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 $A_2$, 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 $M_1 \parallel A_2 \models \exists \varphi$ if multiple refinements of $A_2$ 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 $A_2$ 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 $\varphi$, 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:

$$M_1 \models \exists \varphi \text{ and } M_2 \models \nexists \varphi \text{ } \Rightarrow \text{ } M_1 \models \nexists \varphi$$

Given a system with two components modeled by $M_1$ and $M_2$ and a property $\varphi$, the AG-NC proof rule says that if $M_1$ can satisfy a property $\varphi$ under an assumption $A_2$ and $A_2$ is an abstraction of $M_2$, $M_2$ can be simulated by $A_2$, denoted by $M_2 \leq A_2$ as formulated in Section III-A), then we can conclude that $M_1 \parallel M_2$ satisfies $\varphi$. 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 $A_2$. In general, the assumption should be kept as small as possible, i.e., containing only sufficient details to prove or disprove $M_1 \parallel A_2 \models \varphi$. Besides relying on human creativity to create $A_2$ 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 $A_2$ is refined all the way to $M_2$. As a result, all the effort on finding the assumptions and checking $M_1 \parallel A_2 \models \varphi$, 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 variables1. 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 $A_2$ for $G_2$ but also an abstraction $A_1$ for $G_1$. Initially, we set the transition relation of $A_2$ to be TRUE, which is the weakest over-approximation, and $A_1$ to be $G_1$. We then model check $A_1 \parallel A_2$. If $A_1 \parallel A_2$ satisfies the property $\varphi$, we prove the system satisfies $\varphi$. 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

1Our work can be extended to support messaging or barrier synchronization.
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 $A_2$ (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 $A_1$ of $G_1$ (so as to avoid details of the processes which are irrelevant 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 $A_2$ 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 $A_1$ and $A_2$ 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, cell$_1$ and cell$_2$ in Fig. 2. Each component cell$_i$ for $i \in \{1, 2\}$ has three Boolean variables as follows. The in$_i$ variable indicates whether the carry-in value of cell$_i$ is asserted. The bit$_i$ variable stores the current bit value of 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 out$_i$ indicates whether the current bit value of cell$_i$ should be carried out. If the bit value of cell$_1$ is carried out, then the carry-in value of cell$_2$ should be asserted. The initial condition $I_i$ and transition relation $T_i$ of the two components are encoded as follows, respectively.

- $I_1: \neg$bit$_1 \land$ in$_1$
- $I_2: \neg$bit$_2$
- $T_1: (\text{bit}_1 \leftrightarrow \text{bit}_1 \oplus \text{in}_1) \land (\text{out}_1 \leftrightarrow \text{bit}_1 \land \text{in}_1)$
- $T_2: (\text{in}_2' \leftrightarrow \text{out}_1) \land (\text{bit}_2' \leftrightarrow \text{bit}_2 \oplus \text{in}_2) \land (\text{out}_2' \leftrightarrow \text{bit}_2 \land \text{in}_2)$

Suppose we want to verify the property $\varphi$ requiring that out$_2$, bit$_2$, and in$_2$ do not hold simultaneously, i.e., $G \neg(\text{out}_2 \land \text{bit}_2 \land \text{in}_2)$. Let cell$_1$ be $M_1$ and cell$_2$ be $M_2$, respectively. We use $T_1^l$ to denote the over-approximation of $T_1$ after $l$-th iteration. Initially in our approach, $T_2^0$ is set to the weakest transition relation $\top$, and $T_1^0$ is kept as $T_1$. Let $A_1^l$ be the component encoded by the initial condition $I_i$ and the abstract transition relation $T_i^l$.

In the first iteration, a counterexample is found in one step when model checking $A_1^0 \parallel A_2^0 \models \varphi$. 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 $T_1^0$ should be strengthened. From the proof of unsatisfiability, we obtain the symmetric interpolant $\Omega_1 = \top$ for $T_1$ and $\Omega_2 = \text{bit}_2 \lor \neg$out$_2'$ for $T_2$, respectively (c.f. Section III-B for details). We use the obtained interpolant to weaken $T_1^0$ and strengthen $T_2^0$ as follows: $T_1^1 = \Omega_1 = \top$ and $T_2^1 = T_2^0 \land \Omega_2 = (\text{bit}_2 \lor \neg$out$_2')$. By the characteristics of interpolants, $T_1^1$ and $T_2^1$ are over-approximations of $T_1$ and $T_2$, respectively. In addition, $I_1 \land T_1^1 \land I_2 \land T_2^1$ does not admit any one-step counterexamples.

In the second iteration, $A_1^1 \parallel A_2^1 \models \varphi$ 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 $T_3^1$ still needs to be strengthened. We obtain the interpolants $\Omega_1^1$ and $\Omega_2^1$ from the unsatisfiability proof to refine $T_1^1$ and $T_2^1$, respectively.

Fig. 1. Overall Flow
respectively, as follows: $\hat{T}_2^2 = \Omega_2' = \text{bit}_1 \lor \neg \text{out}_1'$ and $\hat{T}_2^3 = \text{in}_2 \land \neg \text{bit}_2 \lor \text{bit}_1 \lor \neg \text{out}_1' \lor \neg \text{in}_2 \lor \text{in}_2 \land \text{bit}_2$.

In the third iteration, a spurious counterexample in seven steps is found, and $\hat{T}_2^3$ and $\hat{T}_2^4$ are strengthened by interpolants as follows: $\hat{T}_2^4 = \text{bit}_1 \lor \neg \text{out}_1'$ and $\hat{T}_2^5 = \text{in}_2 \land \neg \text{bit}_2 \lor \text{bit}_1 \lor \neg \text{out}_1' \lor \neg \text{in}_2 \lor \text{in}_2 \land \text{bit}_2$.

In the fourth iteration, $A_1 \parallel A_2 \models \phi$ is verified by model checking again, but no counterexamples are found this time meaning that $\text{cell}_1 \parallel \text{cell}_2 \models \phi$. We remark here that abstracting $T_2$ is optional, but doing so reduces the state explosion problem when checking $M_1 \parallel A_2 \models \phi$.

Let us do the verification again, but this time let cell$2$ be $M_1$ instead of cell$1$. The verification can be done in one iteration, where $A_1 = \text{cell}_2$ and $A_2$ with the weakest transition relation $\mathcal{T}$. This is because 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 $\phi_j$: $\neg (\text{out}_j \land \text{bit}_j \land \text{in}_j)$ for $j \in \{1, 2, \ldots, n\}$, our approach is able to detect that cell$2$ is the only necessary component to prove $\phi_j$, i.e., cell$1$ 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 $\phi_j$).

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 $\text{TRUE}$ and $\text{FALSE}$, respectively. Let $x$ be a set of Boolean variables and $|x|$ the size of $x$. A Boolean formula $\phi(x)$ over $x$ is a function from $\mathbb{B}^{|x|}$ to $\mathbb{B}$. A valuation $\nu : x \rightarrow \mathbb{B}$ over $x$ is a function from Boolean variables to truth values. We use $\phi[\nu]$ to denote the result of evaluating $\phi$ by replacing each $x \in x$ with $\nu(x)$.

To represent transition systems symbolically, we also define a set of Boolean variables $x' = \{x' | x \in x\}$, which corresponds to $x$ such that $x \in x$ represents the current value of $x$, while $x' \in x'$ represents the value of $x$ in the next state. Moreover, let $\phi(x, x')$ be a Boolean formula over $x$ and $x'$. If $\nu$ and $\nu'$ are valuations over $x$ and $x'$, respectively, we use $\phi[\nu, \nu']$ to denote the result of evaluating $\phi$ by replacing each $x \in x$ with $\nu(x)$ and replacing each $x' \in x'$ with $\nu'(x')$. Let $\mathcal{C}$ be a set of formulas. We use $\land \mathcal{C}$ to denote the conjunction of all formulas.

A transition system $M = (x, I(x), T(x, x'))$ consists of its state variables $x$, its initial predicate $I(x)$, and its transition relation $T(x, x')$. We sometimes write $(x, I, T)$ 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 $\nu_i$ is a valuation over $x$, such that $I(\nu_0) = \top$ and $T(\nu_i, \nu_{i+1}) = \top$ for all $i \in \{0, 1, \ldots, k\}$. The language of $M$, denoted by $\mathcal{L}(M)$, contains all the traces of $M$. A state predicate $\phi(x)$ over $x$ is a Boolean formula over $x$. We say $M$ satisfies $\phi$, denoted by $M \models \phi$, if for each $\sigma = \nu_0 \nu_1 \cdots \nu_k \in \mathcal{L}(M)$, we have $\phi[\nu] = \top$ for all $i \in \{0, 1, \ldots, k\}$. A counterexample of $M \models \phi$ is a trace $\nu_0 \nu_1 \cdots \nu_k$ of $M$ such that $\phi[\nu] = \top$ for all $i \in \{0, 1, \ldots, t-1\}$ but $\phi[\nu^t] = \bot$.

Let $M = (x, I(x), T(x, x'))$ and $A = (x, I_A(x), T_A(x, x'))$ be two transitions systems over $x$. We say $M$ is simulated by $A$ or $A$ simulates $M$, denoted by $M \preceq A$, if $\forall x. I(x) \implies I_A(x)$ and $\forall x'. T(x, x') \implies T_A(x, x')$. That is, the initial condition of $M$ is stronger than that of $A$ and every transition in $M$ is also allowed in $A$. Obviously, if $M \preceq A$, then $\mathcal{L}(M) \subseteq \mathcal{L}(A)$ holds. Let $M_i = (x_i, I_i(x_i), T_i(x_i, x'_i))$ be two transition systems for $i \in \{1, 2\}$. The parallel composition of $M_1$ and $M_2$ is the transition system $M_1 \parallel M_2 = (x_1 \cup x_2, I_1(x_1) \land I_2(x_2), T_1(x_1, x'_1) \land T_2(x_2, x'_2))$.

Given a transition system $M = (x, I(x), T(x, x'))$ and a state predicate $\phi(x)$, whether $\phi$ is $k$-reachable in $M$ can be expressed symbolically as a Boolean formula. For each variable $x_i \in x$ and a natural number $i$, we use $x_i^{(i)}$ to denote the variable $x_i$ with $i$ primes added, which represents the value of $x_i$ at time $i$. For example, $x_i^{(3)} = x_i^{(0)}$ represents the value of $x_i$ at time 3. We also extend this notation to the set of variables and formulas. Thus, $x^{(i)}$ contains variables with $i$ primes added, $\phi(x)^{(i)}$ is the formula over $x^{(i)}$, and $\phi(x_i^{(i)})$ is the formula over $x_i^{(i)}$ and $x^{(i+1)}$. A state predicate $\phi(x)$ is $k$-reachable in $(x, I(x), T(x, x'))$ if the following bounded model checking (BMC) formula is satisfiable:

$$I(x)^{(0)} \land T(x, x')^{(0)} \land T(x, x')^{(1)} \land \cdots \land T(x, x')^{(k-1)} \land \phi(x)^{(k)}$$
Algorithm 1: Verification by Transition Approximation

```
input: (x, I, T): the concrete transition system;
\varphi: the property to be checked
output: yes/no, with a counterexample

1. | \hat{T} \leftarrow \top ;
2. while True do
3.   if (x, I, \hat{T}) \models \varphi then
4.     return yes
5.   else
6.     Suppose \neg \varphi is k-reachable in (x, I, \hat{T}) ;
7.     \Theta \leftarrow \{I(0), T(0), \ldots, T(k-1), \neg \varphi(k)\} ;
8.     if \ \Theta \ is satisfied by a valuation \nu \ then
9.       return (no, \nu) ;
10.  else
11.    Let \hat{\Theta} = \{\hat{I}(0), \hat{T}(0), \ldots, \hat{T}(k-1), \neg \varphi(k)\} be the symmetric interpolant for \Theta ;
12.    \hat{T} \leftarrow \hat{T} \land \bigwedge_{i=0}^{k-1} (\hat{T}(i))^{(-i)} ;
```

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 \( A \land B \) 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. \( \hat{A} \land B \) is unsatisfiable
3. \( \hat{A} \) refers only to the common variables of \( A \) and \( B \).

If \( A \land B \) 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 \( (x, I, \hat{T}) \models \varphi \) holds, we can conclude \( (x, I, T) \models \varphi \) also holds because \( T \equiv \hat{T} \) (lines 3–4).

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

\[
I(0) \land \tilde{T}_{[0]}(0) \land \tilde{T}_{[1]}(1) \land \ldots \land \tilde{T}_{[k]}(k-1) \land \neg \varphi(k)
\]

is unsatisfiable, i.e., \( \bigwedge_{i=0}^{k-1} \tilde{T}_{[i]} \) admits no path in \( k \) steps from \( I \) to \( \neg \varphi \). Thus, \( \tilde{T} \) is refined as \( T \land \bigwedge_{i=0}^{k-1} \tilde{T}_{[i]} \), 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 \( A_2 \), we can construct \( A_2 \) 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 \( A_2 \) is an over-approximation of that of \( M_2 \), the second condition of Equation 1, \( M_2 \preceq A_2 \), holds naturally. We only have to check whether the first condition, \( M_1 \parallel M_2 \models \varphi \), 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 \( A_2 \) 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.

\[
A_1 \parallel A_2 \models \varphi \quad \text{and} \quad M_2 \preceq A_2 \quad \text{and} \quad M_1 \preceq A_1
\]

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

\[ \text{input} : M_1 = (x_1, I_1, T_1) \text{ and } M_2 = (x_2, I_2, T_2): \text{ concrete transition systems; } \varphi: \text{ the property to be checked} \]

\[ \text{output: yes/no, with a counterexample} \]

1. \( T_1 \leftarrow T_1; \)
2. \( T_2 \leftarrow \top; \)
3. \textbf{while} True \textbf{do}
   4. \textbf{if} \((x_1, I_1, T_1) \parallel (x_2, I_2, T_2) \models \varphi\) \textbf{then}
      5. \textbf{return} yes
   \textbf{else}
   7. Suppose \( \neg \varphi \) is \( k \)-reachable in \((x_1, I_1, T_1) \parallel (x_2, I_2, T_2)\);
   8. \( \Theta \leftarrow \{I_1^{(0)}, I_2^{(0)}, T_1^{(0)}, T_2^{(0)}, T_1^{(1)}, T_2^{(1)}, \ldots, T_1^{(k-1)}, T_2^{(k-1)}, \neg \varphi^{(k)}\}; \)
   9. \textbf{if} \( \wedge \Theta \) is satisfied by a valuation \( \nu \) \textbf{then}
      10. \textbf{return} (no, \( \nu \))
   \textbf{else}
   12. \( \hat{T}_1 \leftarrow T_1 \land \wedge_{i=0}^{k-1} (T_1^{(i)})^{(-i)}; \)
13. \( \hat{T}_2 \leftarrow T_2 \land \wedge_{i=0}^{k-1} (T_2^{(i)})^{(-i)}; \)
14. \( \hat{T}_1 \leftarrow \wedge_{i=0}^{k-1} (T_1^{(i)})^{(-i)}; \) \text{ // Abstracting } M_1 \text{ (optional)}

Because of the properties of symmetric interpolants, the following bounded model checking formula

\[ I_1^{(0)} \land I_2^{(0)} \land \bigwedge_{i=0}^{k-1} \hat{T}_1^{(i)} \land \bigwedge_{i=0}^{k-1} \hat{T}_2^{(i)} \land \neg \varphi^{(k)} \]

is unsatisfiable. That is to say, \( \bigwedge_{i=0}^{k-1} \hat{T}_1^{(i)} \) and \( \bigwedge_{i=0}^{k-1} \hat{T}_2^{(i)} \) admit no path in \( k \) steps from \( I_1 \land I_2 \) to violate \( \varphi \). Note that \( \bigwedge_{i=0}^{k-1} \hat{T}_1^{(i)} \) is an over-approximation of \( T_1 \) as well as a refinement of \( T_2 \). Thus, we strengthen \( T_2 \) as \( T_2 \land \wedge_{i=0}^{k-1} \hat{T}_2^{(i)} \) for the next iteration (line 13). In addition, since \( \bigwedge_{i=0}^{k-1} \hat{T}_1^{(i)} \) is an over-approximation of \( T_1 \), we can optionally abstract \( \hat{T}_1 \) as \( \bigwedge_{i=0}^{k-1} \hat{T}_1^{(i)} \) in line 14, which alleviates the state space explosion problem when checking whether \((x_1, I_1, T_1) \parallel (x_2, I_2, T_2) \models \varphi\).

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 \( M_1 = (x_1, I_1, T_1) \) and \( M_2 = (x_2, I_2, T_2) \) be the transition systems with respect to \( T_1 \) and \( T_2 \), respectively. Since \( T_1 \) and \( T_2 \) are obtained by interpolations, both \( T_1 \models T_1 \) and \( T_2 \models T_2 \) hold, i.e., \( M_1 \) and \( M_2 \) are the abstractions of \( M_1 \) and \( M_2 \), respectively. Algorithm 2 returns “yes” only when \( \wedge \Theta \) is a bounded model checking formula to check whether \( \neg \varphi \) is reachable within \( k \)-steps in \( M_1 \parallel M_2 \), the valuation \( \nu \) is a witness of \( M_1 \parallel M_2 \not\models \varphi \). On the other hand, Algorithm 2 returns “no” only when \( \wedge \Theta \) is satisfiable by a valuation \( \nu \). Since \( \wedge \Theta \) is a bounded model checking formula, we can conclude that Algorithm 2 is correct. \( \blacksquare \)
Theorem 2: Algorithm 2 terminates.

Proof: To establish the termination of Algorithm 2, we want to prove that the number of refinement iterations for \( T_1 \) and \( T_2 \) is finite. In Algorithm 2, \( T_2 \) is initialized as \( T \), and \( T_1 \) is set to the most abstract group after the first iteration. In the following iterations of Algorithm 2, \( T_1 \) and \( T_2 \) are refined and approaching to \( T_1 \) and \( T_2 \), respectively. For finite state systems, the refinement loop for \( T_1 \) and \( T_2 \) in Algorithm 2 must terminate. This is simply because we cannot strengthen a formula with a finite number of models infinitely. That is, \( M_1 \cup M_2 \models \varphi \) 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 \( A_0 \) is the most abstract one, whose transition relation is \( T \). 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 \( n \) steps for the \( n \) components where \( C_j = \langle x_j, I_j, T_j \rangle \) and \( j \in \{1, 2, \ldots, n\} \).

\[
\Psi = \bigwedge_{j=1}^{n} I_j^{(0)} \land \bigwedge_{i=0}^{k-1} \bigwedge_{j=1}^{n} T_j^{(i)} \land \neg \varphi^{(k)}
\]

If \( \Psi \) is not satisfiable, the property is not going to be violated in \( k \) steps. We can obtain its unsatisfiability core, denoted by \( U_\Psi \), 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, \( \text{PARTITION} \), 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 \( \Psi \) in \( k \) steps is checked by a decision procedure. If it is unsatisfiable, we obtain its unsatisfiability core, denoted by \( U_\Psi \) (line 2). If a component \( C_j \) for some \( j \in \{1, 2, \ldots, n\} \) has a variable appearing in the unsatisfiability core \( U_\Psi \), 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 \( U_\Psi \) 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 a valuation \( \nu \) (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 \( \Psi \) 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

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

\begin{algorithmic}
\State \textbf{while True do}
\State \hspace{1em} Let \((M_1, M_2)\) be a partition where \(M_i = (x_i, I_i, T_i)\) for \(i \in \{1, 2\}\);
\State \hspace{1em} \(\hat{T}_1 \leftarrow T_1\);
\State \hspace{1em} \(\hat{T}_2 \leftarrow \top\);
\State \hspace{1em} \textbf{while True do}
\State \hspace{2em} if \((x_1, I_1, \hat{T}_1) \parallel (x_2, I_2, \hat{T}_2)\) |\(\varphi\) then
\State \hspace{3em} return yes
\State \hspace{2em} \textbf{else}
\State \hspace{3em} \hspace{1em} Suppose \(\neg \varphi\) is \(k\)-reachable in \((x_1, I_1, \hat{T}_1) \parallel (x_2, I_2, \hat{T}_2)\);
\State \hspace{3em} \hspace{1em} \(\Theta \leftarrow \{I_1^{(0)}, I_2^{(0)}, \hat{T}_1^{(0)}, \hat{T}_2^{(0)}, T_1^{(1)}, T_2^{(1)}, \ldots, T_1^{(k-1)}, T_2^{(k-1)}, \neg \varphi^{(k)}\}\);
\State \hspace{3em} \hspace{1em} if \(\land \Theta\) is satisfied by a valuation \(\nu\) then
\State \hspace{4em} return \((\text{no}, \nu)\)
\State \hspace{2em} \textbf{else}
\State \hspace{3em} \hspace{1em} \((M_1', M_2') \leftarrow \text{PARTITION}({C_1, \ldots, C_n}, k)\);
\State \hspace{3em} \hspace{1em} if \(M_1' \setminus M_1 \neq \emptyset\) then
\State \hspace{4em} \hspace{2em} \(M_1 \leftarrow M_1 \cup (M_1' \setminus M_1)\);
\State \hspace{4em} \hspace{2em} \(M_2 \leftarrow \{C_1, C_2, \ldots, C_n\} \setminus M_1\);
\State \hspace{4em} goto Line 2 ;
\State \hspace{3em} \hspace{1em} \(\hat{T}_2 \leftarrow \hat{T}_2 \land \bigwedge_{i=1}^{k-1} (\hat{T}_2^{(i)})^{(-i)}\);
\State \hspace{3em} \hspace{1em} \(\hat{T}_1 \leftarrow \bigwedge_{i=1}^{k-1} (\hat{T}_1^{(i)})^{(-i)}\); \hspace{1em} // Abstracting \(M_1\) (optional)
\end{algorithmic}
interpolation-guided compositional verification, and C-ITP\textsubscript{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\textsubscript{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 led 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\textsubscript{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\textsubscript{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\textsubscript{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\textsubscript{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

---

### Table I. Verification Results

| System | $n$ | $|\varphi|$ | BDD | Mc-ITP | C-ITP | C-ITP\textsubscript{A} | C-ITP\textsubscript{P+A} |
|--------|-----|---------|------|--------|------|----------------|----------------|
| FSM-02 | 8   | 6       | 10.9 | 2.0    | 24   | 1.1  | 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      | *    | 20.5   | 37.8 | 56  | 19.3| 60 | 1.1 | 0  |
| FSM-10 | 40  | 30      | *    | 83.1   | 120  | *   | 255.0| 104| 251.6| 108 | 2.7 | 0  |
| FSM-12 | 48  | 36      | *    | 83.1   | 120  | *   | 255.0| 104| 251.6| 108 | 2.7 | 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      | *    | 586.9  | 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      | *    | 1020.8 | 272  | *   | 885.5| 276| 17.9 | 0  |

$|\varphi|$: number of verified properties;
|Time| verification time (in secs);

$\star$: out of memory;
\$\$: time out (30 minutes)
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 de-
duce 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 com-
plexity. 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 algo-
  rithm [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,
  \(M_2 \preceq A_2\), because \(A_2\) is an abstraction of \(M_2\)
  by construction according to the characteristic of
  interpolations. However, in [12], \(M_2 \preceq A_2\) has to
  be verified by model checking each time when a
  candidate assumption \(A_2\) 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 assump-
tion \(A_2\) 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 \(M_1 \parallel A_2 \models \varphi\). 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

  verification by learning assumptions. In CAV, volume 3576 of LNCS,
  for automated compositional verification through learning. In SAS/CBS,
  of infinite state systems compositionally and automatically. In CAV,
[8] N. H. Bshouty. Exact learning boolean function via the monotone
  State/event software verification for branching-time specifications.
  In ICFM, volume 3771 of LNCS, pages 53–69, 2005.
  tional abstraction refinement for concurrent c programs: A two-level
  approach. Electronic Notes in Theoretical Computer Science, 89(3),
  2003.
[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
[14] E. M. Clarke and E. A. Emerson. Design and synthesis of synchro-
  nization skeletons using branching time temporal logic. In Logics of
  assumptions for compositional verification. In TACAS, volume 2619 of


