
    var bit2: bool;
    var in2: bool;
    var out2: bool;

    init(bit2) := FALSE;

    next(in2) := out1;
    next(bit2) := bit2 xor in2;
    next(out2) := bit2 & in2;
END MODULE
```

Fig. 2. The Counter Example

respectively, as follows: $\hat{T}_1^2 = \Omega_1' = \texttt{bit}_1 \vee \neg \texttt{out}_1'$ and $\hat{T}_2^2 = \hat{T}_2^1 \wedge \Omega_2' = (\neg \texttt{out}_2' \wedge \texttt{out}_1) \vee (\neg \texttt{out}_2' \wedge \neg \texttt{in}_2') \vee (\texttt{in}_2 \wedge \texttt{bit}_2)$.

In the third iteration, a spurious counterexample in seven steps is found, and $\hat{T}_1^2$ and $\hat{T}_2^2$ are strengthen by interpolants as follows: $\hat{T}_1^3 = \texttt{bit}_1 \vee \neg \texttt{out}_1'$ and $\hat{T}_2^3 = \hat{T}_2^2 \wedge (\texttt{bit}_2' \vee \texttt{out}_2')$. In the fourth iteration, $\mathcal{A}_1^3 \parallel \mathcal{A}_2^3 \models \varphi$ is verified by model checking again, but no counterexamples are found this time meaning that $\texttt{cell}_1 \parallel \texttt{cell}_2 \models \varphi$. We remark here that abstracting T_1 is optional, but doing so reduces the state explosion problem when checking $M_1 \parallel \mathcal{A}_2 \models \varphi$.

Let us do the verification again, but this time let $\texttt{cell}_2$ be M_1 instead of $\texttt{cell}_1$. The verification can be done in one iteration, where $\mathcal{A}_1 = \texttt{cell}_2$ and $\mathcal{A}_2$ with the weakest transition relation TRUE. This is because $\texttt{cell}_2$ is sufficient to prove the property. From this example, we can observe the importance of partitioning components for AGR. In our approach, we utilize the unsatisfiability core to predict the components which are necessary to prove the property. Within each iteration, if the BMC formula for checking the spuriousness of counterexamples is unsatisfiable, we obtain its unsatisfiability core. Any component whose variables are appearing in the unsatisfiability core might be necessary for proving the property and is included into the M_1 group. Once the M_1 group is changed, the verification is restart for the new partition in the next iteration.

For the same counter example, if we have n cells (n-bit counter) and suppose we want to verify the property φ_j: G $\neg(\texttt{out}_j \wedge \texttt{bit}_j \wedge \texttt{in}_j)$ for $j \in \{1, 2, \ldots, n\}$, our approach is able to detect that $\texttt{cell}_j$ is the only necessary component to prove φ_j, i.e., $\texttt{cell}_j$ is in the M_1 group and the rest are in the M_2 group, which is the best partition (only one iteration is required for verifying φ_j).

III. Background

In Section III-A, we review some definitions, borrowed from [12], [23], of symbolic model checking and bounded model checking. Then, we briefly recall the transition approximation based on interpolations [22], [23], in Section III-B.

A. Preliminaries

Define $\mathbb{B} = \{\top, \bot\}$ to be the Boolean domain where $\top$ and $\bot$ denote the truth values TRUE and FALSE, respectively. Let $\mathbf{x}$ be a set of Boolean variables and $|\mathbf{x}|$ the size of $\mathbf{x}$. A Boolean formula $\phi(\mathbf{x})$ over $\mathbf{x}$ is a function from $\mathbb{B}^{|\mathbf{x}|}$ to $\mathbb{B}$. A valuation $\nu : \mathbf{x} \to \mathbb{B}$ over $\mathbf{x}$ is a function from Boolean variables to truth values. We use $\phi[\nu]$ to denote the result of evaluating ϕ by replacing each $x \in \mathbf{x}$ with $\nu(\mathbf{x})$. To represent transition systems symbolically, we also define a set of Boolean variables $\mathbf{x}' = \{x' \mid x \in \mathbf{x}\}$, which corresponds to $\mathbf{x}$ such that $x \in \mathbf{x}$ represents the current value of x, while $x' \in \mathbf{x}'$ represents the value of x in the next state. Moreover, let $\phi(\mathbf{x}, \mathbf{x}')$ be a Boolean formula over $\mathbf{x}$ and $\mathbf{x}'$. If ν and ν' are valuations over $\mathbf{x}$ and $\mathbf{x}'$, respectively, we use $\phi[\nu, \nu']$ to denote the result of evaluating ϕ by replacing each $x \in \mathbf{x}$ with $\nu(x)$ and replacing each $x' \in \mathbf{x}'$ with $\nu'(x')$. Let $\mathcal{C}$ be a set of formulas. We use $\bigwedge \mathcal{C}$ to denote the conjunction of all formulas.

A transition system $M = (\mathbf{x}, I(\mathbf{x}), T(\mathbf{x}, \mathbf{x}'))$ consists of its state variables $\mathbf{x}$, its initial predicate $I(\mathbf{x})$, and its transition relation $T(\mathbf{x}, \mathbf{x}')$. We sometimes write $(\mathbf{x}, I, T)$ to denote a transition system if there is no risk of confusion. A trace of M is a finite sequence of valuations $\sigma = \nu^0 \nu^1 \cdots \nu^k$, where ν_i is a valuation over $\mathbf{x}$, such that $I(\nu^0) = \top$ and $T(\nu^i, \nu^{i+1}) = \top$ for all $i \in \{0, 1, \cdots, k\}$. The language of M, denoted by $\mathcal{L}(M)$, contains all the traces of M. A state predicate $\varphi(\mathbf{x})$ is a Boolean function over $\mathbf{x}$. We say M satisfies φ, denoted by $M \models \varphi$, if for each $\sigma = \nu^0 \nu^1 \cdots \nu^k \in \mathcal{L}(M)$, we have $\varphi[\nu^i] = \top$ for all $i \in \{0, 1, \ldots, k\}$. A counterexample of $M \models \varphi$ is a trace $\nu^0 \nu^1 \cdots \nu^t$ of M such that $\varphi[\nu^i] = \top$ for all $i \in \{0, 1, \ldots, t-1\}$ but $\varphi[\nu^t] = \bot$.

Let $M = (\mathbf{x}, I(\mathbf{x}), T(\mathbf{x}, \mathbf{x}'))$ and $\mathcal{A} = (\mathbf{x}, I_{\mathcal{A}}(\mathbf{x}), T_{\mathcal{A}}(\mathbf{x}, \mathbf{x}'))$ be two transitions systems over $\mathbf{x}$. We say M is simulated by $\mathcal{A}$ or $\mathcal{A}$ simulates M, denoted by $M \preceq \mathcal{A}$, if $\forall \mathbf{x} \,.\, I(\mathbf{x}) \implies I_{\mathcal{A}}(\mathbf{x})$ and $\forall \mathbf{x}\mathbf{x}' \,.\, T(\mathbf{x}, \mathbf{x}') \implies T_{\mathcal{A}}(\mathbf{x}, \mathbf{x}')$. That is, the initial condition of M is stronger than that of $\mathcal{A}$ and every transition in M is also allowed in $\mathcal{A}$. Obviously, if $M \preceq \mathcal{A}$ holds, then $\mathcal{L}(M) \subseteq \mathcal{L}(\mathcal{A})$ holds. Let $M_i = (\mathbf{x}_i, I_i(\mathbf{x}_i), T_i(\mathbf{x}_i, \mathbf{x}_i'))$ be two transition systems for $i \in \{1, 2\}$. The *parallel composition* of M_1 and M_2 is the transition system $M_1 \parallel M_2 = (\mathbf{x}_1 \cup \mathbf{x}_2, I_1(\mathbf{x}_1) \wedge I_2(\mathbf{x}_2), T_1(\mathbf{x}_1, \mathbf{x}_1') \wedge T_2(\mathbf{x}_2, \mathbf{x}_2'))$.

Given a transition system $M = (\mathbf{x}, I(\mathbf{x}), T(\mathbf{x}, \mathbf{x}'))$ and a state predicate $\varphi(\mathbf{x})$, whether φ is k-reachable in M can be expressed symbolically as a Boolean formula. For each variable $x \in \mathbf{x}$ and a natural number i, we use $x^{\langle i \rangle}$ to denote the variable x with i primes added, which represents the value of x at time i. For example, $x^{\langle 3 \rangle} = x'''$ represents the value of x at time 3. We also extend this notation to the set of variables and formulas. Thus, $\mathbf{x}^{\langle i \rangle}$ contains variables with i primes added, $\phi(\mathbf{x})^{\langle i \rangle}$ is the formula over $\mathbf{x}^{\langle i \rangle}$, and $\phi(\mathbf{x}, \mathbf{x}')^{\langle i \rangle}$ is the formula over $\mathbf{x}^{\langle i \rangle}$ and $\mathbf{x}^{\langle i+1 \rangle}$. A state predicate $\varphi(\mathbf{x})$ is k-reachable in $(\mathbf{x}, I(\mathbf{x}), T(\mathbf{x}, \mathbf{x}'))$ if the following bounded model checking (BMC) formula is satisfiable.

$$I(\mathbf{x})^{\langle 0 \rangle} \wedge T(\mathbf{x}, \mathbf{x}')^{\langle 0 \rangle} \wedge T(\mathbf{x}, \mathbf{x}')^{\langle 1 \rangle} \wedge \cdots \wedge T(\mathbf{x}, \mathbf{x}')^{\langle k-1 \rangle} \wedge \varphi(\mathbf{x})^{\langle k \rangle}$$

Algorithm 1: Verification by Transition Approximation

input : $(\mathbf{x}, I, T)$: the concrete transition system;
φ: the property to be checked
output: yes/no, with a counterexample

1 $\hat{T} \longleftarrow \top$;
2 **while** *True* **do**
3 **if** $(\mathbf{x}, I, \hat{T}) \models \varphi$ **then**
4 **return** *yes*
5 **else**
6 Suppose $\neg\varphi$ is k-reachable in $(\mathbf{x}, I, \hat{T})$;
7 $\Theta \longleftarrow \{I^{\langle 0\rangle}, T^{\langle 0\rangle}, \ldots, T^{\langle k-1\rangle}, \neg\varphi^{\langle k\rangle}\}$;
8 **if** $\bigwedge\Theta$ *is satisfied by a valuation* ν **then**
9 **return** (no, ν) ;
10 **else**
11 Let $\hat{\Theta} = \{\hat{I}^{\langle 0\rangle}, \hat{T}^{\langle 0\rangle}, \ldots, \hat{T}^{\langle k-1\rangle}, \neg\hat{\varphi}^{\langle k\rangle}\}$ be the symmetric interpolant for Θ ;
12 $\hat{T} \longleftarrow \hat{T} \wedge \bigwedge_{i=0}^{k-1} (\hat{T}^{\langle i\rangle})^{\langle -i\rangle}$;

B. Interpolation-based Approximation of Transition Relations

In [22], [23], transition relations are approximated by *interpolations* [18], as formulated in Definition 1, obtained from unsatisfiability proofs of bounded model checking.

Definition 1: Given a pair of Boolean formulas (A, B) such that $A \wedge B$ is unsatisfiable, an *interpolant* for (A, B) is a formula $\hat{A}$ satisfying the following properties:

1) A implies $\hat{A}$, i.e., $A \implies \hat{A}$
2) $\hat{A} \wedge B$ is unsatisfiable
3) $\hat{A}$ refers only to the common variables of A and B.

If $A \wedge B$ is unsatisfiable with an unsatisfiability proof, an interpolant for (A, B) can be obtained from the proof [31]. In a formula of a k-step bounded model checking problem, if the formula is unsatisfiable, the over-approximation of the transition relation can be obtained from the *symmetric interpolants* [22], [23], as formulated in Definition 2, among the transition relations from steps 0 to $k-1$.

Definition 2: Given an indexed set of Boolean formulas $A = \{a_1, a_2, \ldots, a_n\}$ such that $\bigwedge A$ is inconsistent, a *symmetric interpolant* for A is an indexed set of Boolean formulas $\hat{A} = \{\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_n\}$ satisfying the following conditions:

1) $a_i \implies \hat{a}_i$ for all $i \in \{1, 2, \ldots, n\}$
2) $\bigwedge \hat{A}$ is inconsistent
3) $\hat{a}_i$ refers to the variables common to a_i and $A \setminus \{a_i\}$.

Algorithm 1 shows a verification approach by over-approximating the transition relation based on symmetric interpolants [22], [23]. The details are as follows:

- Initially, the approximation $\hat{T}$ is initialized as $\top$ (line 1).
- If $(\mathbf{x}, I, \hat{T}) \models \varphi$ holds, we can conclude $(\mathbf{x}, I, T) \models \varphi$ also holds because $T \implies \hat{T}$ (lines 3–4).
- If $\neg\varphi$ is k-reachable in $(\mathbf{x}, I, \hat{T})$, there could be two cases where either $\neg\varphi$ is also k-reachable in $(\mathbf{x}, I, T)$, or $\hat{T}$ is too weak an approximation. Bounded model checking can help to find out which case it is. We construct a set of formulas $\Theta = \{I^{\langle 0\rangle}, T^{\langle 0\rangle}, T^{\langle 1\rangle}, \ldots, T^{\langle k-1\rangle}, \neg\varphi^{\langle k\rangle}\}$ where $\bigwedge\Theta$ is exactly the BMC formula. We use a decision procedure to determine the satisfiability. If $\bigwedge\Theta$ is satisfiable, then $(\mathbf{x}, I, T) \models \varphi$ does not hold (lines 8–9). If $\bigwedge\Theta$ is not satisfiable, then $\hat{T}$ is too weak and needs to be refined. Let $\hat{\Theta} = \{\hat{I}^{\langle 0\rangle}, \hat{T}^{\langle 0\rangle}, \hat{T}^{\langle 1\rangle}, \ldots, \hat{T}^{\langle k-1\rangle}, \neg\hat{\varphi}^{\langle k\rangle}\}$ be the symmetric interpolant for Θ. Let us define $\hat{T}_{[\![i]\!]} = (\hat{T}^{\langle i\rangle})^{\langle -i\rangle}$ where $(\hat{T}^{\langle i\rangle})^{\langle -i\rangle}$ denotes the formula obtained by removing i primes from $\hat{T}^{\langle i\rangle}$ if possible. Because of the properties of symmetric interpolants, the formula

$$I^{\langle 0\rangle} \wedge \hat{T}^{\langle 0\rangle}_{[\![0]\!]} \wedge \hat{T}^{\langle 1\rangle}_{[\![1]\!]} \wedge \cdots \wedge \hat{T}^{\langle k-1\rangle}_{[\![k-1]\!]} \wedge \neg\varphi^{\langle k\rangle}$$

is unsatisfiable, i.e., $\bigwedge_{i=0}^{k-1} \hat{T}_{[\![i]\!]}$ admits no path in k steps from I to $\neg\varphi$. Thus, $\hat{T}$ is refined as $\hat{T} \wedge \bigwedge_{i=0}^{k-1} \hat{T}_{[\![i]\!]}$, which becomes the new approximation in the next iteration for verification (lines 11–12).

The process continues until a verification result can be concluded. The correctness and termination of Algorithm 1 are proved in [22], [23].

IV. Improving Compositional Verification by Interpolations

In this section, we introduce how the compositional verification based on assume-guarantee reasoning (AGR), can be improved using interpolations. We first show how our approach works for systems with two processes in Section IV-A. Next, we show how to extend our approach to systems with many processes in Section IV-B.

A. Generating Assumptions by Interpolations

Let us recall the AG-NC proof rule in Equation 1. To automatically generate the assumption $\mathcal{A}_2$, we can construct $\mathcal{A}_2$ as the symmetric interpolants of M_2 from the bounded model checking problem of $M_1 \parallel M_2 \models \varphi$. Since the transition relation of $\mathcal{A}_2$ is an over-approximation of that of M_2, the second condition of Equation 1, $M_2 \preceq \mathcal{A}_2$, holds naturally. We only have to check whether the first condition, $M_1 \parallel \mathcal{A}_2 \models \varphi$, holds or not. If it does, then we have a conclusive result showing that $M_1 \parallel M_2 \models \varphi$. If it does not hold with a counterexample in k steps, the transition relation of the assumption $\mathcal{A}_2$ is refined (strengthened) by the interpolants obtained from the k-step bounded model checking problem of $M_1 \parallel M_2 \models \varphi$ provided that the problem is unsatisfiable. Furthermore, applying the AGR rule twice, it is easy to see that the following rule holds.

$$\frac{\mathcal{A}_1 \parallel \mathcal{A}_2 \models \varphi \quad \text{and} \quad M_2 \preceq \mathcal{A}_2 \quad \text{and} \quad M_1 \preceq \mathcal{A}_1}{M_1 \parallel M_2 \models \varphi} \tag{2}$$

Thus, using the same formula for bounded model checking of $M_1 \parallel M_2$, we can obtain the symmetric interpolant of the

Algorithm 2: Compositional Verification based on Interpolation

input : $M_1 = (\mathbf{x}_1, I_1, T_1)$ and $M_2 = (\mathbf{x}_2, I_2, T_2)$: concrete transition systems; φ: the property to be checked
output: yes/no, with a counterexample

1 $\hat{T}_1 \longleftarrow T_1$;
2 $\hat{T}_2 \longleftarrow \top$;
3 **while** *True* **do**
4 **if** $(\mathbf{x}_1, I_1, \hat{T}_1) \parallel (\mathbf{x}_2, I_2, \hat{T}_2) \models \varphi$ **then**
5 **return** *yes*
6 **else**
7 Suppose $\neg\varphi$ is k-reachable in $(\mathbf{x}_1, I_1, \hat{T}_1) \parallel (\mathbf{x}_2, I_2, \hat{T}_2)$;
8 $\Theta \longleftarrow \{I_1^{\langle 0\rangle}, I_2^{\langle 0\rangle}, T_1^{\langle 0\rangle}, T_2^{\langle 0\rangle}, T_1^{\langle 1\rangle}, T_2^{\langle 1\rangle}, \ldots, T_1^{\langle k-1\rangle}, T_2^{\langle k-1\rangle}, \neg\varphi^{\langle k\rangle}\}$;
9 **if** $\bigwedge\Theta$ *is satisfied by a valuation* ν **then**
10 **return** (no, ν)
11 **else**
12 Let $\hat{\Theta} = \{\hat{I}_1^{\langle 0\rangle}, \hat{I}_2^{\langle 0\rangle}, \hat{T}_1^{\langle 0\rangle}, \hat{T}_2^{\langle 0\rangle}, \hat{T}_1^{\langle 1\rangle}, \hat{T}_2^{\langle 1\rangle}, \ldots, \hat{T}_1^{\langle k-1\rangle}, \hat{T}_2^{\langle k-1\rangle}, \hat{\neg\varphi}^{\langle k\rangle}\}$ be the symmetric interpolant for Θ ;
13 $\hat{T}_2 \longleftarrow \hat{T}_2 \wedge \bigwedge_{i=0}^{k-1} (\hat{T}_2^{\langle i\rangle})^{\langle -i\rangle}$;
14 $\hat{T}_1 \longleftarrow \bigwedge_{i=0}^{k-1} (\hat{T}_1^{\langle i\rangle})^{\langle -i\rangle}$; // Abstracting M_1 (optional)

transition relation for M_1 as well, which gives $\mathcal{A}_1$, i.e., the abstraction of M_1.

Algorithm 2 shows the pseudo-code of the proposed automatic compositional verification approach based on interpolations (with $\mathcal{A}_2$ strengthened and $\mathcal{A}_1$ abstracted simultaneously). The details are described as follows.

Initially, the approximation $\hat{T}_1$ is initialized as T_1, and the approximation $\hat{T}_2$ is initialized as $\top$, respectively (lines 1–2). If $(\mathbf{x}_1, I_1, \hat{T}_1) \parallel (\mathbf{x}_2, I_2, \hat{T}_2) \models \varphi$ holds, then we can conclude that $(\mathbf{x}_1, I_1, T_1) \parallel (\mathbf{x}_2, I_2, T_2) \models \varphi$ also holds (lines 4–5) because $\hat{T}_1$ and $\hat{T}_2$ are over-approximations of T_1 and T_2, respectively. Note that both $T_1 \implies \hat{T}_1$ and $T_2 \implies \hat{T}_2$ hold according to the properties of interpolations (cf. Definition 2). If $\neg\varphi$ is k-reachable in $(\mathbf{x}_1, I_1, \hat{T}_1) \parallel (\mathbf{x}_2, I_2, \hat{T}_2)$, there could be two cases: (1) $\neg\varphi$ is also k-reachable in $(\mathbf{x}_1, I_1, T_1) \parallel (\mathbf{x}_2, I_2, T_2)$, or (2) $\hat{T}_2$ is too weak an approximation. We construct a set of formulas

$$\Theta = \{I_1^{\langle 0\rangle}, I_2^{\langle 0\rangle}, T_1^{\langle 0\rangle}, T_2^{\langle 0\rangle}, \ldots, T_1^{\langle k-1\rangle}, T_2^{\langle k-1\rangle}, \neg\varphi^{\langle k\rangle}\}$$

where $\bigwedge\Theta$ is exactly the bounded model checking formula. We use a decision procedure to determine its satisfiability.

- If $\bigwedge\Theta$ is satisfiable, we can conclude that $(\mathbf{x}_1, I_1, T_1) \parallel (\mathbf{x}_2, I_2, T_2)$ violates the property φ because there exists a real counterexample in k steps (lines 9–10).
- If $\bigwedge\Theta$ is unsatisfiable, then $\hat{T}_2$ is too weak and needs to be strengthened, which can be done as follows. Let

  $$\hat{\Theta} = \{\hat{I}_1^{\langle 0\rangle}, \hat{I}_2^{\langle 0\rangle}, \hat{T}_1^{\langle 0\rangle}, \hat{T}_2^{\langle 0\rangle}, \ldots, \hat{T}_1^{\langle k-1\rangle}, \hat{T}_2^{\langle k-1\rangle}, \hat{\neg\varphi}^{\langle k\rangle}\}$$

  be the symmetric interpolant for Θ. Let us define $\hat{T}_{[\![i,j]\!]} = (\hat{T}_j^{\langle i\rangle})^{\langle -i\rangle}$ for $i \in \{0, 1, \ldots, k-1\}$ and $j \in \{1, 2\}$ where $(\hat{T}_j^{\langle i\rangle})^{\langle -i\rangle}$ denotes the formula obtained by removing i primes from $\hat{T}_j^{\langle i\rangle}$ if possible. Because of the properties of symmetric interpolants, the following bounded model checking formula

  $$I_1^{\langle 0\rangle} \wedge I_2^{\langle 0\rangle} \wedge \bigwedge_{i=0}^{k-1} \hat{T}_{[\![i,1]\!]}^{\langle i\rangle} \wedge \bigwedge_{i=0}^{k-1} \hat{T}_{[\![i,2]\!]}^{\langle i\rangle} \wedge \neg\varphi^{\langle k\rangle}$$

  is unsatisfiable. That is to say, $\bigwedge_{i=0}^{k-1} \hat{T}_{[\![i,1]\!]}$ and $\bigwedge_{i=0}^{k-1} \hat{T}_{[\![i,2]\!]}$ admit no path in k steps from $I_1 \wedge I_2$ to violate φ. Note that $\bigwedge_{i=0}^{k-1} \hat{T}_{[\![i,2]\!]}$ is an over-approximation of T_2 as well as a refinement of $\hat{T}_2$. Thus, we strengthen $\hat{T}_2$ as $\hat{T}_2 \wedge \bigwedge_{i=0}^{k-1} \hat{T}_{[\![i,2]\!]}$ for the next iteration (line 13). In addition, since $\bigwedge_{i=0}^{k-1} \hat{T}_{[\![i,1]\!]}$ is an over-approximation of T_1, we can optionally abstract $\hat{T}_1$ as $\bigwedge_{i=0}^{k-1} \hat{T}_{[\![i,1]\!]}$ in line 14, which alleviates the state space explosion problem when checking whether $(\mathbf{x}_1, I_1, \hat{T}_1) \parallel (\mathbf{x}_2, I_2, \hat{T}_2) \models \varphi$ holds.

Theorems 1 and 2 prove the correctness and termination of the proposed interpolation-based approach.

Theorem 1: Algorithm 2 is correct.

Proof: To establish the correctness of Algorithm 2, we want to prove that it returns "yes" only if $M_1 \parallel M_2 \models \varphi$, and returns "no" with a counterexample only if $M_1 \parallel M_2 \not\models \varphi$. Let $\hat{M}_1 = (\mathbf{x}_1, I_1, \hat{T}_1)$ and $\hat{M}_2 = (\mathbf{x}_2, I_2, \hat{T}_2)$ be the transition systems with respect to $\hat{T}_1$ and $\hat{T}_2$, respectively. Since $\hat{T}_1$ and $\hat{T}_2$ are obtained by interpolations, both $T_1 \implies \hat{T}_1$ and $T_2 \implies \hat{T}_2$ hold, i.e., $\hat{M}_1$ and $\hat{M}_2$ are the abstractions of M_1 and M_2, respectively. Algorithm 2 returns "yes" only when $\hat{M}_1 \parallel \hat{M}_2 \models \varphi$, which implies $M_1 \parallel M_2 \models \varphi$. On the other hand, Algorithm 2 returns "no" only when $\bigwedge\Theta$ is satisfiable by a valuation ν. Since $\bigwedge\Theta$ is a bounded model checking formula to check whether $\neg\varphi$ is reachable within k-steps in $M_1 \parallel M_2$, the valuation ν is a witness of $M_1 \parallel M_2 \not\models \varphi$. From the above arguments, we can conclude that Algorithm 2 is correct. ■

Algorithm 3: PARTITION

input : $\{C_1, C_2, \ldots, C_n\}$: a set of components;
k: the number of steps
output: (M_1, M_2): the partition of all components

1 $M_1 \longleftarrow M_2 \longleftarrow \emptyset$;
2 Let U_Ψ be the unsatisfiability core of Ψ;
3 **for** $j = 1$ *to* n **do**
4 **if** C_j *has any variable appearing in* U_Ψ **then**
5 $M_1 \longleftarrow M_1 \cup \{C_j\}$;
6 $M_2 \longleftarrow \{C_1, C_2, \ldots, C_n\} \setminus M_1$;
7 **return** (M_1, M_2);

Theorem 2: Algorithm 2 terminates.

Proof: To establish the termination of Algorithm 2, we want to prove that the number of refinement iterations for $\hat{T}_1$ and $\hat{T}_2$ is finite. In Algorithm 2, $\hat{T}_2$ is initialized as $\top$, and $\hat{T}_1$ is set to the most abstract over-approximation after the first iteration. In the following iterations of Algorithm 2, $\hat{T}_1$ and $\hat{T}_2$ are refined and approaching to T_1 and T_2, respectively. For finite state systems, the refinement loop for $\hat{T}_1$ and $\hat{T}_2$ in Algorithm 2 must terminate. This is simply because we cannot strengthen a formula with a finite number of models infinitely. That is, $M_1 \parallel M_2 \models \varphi$ will be either proved or disproved in Algorithm 2 within a finite number of iterations. ■

B. Generalization to Multiple Components

The proposed compositional verification approach based on interpolation is presented in the context of two components. If a system consists of n components modeled by $M = \{C_1, C_2, \ldots, C_n\}$ where $n \geq 3$, an intuitive approach to generalize our approach is to partition the components into two groups to fit the AG-NC proof rule. For example, if $n = 4$, we can obtain $M_1 = C_1 \parallel C_2$ and $M_2 = C_3 \parallel C_4$, and apply our approach on M_1 and M_2.

However, the number of possible partitions is $2^n - 2$, which is exponential to the number of components. In addition, Cobleigh et al. [17] showed that a good partition is very important to AGR with the AG-NC proof rule. With a bad partition, assume-guarantee reasoning may not be beneficial, which is corroborated in our experiments in Section V.

In the following, we would like to show that bounded model checking can help to find good partitions efficiently. Let us recall the AG-NC proof rule for AGR. An ideal case is that we can have a conclusive verification result when the assumption $\mathcal{A}_2$ is the most abstract one, whose transition relation is $\top$. That is to say, considering only the M_1 group is sufficient to have a conclusive result, or the property to be verified is only related to the M_1 group. Based on this observation, we propose a partition heuristic based on the unsatisfiability core of BMC formula. Consider the following bounded model checking formula in k steps for the n components where $C_j = (\mathbf{x}_j, I_j, T_j)$ and $j \in \{1, 2, \ldots, n\}$.

$$\Psi = \bigwedge_{j=1}^{n} I_j^{\langle 0 \rangle} \wedge \bigwedge_{i=0}^{k-1} \bigwedge_{j=1}^{n} T_j^{\langle i \rangle} \wedge \neg \varphi^{\langle k \rangle}$$

If Ψ is not satisfiable, the property is not going to be violated in k steps. We can obtain its unsatisfiability core, denoted by U_Ψ, which includes the formula showing why the property cannot be violated in k steps. In the other hand, the unsatisfiability core also gives us a hint of which components are necessary to prove that the property is satisfied.

The heuristic, PARTITION, for partitioning components is shown in Algorithm 3. Initially, groups M_1 and M_2 are initialized as empty, respectively (line 1). The satisfiability of the bounded model checking formula Ψ in k steps is checked by a decision procedure. If it is unsatisfiable, we obtain its unsatisfiability core, denoted by U_Ψ (line 2). If a component C_j for some $j \in \{1, 2, \ldots, n\}$ has a variable appearing in the unsatisfiability core U_Ψ, we include C_j into the group M_1 because it is strongly necessary to prove that the property is satisfied (lines 3–5). The remaining components that do not have any variables appearing in U_Ψ are included into the group M_2 (line 6), and the final partitioned groups M_1 and M_2 are returned (line 7).

Algorithm 4 gives the pseudo-code of the generalized interpolation-guided compositional verification for multiple components. Initially, we assume that there is an initial partition of groups M_1 and M_2 (line 2). Then Algorithm 4 works similarly to Algorithm 2 as if there are only two hypothetical components M_1 and M_2. When a counterexample is found in abstract components in k steps (line 9), a BMC of length k is performed to check whether there exists any k-step counterexample in the concrete components (line 10). If the BMC formula is satisfied by an valuation ν (line 11), then a real counterexample is found and returned (line 12). If the BMC formula is not satisfiable (line 13), the partition heuristic is performed (line 14) with the value k to obtain a new partition (M_1', M_2'). If there is any component in M_1' but not in M_1, it is then included into M_1 (lines 15–17), and the verification restarts from scratch for the new partition (line 18). If there is no re-partition that can be made (line 19), the process continues similarly to Algorithm 2 until a verification result can be concluded. We remark that the k-step BMC formula Ψ in the partition heuristic is equivalent to the formula $\bigwedge \Theta$ for checking whether $\neg\varphi$ is k-reachable in the concrete components. Thus, the formula could be solved only once such that the unsatisfiability core as well as the interpolants are obtained from the same unsatisfiability proof.

The correctness of Algorithm 4 can be proved by Theorem 1 as well, while the termination has to be established based on Theorem 2 plus the finite number of re-partition iterations. Notice that the number of components in the M_1 group is strictly increasing, and therefore the number of re-partitions in Algorithm 4 is at most n iterations. Since the re-partitions are finite and the verification terminates for each new partition (by Theorem 2), we can conclude that Algorithm 4 terminates in a finite number of iterations.

V. EVALUATION

The proposed interpolation-based compositional verification framework has been implemented in the PAT model checker [38]. We use MathSAT [13] (an SMT solver) to obtain interpolations. MathSAT supports three different ways to obtain interpolations from unsatisfiability formulas. We use the

Algorithm 4: Generalized Interpolation-based Compositional Verification

input : $\{C_1, C_2, \ldots, C_n\}$: a set of components; φ: the property to be checked
output: yes/no, with a counterexample

1 **while** *True* **do**
2 Let (M_1, M_2) be a partition where $M_i = (\mathbf{x}_i, I_i, T_i)$ for $i \in \{1, 2\}$;
3 $\hat{T}_1 \longleftarrow T_1$;
4 $\hat{T}_2 \longleftarrow \top$;
5 **while** *True* **do**
6 **if** $(\mathbf{x}_1, I_1, \hat{T}_1) \parallel (\mathbf{x}_2, I_2, \hat{T}_2) \models \varphi$ **then**
7 **return** *yes*
8 **else**
9 Suppose $\neg\varphi$ is k-reachable in $(\mathbf{x}_1, I_1, \hat{T}_1) \parallel (\mathbf{x}_2, I_2, \hat{T}_2)$;
10 $\Theta \longleftarrow \{I_1^{\langle 0\rangle}, I_2^{\langle 0\rangle}, T_1^{\langle 0\rangle}, T_2^{\langle 0\rangle}, T_1^{\langle 1\rangle}, T_2^{\langle 1\rangle}, \ldots, T_1^{\langle k-1\rangle}, T_2^{\langle k-1\rangle}, \neg\varphi^{\langle k\rangle}\}$;
11 **if** $\bigwedge\Theta$ *is satisfied by a valuation* ν **then**
12 **return** (no, ν)
13 **else**
14 $(M_1', M_2') \longleftarrow$ PARTITION($\{C_1, \ldots, C_n\}$, k) ;
15 **if** $M_1' \setminus M_1 \neq \emptyset$ **then**
16 $M_1 \longleftarrow M_1 \cup (M_1' \setminus M_1)$;
17 $M_2 \longleftarrow \{C_1, C_2, \ldots, C_n\} \setminus M_1$;
18 **goto** Line 2 ;
19 Let $\hat{\Theta} = \{\ldots, \hat{T}_1^{\langle 0\rangle}, \hat{T}_2^{\langle 0\rangle}, \hat{T}_1^{\langle 1\rangle}, \hat{T}_2^{\langle 1\rangle}, \ldots, \hat{T}_1^{\langle k-1\rangle}, \hat{T}_2^{\langle k-1\rangle}, \neg\hat{\varphi}^{\langle k\rangle}\}$ be the symmetric interpolant for Θ;
20 $\hat{T}_2 \longleftarrow \hat{T}_2 \wedge \bigwedge_{i=0}^{k-1} (\hat{T}_2^{\langle i\rangle})^{\langle -i\rangle}$;
21 $\hat{T}_1 \longleftarrow \bigwedge_{i=0}^{k-1} (\hat{T}_1^{\langle i\rangle})^{\langle -i\rangle}$; // Abstracting M_1 (optional)

approach proposed by McMillan [31] in our implementation. To demonstrate the feasibility and benefits of our approach, the following systems are used as benchmarks.

- **FMS**. A flexible manufacturing system (FMS) [36], [26] produces blocks with a cylindrical painted pin from raw blocks and raw pegs. The manufacturing devices are connected through buffers, and the capacity of each buffer is one. We verify the properties requiring that each buffer should not overflow.
- **DP**. The dining philosophers (DP) problem illustrates a resource sharing problem in concurrent systems. Philosophers sit at a round table, and there is only one fork between any two philosophers. A philosopher requires two forks (shared with his/her neighbors) to eat. We verify the properties requiring that any pair of neighboring philosophers cannot eat simultaneously.
- **AIP**. The AIP manufacturing system [24], [27], [28] produces two products from two types of materials in different production routes. We verify the properties requiring that the routes of the two types of materials should be opposite.
- **SBA**. The synchronous bus arbiter (SBA) is a bus arbitration protocol for synchronous circuits [30]. A bus is connected by nodes (the components to access it) in a ring, and a token is passed around the nodes. We verified the properties requiring that a bus cannot be accessed simultaneously by more than two nodes.
- **MSI**. In the MSI cache coherence protocol [30], a memory is shared by n nodes, each of which has a cache. A bus connects the caches of the nodes and the memory. We verified the properties requiring that the bus cannot be owned simultaneously by more than two nodes.

The system models[2] and verified properties of all the experiments, and the implementation of our framework can be found in [2] on-line. In our experiments, all the properties are satisfied. We compare three verification techniques: traditional BDD-based model checking [30], [33], McMillan's interpolation-based transition over-approximation [22], [23], and our interpolation-guided compositional verification. Since both of McMillan's and our approaches require an underlying verification engine, we adopt the traditional BDD-based model checking [3]. The following experimental results were obtained by running the PAT model checker on a 64-bit Windows 7 laptop with a 2.8 GHz Intel(R) Core(TM) i7-2640M processor and 4 GB RAM.

Table I shows the verification results of different techniques, where BDD denotes the traditional BDD-based model checking, Mc-ITP denotes McMillan's interpolation-based transition over-approximation, C-ITP denotes the proposed

[2]The input language of our models, which is a simplified version of NuSMV's input language, does not support parameterized module definitions.
[3]We integrate the CUDD library [3] in our implementation, and the default settings are used for all experiments.

TABLE I. VERIFICATION RESULTS

			BDD	Mc-ITP		C-ITP		C-ITP$_A$		C-ITP$_{P+A}$	
System	n	$\|\varphi\|$	Time	Time	$\|R\|$	Time	$\|R\|$	Time	$\|R\|$	Time	$\|P\|$
FSM-02	8	6	10.9	2.0	24	1.1	12	1.8	16	0.3	0
FSM-04	16	12	⋆	9.5	48	12.3	32	7.1	36	0.6	0
FSM-06	24	18	⋆	20.5	72	37.8	56	19.3	60	1.1	0
FSM-08	32	24	⋆	⋆		77.1	80	47.7	84	1.8	0
FSM-10	40	30	⋆	83.1	120	255.0	104	251.6	108	2.7	0
FSM-12	48	36	⋆	⋆		⋆		⋆		3.8	0
FSM-14	56	42	⋆	192.8	168	258.0	152	190.1	156	5.2	0
FSM-16	64	48	⋆	298.1	192	360.5	176	282.8	180	7.3	0
FSM-18	72	54	⋆	368.3	216	512.0	200	432.7	204	9.1	0
FSM-20	80	60	⋆	506.9	240	718.4	224	576.8	228	11.8	0
FSM-24	96	72	⋆	◷		1020.8	272	885.5	276	17.9	0
FSM-30	120	90	⋆	◷		◷		◷		28.6	0
DP-04	8	4	4.8	198.9	14	5.3	13	1.4	13	0.5	5
DP-06	12	6	⋆	7.5	22	⋆		2.1	19	1.1	7
DP-08	16	8	⋆	18.7	28	⋆		3.7	25	1.9	11
DP-10	20	10	⋆	19.0	34	⋆		8.7	32	2.9	14
DP-20	40	20	⋆	35.7	63	⋆		26.0	61	13.3	29
DP-30	60	30	⋆	115.6	106	⋆		118.9	101	32.6	44
DP-40	80	40	⋆	200.0	124	⋆		138.6	122	68.5	58
DP-50	100	50	⋆	281.1	154	⋆		276.2	154	122.2	74
DP-60	120	60	⋆	493.2	187	⋆		469.7	185	199.7	89
DP-70	140	70	⋆	645.2	214	⋆		695.9	215	313.6	104
AIP-01	8	2	106.5	1.7	10	10.8	9	3.7	10	10.5	0
AIP-02	16	4	⋆	6.6	20	168.3	20	12.4	20	39.9	12
AIP-04	32	8	⋆	30.0	40	⋆		86.8	44	81.5	24
AIP-06	48	12	⋆	87.0	60	⋆		267.5	68	128.3	36
AIP-08	64	16	⋆	189.8	80	⋆		611.9	92	180.7	48
AIP-10	80	20	⋆	352.5	100	⋆		1170.6	116	242.7	60
AIP-11	88	22	⋆	459.8	110	⋆		1539.4	128	274.6	66
AIP-12	96	24	⋆	614.2	120	⋆		◷		317.8	72
SBA-02	8	12	⋆	⋆		140.3	44	13.1	48	3.7	36
SBA-03	12	18	⋆	⋆		631.1	74	214.7	78	6.7	54
SBA-04	16	24	⋆	⋆		1086.2	104	420.5	108	11.5	72
SBA-05	20	30	⋆	⋆		1602.1	134	651.0	138	17.1	90
SBA-06	24	36	⋆	⋆		◷		897.1	168	22.9	108
SBA-07	28	42	⋆	⋆		◷		1038.4	198	31.9	126
SBA-08	32	48	⋆	⋆		◷		1291.7	228	41.7	144
SBA-09	36	54	⋆	⋆		◷		1536.3	258	53.2	162
SBA-10	40	60	⋆	⋆		◷		◷		71.4	180
MSI-02	8	1	⋆	0.3	3	2.7	3	0.7	3	1.4	1
MSI-03	11	3	⋆	3.2	11	◷		◷		4.4	3
MSI-04	14	6	⋆	◷		◷		◷		12.7	6
MSI-05	17	10	⋆	⋆		⋆		⋆		⋆	

n: number of components; $|\varphi|$: number of verified properties;
Time: verification time (in secs); $|R|$: number of refinements; $|P|$: number of re-partitions
⋆: out of memory; ◷: time out (30 minutes)

interpolation-guided compositional verification, and C-ITP$_A$ denotes the C-ITP approach with abstraction of M_1. We remark in the above experiments the number of components involved in the systems (denoted by n) is more than 2 and therefore we need to partition the components into two groups for the C-ITP and C-ITP$_A$ approaches. Specifically, we put the first four components in the M_1 group and the remains in the M_2 group. Note that the order of components can be specified by users in the input model. In this set of experiments, we randomly picked one possible order and fixed it for all experiments unless the partition heuristic is performed.

As we expected, BDD-based model checking performed worst because it ran out of all available memory for most of the cases. In average, McMillan's approach performed better than the C-ITP approach because the partition of the M_1 and M_2 groups is not good, which leaded to many cases of running out of memory or time out. However, with the abstraction of M_1, most of these cases can be verified by the C-ITP$_A$ approach in 30 minutes, which shows the significant benefit of abstracting M_1 in assume-guarantee reasoning. We remark here that the integration of the SMT solver, MathSAT, is done by interprocess communications, i.e., a dedicated process is created for MathSAT, and the problems (in SMT-LIB [1] format) to be solved as well as the output interpolations or the unsatisfiability cores are stored in shared string buffers. This implementation is not optimized because it invokes system calls many times, which is time-consuming. The performance could be improved if MathSAT is integrated natively as a library.

We also applied our generalized approach (with the partition heuristic as well as abstracting M_1), denoted by C-ITP$_{P+A}$, on the application examples, and the verification results are shown in the right-most column. The initial partition is obtained by performing the partition heuristic with length two, which is short but gives a rough understanding of the components. We did not list the number of refinements for the C-ITP$_{P+A}$ approach in the table because the partition heuristic is able to find good partitions where all the components related to the property are put into the M_1 group so that the property can be proved to hold with the most abstract assumption whose transition relation is $\top$, i.e., no refinements are required. Instead, we list the number of re-partitions for the C-ITP$_{P+A}$ approach. In the FSM example, good partitions can be found initially, while other examples require re-partitions. In the MSI example, no approach can handle the case of five nodes, which consists of seventeen components, because of

running out of memory. After our investigation, we found that the bottleneck is the underlying BDD-based verification engine. Since the transition relations of the MSI components are rather complicated, the underlying BDD-based verification easily runs out of memory. In average, the C-ITP$_{P+A}$ approach is the best one, especially when the system size is large.

VI. Related Work

Model checking [14], [35] suffers from the *state explosion* problem. To alleviate the problem, Pnueli firstly proposed the assume-guarantee paradigm [34] to verify system components individually and use the individual verification results to deduce additional properties of the system. Clarke et al. [15] used interface processes to model the abstract environment for a component, which is much smaller than the real one, such that the state space is reduced. For formal verification that is not based on model checking, Xu et al. [39] proposed a proof system based on the assume-guarantee paradigm for verifying shared variable concurrent programs. Henzinger et al. [21] reported several case studies about applying assume-guarantee reasoning on real world systems.

Cobleigh et al. [16] proposed a framework that generates the abstract environment of components automatically using the L* algorithm [5] based on the AG-NC proof rule. This work is a pioneer of automating the compositional verification based on learning techniques. Consequently, several improvements [11], [37], [19] have been proposed to further reduce the complexity. These improvements focus on reducing the size of the alphabet during learning, which dominates the time complexity of the membership query required the L* algorithm. Instead of adopting the non-circular AG-NC proof rule, Barringer et al. used the L* algorithm to learn assumptions automatically for AGR based on the circular and symmetric proof rule [6]. Lin et al. extended the learning-based compositional verification on timed systems [25], [29], [26].

In traditional assume-guarantee reasoning (AGR), the M_1 component in the AG-NC proof rule is never changed during the whole verification process, which is very different from *compositional abstraction* [7], [10], [9] where each component is abstracted and refined iteratively. The approach proposed in this work breaks with tradition of AGR such that both of the M_1 and M_2 components are abstracted and refined by the interpolants obtained from unsatisfiability proofs of bounded model checking formulas.

The closest work to the proposed approach in this paper is [12], which focuses on automatic assumption generation for compositional symbolic verification as well. We have tried to obtain an implementation of [12] for experimental comparisons, but failed. The differences between this work and [12] are listed as follows, and we compare them in theoretical point of views.

- Our approach uses interpolation techniques to generate the assumption, while [12] uses the CDNF algorithm [8], which is an active algorithm for learning Boolean formulas from membership and candidate queries.
- Regarding the AG-NC proof rule in Equation 1, our approach need not check the second condition, $M_2 \preceq \mathcal{A}_2$, because $\mathcal{A}_2$ is an abstraction of M_2 by construction according to the characteristic of interpolations. However, in [12], $M_2 \preceq \mathcal{A}_2$ has to be verified by model checking each time when a candidate assumption $\mathcal{A}_2$ is constructed, which is an additional overhead compared to our approach.
- The partition problem in AGR is not solved in [12], i.e., the partition has to be given manually, while our approach solves it by unsatisfiability cores of BMC formulas.

VII. Conclusion and Future Work

In this work, we propose an automatic compositional symbolic verification based on interpolations. The assumption $\mathcal{A}_2$ required by assume-guarantee reasoning is obtained by symmetric interpolants from the unsatisfiability proofs of bounded model checking. In addition, the proposed approach also weakens the component M_1 based on interpolations during the verification, which further alleviates the state space explosion problem when checking $M_1 \parallel \mathcal{A}_2 \models \varphi$. Currently, we use McMillan's interpolation technique. In the future, we plan to use different interpolation techniques to generate the assumptions and to compare the verification results based on different interpolation techniques.

References

[1] http://smt-lib.org/.

[2] https://sites.google.com/site/shangweilin/itpagr.

[3] http://vlsi.colorado.edu/ fabio/cudd/.

[4] R. Alur, P. Madhusudan, and W. Nam. Symbolic compositional verification by learning assumptions. In *CAV*, volume 3576 of *LNCS*, pages 548–562, 2005.

[5] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[6] H. Barringer, D. Giannakopoulou, and C. S. Păsăreanu. Proof rules for automated compositional verification through learning. In *SAVCBS*, pages 14–21, 2003.

[7] S. Bensalem, Y. Lakhnech, and S. Owre. Computing abstractions of infinite state systems compositionally and automatically. In *CAV*, volume 1427 of *LNCS*, pages 319–331, 1998.

[8] N. H. Bshouty. Exact learning boolean function via the monotone theory. *Information and Computation*, 123(1):146–153, 1995.

[9] S. Chaki, E. Clarke, O. Grumberg, J. Ouaknine, N. Sharygina, T. Touili, and H. Veith. State/event software verification for branching-time specifications. In *IFM*, volume 3771 of *LNCS*, pages 53–69, 2005.

[10] 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*, 89(3), 2003.

[11] S. Chaki and O. Strichman. Optimized L*-based assume-guarantee reasoning. In *TACAS*, volume 4424 of *LNCS*, pages 276–291, 2007.

[12] Y.-F. Chen, E. M. Clarke, A. Farzan, M.-H. Tsai, Y.-K. Tsay, and B.-Y. Wang. Automated assume-guarantee reasoning through implicit learning. In *CAV*, volume 6174 of *LNCS*, pages 511–526, 2010.

[13] A. Cimatti, A. Griggio, B. Schaafsma, and R. Sebastiani. The MathSAT5 SMT Solver. In *TACAS*, volume 7795 of *LNCS*, 2013.

[14] E. M. Clarke and E. A. Emerson. Design and sythesis of synchronization skeletons using branching time temporal logic. In *Logics of Programs Workshop*, volume 131, pages 52–71, 1981.

[15] E. M. Clarke, D. E. Long, and McMillan K. L. Compositional model checking. In *LICS 1989*, pages 353–362, 1989.

[16] J. M. Cobleigh, D. Giannakopoulou, and C. S. Păsăreanu. Learning assumptions for compositional verification. In *TACAS*, volume 2619 of *LNCS*, pages 331–346, 2003.

[17] J.M. Cobleigh, G. S. Avrunin, and L. A. Clarke. Breaking up is hard to do: An investigation of decomposition for assume-guarantee reasoning. In *ISSTA*, pages 97–108, 2006.

[18] M. Craig. A new form of the herbrand-gentzen theorem. *Journal of Symbolic Logic*, 22(3):250–268, 1957.

[19] M. Gheorghiu, D Giannakopoulou, and C. S. Păsăreanu. Refining interface alphabets for compositional verification. In *TACAS*, volume 4424 of *LNCS*, pages 292–307, 2007.

[20] O. Grumberg and D. E. Long. Model checking and modular verification. In *CONCUR*, volume 527 of *LNCS*, pages 250–265, 1991.

[21] T. A. Henzinger, S. Qadeer, and S. K. Rajamani. You assume, we guarantee: Methodology and case studies. In *CAV*, volume 1427 of *LNCS*, pages 440–451, 1998.

[22] R. Jhala and K. L. McMillan. Interpolant-based transition relation approximation. In *CAV*, volume 3576 of *LNCS*, pages 39–51, 2005.

[23] R. Jhala and K. L. McMillan. Interpolant-based transition relation approximation. *Logical Methods in Computer Science*, 3(4), 2007.

[24] R. J. Leduc, M. Lawford, and P. C. Dai. Hierarchical interface-based supervisory control of a flexible manufacturing system. *IEEE Transactions on Control Systems Technology*, 14(4):654–668, 2006.

[25] S.-W. Lin, É. André, J. S. Dong, J. Sun, and Y. Liu. An efficient algorithm for learning event-recording automata. In *ATVA*, volume 6996 of *LNCS*, pages 463–472, 2011.

[26] S.-W. Lin, É. André, Y. Liu, J. Sun, and J. S. Dong. Learning assumptions for compositional verification of timed systems. *IEEE Transactions on Software Engineering (TSE)*, 40(2):137–153, 2014.

[27] S.-W. Lin and P. A Hsiung. Counterexample-guided assume-guarantee synthesis through learning. *IEEE Transactions on Computers*, 60(5):734–750, 2011.

[28] S.-W. Lin and P.-A. Hsiung. Compositional synthesis of concurrent systems through causal model checking and learning. In *FM*, volume 8442 of *LNCS*, pages 416–431, 2014.

[29] S.-W. Lin, Y. Liu, J. Sun, J. S. Dong, and É. André. Automatic compositional verification of timed systems. In *FM*, volume 7436 of *LNCS*, pages 272–276, 2012.

[30] K. L. McMillan. *Symbolic Model Checking: An approach to the state explosion problem*. Ph. D. Thesis, Carnegie Mellon University, 1992.

[31] K. L. McMillan. Interpolation and sat-based model checking. In *CAV*, volume 2725 of *LNCS*, pages 1–13, 2003.

[32] W. Nam and R. Alur. Learning-based symbolic assume-guarantee reasoning with automatic decomposition. In *ATVA*, volume 4218 of *LNCS*, pages 170–185, 2006.

[33] T. K. Nguyen, J. Sun, Y. Liu, and J. S. Dong. A model checking framework for hierarchical systems. In *ASE*, pages 633–636, 2011.

[34] A. Pnueli. In transition from global to modular temporal reasoning about programs. In *Logics and Models of Concurrent Systems*, pages 123–144, 1985.

[35] J. P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *International Symposium on Programming*, volume 137, pages 337–351, 1982.

[36] M. H. Queiroz, J. E. R. Cury, and W. M. Wonham. Multitasking supervisory control of discrete-event systems. *Discrete Event Dynamic Systems*, 15(4), 2005.

[37] N. Sinha and E. M. Clarke. SAT-based compositional verification using lazy learning. In *CAV*, volume 4590 of *LNCS*, pages 39–54, 2007.

[38] J. Sun, Y Liu, J. S. Dong, and J. Pang. PAT: Towards flexible verification under fairness. In *CAV*, volume 5643 of *LNCS*, pages 709–714, 2009.

[39] Q. Xu, W. P de Roever, and J. He. The rely-guarantee method for verifying shared variable concurrent programs. *Formal Aspects of Computing*, 9(2):149–174, 1997.