Compositional Analysis Using Component-Oriented Interpolation

Viet Yen Nguyen[1,2], Benjamin Bittner[3,4],
Joost-Pieter Katoen[2], and Thomas Noll[2]

[1] Fraunhofer IESE, Germany
[2] Software Modeling and Verification Group, RWTH Aachen University, Germany
[3] Embedded Systems Group, Fondazione Bruno Kessler, Italy
[4] ICT School, University of Trento, Italy

Abstract. We present a novel abstraction technique that exploits the compositionality of a concurrent system consisting of interacting components. It uses, given an invariant and a component of interest, bounded model checking (BMC) to quickly interpolate an abstraction of that component's environment. The abstraction may be refined by increasing the BMC bound. Furthermore, it is only defined over variables shared between the component and its environment, resulting in an aggressive abstraction with several applications. We demonstrate its use in a verification setting, as we report on our open source implementation in the NuSMV model checker which was used to perform a practical assessment with industrially-sized models from satellite case studies of ongoing missions. These models are expressed in a formalized dialect of the component-oriented and industrially standardized Architecture Analysis and Design Language (AADL).

1 Introduction

An earlier work [11] reports on the application of a wide range of model checking techniques for validating a satellite platform of an ongoing mission. This industrially-sized model was expressed in a formalized dialect [2] of the Architecture Analysis and Design Language [12]. This AADL dialect is a component-oriented formalism in which components interact through data and event ports (i.e. shared variables). The sheer size of models was particularly visible once failures were injected. The nominal state space of roughly 48 million states exploded by a factor 213563 due to the activation of failure modes and the fault management functionality for handling it. The model checkers used in literature had a hard time on this model. Various techniques have been proposed in literature to cope with similar instances of the infamous state space explosion problem. In the context of this paper, compositional reasoning and interpolation are particularly relevant.

The compositional reasoning technique by [8] was our starting point. It generates a so-called split invariant defined over the system's global variables for each parallel process. The split invariants are then checked against the property

instead of the full composition of processes. It was shown later [15, 18] that this technique, along with Cartesian abstract interpretation [18] and thread-modular verification [13], is conceptually the same as the classical Owicki-Gries paradigm, but differs in the details. They generally work well for parallel systems where processes communicate over a *small* set of *global variables*, i.e. variables that are visible to *all* processes. In the satellite models, components are highly cohesive through *shared variables*, as variables of one component are only visible to a handful of other components. The techniques from the Owicki-Gries paradigm are ineffective here as naively all shared variables would have to be interpreted as global variables, which would make it a near-monolithic model checking problem again. Another branch of compositional reasoning is the rely/assume guarantee/provide techniques. There is a huge body of work behind this. The most related ones are the automated methods that use learning techniques to generate assumptions [5]. Our work is a twist on this, because instead of learning we use *interpolation* to generate an assumptive environment. The use of interpolation techniques [9] in model checking was pioneered by McMillan [19]. Also this led to a substantial body of work. To our knowledge, it however has not been cast into a compositional reasoning scheme as we describe in this paper.

Contributions: The contributions of this paper are as follows.

- A theory inspired by Craig interpolation that results in an aggressive abstraction of the environment of a component of interest, resulting into a component-oriented interpolant.
- A rudimentary (re)verification algorithm that exploits this theory.
- An open source implementation of the algorithm in NuSMV 2.5.4 [4].
- An evaluation of the theory and implementation using industrially-sized models from satellite platform case studies of ongoing missions.

Organization: Section 2 explains applicable background information and introduces the majority of the formal notation used in this paper. Section 3 describes the theoretical contribution of this paper, the component-oriented interpolant. We implemented it into an algorithm and evaluated it using satellite platform case studies on which we report in Section 4. Related work and the conclusions are discussed respectively in Sections 5 and 6.

2 Preliminaries

Our work builds upon existing works in satisfiability (SAT) solving, bounded model checking and Craig interpolation. These are discussed in the following.

SAT Solving Propositional formulae consist of literals, which are Boolean variables (e.g. x_1) that can be negated (e.g. $\neg x_1$), and are combined by AND (i.e. $\wedge$) and OR (i.e. $\vee$) operators. They are typically processed from their conjunctive normal form (CNF) where the formula consists of conjuncted clauses (e.g. $(\neg x_1) \wedge (x_1 \vee \neg x_2)$) and each clause is a disjunction of literals (e.g. $(x_1 \vee \neg x_2)$). As we can view CNF

formulae as a set of clauses, we use the set membership notation to check whether a clause is part of a CNF formulae, e.g. $(x_1 \vee \neg x_2) \in A$ with A being a CNF formula. A classical decision problem is whether, given a propositional formula, there exists a satisfying assignment, i.e. a vector of values holding either true (i.e. $\top$) or false (i.e. $\bot$) for each variable. This is the NP-complete SAT problem. Yearly competitions have highly stimulated research in this area, progressing modern SAT-solvers to handle formulae with thousands of variables in mere seconds. They typically generate the satisfying assignment, denoted as σ, as a proof of satisfiability. In case of unsatisfiability, some SAT-solvers provide a resolution refutation graph as a proof [19]. An example of a resolution refutation graph is shown in Figure 2. It is a directed acyclic graph $G = (V, E)$, where V is a set of clauses (not necessarily a subset of the original formula). If a vertex $v \in V$ is a root (there are usually multiple), then it is a clause in the original formula. Otherwise the vertex has exactly two predecessors, v_1 and v_2 of the form $v_1 = x \vee D$ and $v_2 = \neg x \vee D'$. The clause v is the simplification of $D \vee D'$ and x is its *pivot variable*. There is only one leaf which is the empty clause $\bot$. The resolution graph reasons how clauses, starting from the root clauses, have pivot variables that can be eliminated, as they contribute to the inconsistency. Once all variables are eliminated, the empty clause $\bot$ is reached, indicating unsatisfiability.

Bounded Model Checking Propositional formulae can be used to verify a property (e.g. ϕ) of a model $M = (I, T)$. The initial condition $I(\bar{s})$ is a Boolean formula over a finite set of variables, e.g. $\bar{s} = s_1, \ldots, s_n$. The set of occurring variables is denoted by the predicate *var*, e.g. $var(I) = \{s_1, \ldots, s_n\}$. Whenever a particular valuation σ of $\bar{s}$ satisfies I, i.e. $\sigma(I) = \top$, then σ is an initial state. Multiple distinct initial states may satisfy I. The transition function, denoted as $T(\bar{s} \times \bar{s}')$, is a propositional function with $\bar{s} = s_1, \ldots, s_n$ and $\bar{s}' = s'_1, \ldots, s'_n$. Note that the cardinalities of $\bar{s}$ and $\bar{s}'$ are equal. If for a pair of valuations σ and σ' the transition function holds, i.e. $\sigma\sigma'(T) = \top$, then σ' is a valid successor state to state σ. The initial condition and the transition function are used to compute the reachable states up to a natural bound k using the formula $I[\bar{s}/\bar{r}_0] \wedge \bigwedge_{i=1}^{k} T[\bar{s}/\bar{r}_{i-1}, \bar{s}'/\bar{r}_i]$. It uses the substitution operator $T[x/y]$ to denote that all occurrences of x in T are substituted by y. We refer to $I_0 \wedge T_1 \wedge \cdots \wedge T_k$ as its simplified notation. An invariant ϕ can be verified by conjuncting its negated unrolling, $\bigvee_{i=0}^{k} \neg\phi[\bar{s}/\bar{r}_i]$, to it. To ease notation, we simply refer to this formula as $\neg\phi$. The resulting formula, $I_0 \wedge \bigwedge_{i=1}^{k} T_i \wedge \neg\phi$, can be checked for satisfiability. If it is satisfiable, the satisfying assignment is a counterexample to the invariant. If it is unsatisfiable, then the invariant holds up to bound k, which is denoted by $M \models_k \phi$. An outcome w.r.t. the full state space is however inconclusive [1].

Example 1 (Two-Bit Counter). Part of our running example is a simple two-bit counter that is initialized to 0. It is incremented by 1 with each transition until it hits the value 3. The Boolean encodings of its initial condition and transition

functions look as follows:

$$I = \neg\alpha \wedge \neg\beta$$
$$T = (\neg\alpha \vee \alpha') \wedge (\neg\alpha \vee \beta') \wedge (\alpha \vee \neg\alpha' \vee \beta) \wedge (\alpha \vee \neg\beta \vee \neg\beta') \wedge (\alpha' \vee \beta')$$

Its one-step unrolling looks as follows:

$$I_0 \wedge T_1 = (\neg\alpha_0 \wedge \neg\beta_0) \wedge (\neg\alpha_0 \vee \alpha_1) \wedge (\neg\alpha_0 \vee \beta_1) \\ \wedge (\alpha_0 \vee \neg\alpha_1 \vee \beta_0) \wedge (\alpha_0 \vee \neg\beta_0 \vee \neg\beta_1) \wedge (\alpha_1 \vee \beta_1)$$

(end of example)

Interpolation Our work is heavily inspired by Craig's seminal result [9].

Theorem 1 (Craig's Interpolation Theorem). *Let A and B be formulae of first-order logic. If $A \implies B$ holds, then there exists an interpolant C expressed using the common variables of A and B, i.e. $var(C) \subseteq var(A) \cap var(B)$, such that $A \implies C$ and $C \implies B$ holds.*

A proof of this theorem restricted to propositional logic can be found in [3]. The beauty of this theorem is that the interpolant C is expressed using a subset of the variables in A. This powerful notion inspired us for developing our compositional reasoning technique.

Note that Craig's theorem only postulates the existence of an interpolant when $A \implies B$. This can be verified with a SAT-solver. Observe that $A \implies B$ is equivalent to $\neg(A \wedge \neg B)$. This means its tautology infers the contradiction of $A \wedge \neg B$. By Craig's interpolation theorem it follows that if $A \wedge \neg B$ is unsatisfiable, there exists an interpolant C such that $A \implies C$ holds and $C \wedge \neg B$ is unsatisfiable. Thus in this shape, the unsatisfiability of a formula indicates the existence of an interpolant. It is shown in [20] how an interpolant C is generated from the resolution refutation proof resulting from the unsatisfiability of $A \wedge \neg B$. We use a similar approach to abstract a component's environment as a transition function.

3 Component-Oriented Interpolation

Our setting is a concurrent system composed of processes (also referred to as components with behavior) using a parallel composition operator. We leverage this composition to reason over its behaviour in a compositional manner. In the upcoming, we will describe our approach in the synchronous case only, i.e. all components transit per global transition. It can however be extended to the asynchronous case (i.e. interleaving transitions) by complementing the asynchronous model with an interleaving scheduler component that regulates which component progresses upon a transition.

Consider a synchronous composition of n processes $M^1, \ldots, M^n$, with their associated transition relations T^i and initial conditions I^i such that $T = \bigwedge_{i=1}^{n} T^i$ and $I = \bigwedge_{i=1}^{n} I^i$. When this is applied to the bounded model checking formula,

the result is $\bigwedge_{i=1}^{n} I_0^i \wedge \bigwedge_{i=1}^{n} T_1^i \wedge \ldots \wedge \bigwedge_{i=1}^{n} T_k^i \wedge \neg\phi$. We can now isolate any process M^p such that the remainder processes, i.e. process p's environment, shall be abstracted through interpolation. It then becomes more apparent how A and $\neg B$ are to be determined:

$$\underbrace{I_0^p \wedge T_1^p \wedge \ldots \wedge T_k^p \wedge \neg\phi}_{\neg B} \wedge \underbrace{I_0^{\neq p} \wedge T_1^{\neq p} \wedge \ldots \wedge T_k^{\neq p}}_{A} \tag{1}$$

In the above, $I_0^{\neq p} = \bigwedge_{q\in\{1,\ldots,n\}\setminus\{p\}} I_0^q$, and similarly for $T_i^{\neq p}$.

Example 2 (Counter Monitor). Let us refer to the counter example of Example 1 as M^1. We now add a monitoring process, M^2, that among its functions, raises a flag when the counter exceeds 2. The Boolean encoding of M^2, where both *delta* and γ are flags and the latter being the flag of interest, looks as follows:

$$I^2 = \neg\gamma \wedge \delta$$
$$T^2 = (\neg\alpha \vee \neg\beta \vee \neg\delta') \wedge (\neg\alpha \vee \gamma') \wedge (\alpha \vee \neg\gamma) \wedge (\alpha \vee \neg\gamma') \wedge (\beta \vee \neg\gamma) \wedge (\delta' \vee \gamma)$$

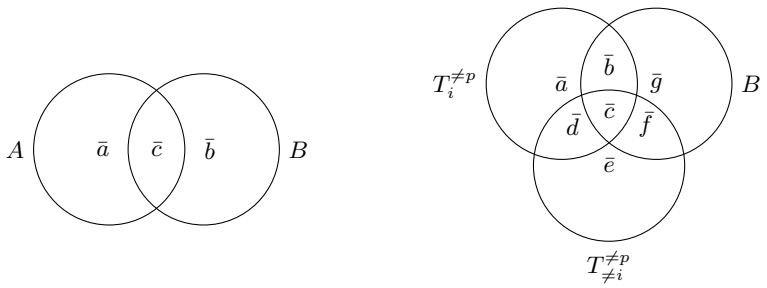
From this point on we shall use the synchronous composition $M^1 \wedge M^2$ as our ongoing running example. Let us say we are interested to see whether the flag γ always stays unraised, i.e. the invariant $\neg\gamma$. And let us isolate process M^1, i.e. $p = 1$, from the synchronous composition. This isolation on a two-step BMC unrolling would look as follows:

$$\underbrace{I_0^1 \wedge T_1^1 \wedge \wedge T_2^1 \wedge (\gamma_0 \vee \gamma_1 \vee \gamma_2)}_{\neg B} \wedge \underbrace{I_0^2 \wedge T_1^2 \wedge \wedge T_2^2}_{A} \tag{2}$$

(end of example)

From Theorem 1, it follows that whenever the invariant holds within bound k, there exists an interpolant C, such that it is implied by A. Intuitively, the interpolant C can be perceived as an abstraction of the k-fold unrolled environment of process p. It is significantly smaller than the original formula representing the k-bounded environment, since it is only defined over the variables used for interacting with process p over bound k. That is, $var(C) \subseteq var(I_0^{\neq p}, T_1^{\neq p}, \ldots, T_k^{\neq p}) \cap var(I_0^p, T_1^p, \ldots, T_k^p, \phi)$, where $var(S_1, \ldots, S_n)$ is a shorthand for $var(S_1) \cup \ldots \cup var(S_n)$. Observe that C is not a formula over current/next states, because it is interpolated from the unrolling of $T^{\neq p}$ instead of $T^{\neq p}$ itself. In that form, the interpolant C is only useful for k-bounded reverification of component p. We strive for a different kind of interpolant that is useful for unbounded analysis of component p. We call it the *component-oriented interpolant*.

To this end, let us have a closer look at the sharing of variables in the component-oriented interpolation setting, as it is slightly different from Craig interpolation. In the latter, there are two sets of variables that can be partitioned into three disjoint sets of variables, namely $var(A)\setminus var(B)$, $var(B)\setminus var(A)$ and $var(A) \cap var(B)$. This is shown in Figure 1a where the sets are respectively denoted as $\bar{a}$, $\bar{b}$, $\bar{c}$. In our component-oriented setting, there are three sets of variables



(a) Craig (b) Component-oriented

Fig. 1: Relation of variables in both interpolation settings.

which can be partitioned into seven disjoint sets. See Figure 1b. Consider any step i by component p's environment, i.e. $T_i^{\neq p}(\bar{a}, \bar{b}, \bar{c}, \bar{d})$. The remainder environment transition steps are $T_{\neq i}^{\neq p}(\bar{e}, \bar{d}, \bar{c}, \bar{f})$ and the transition steps by component p and the property ϕ are $B(\bar{g}, \bar{f}, \bar{c}, \bar{b})$. The variables of $I_0^{\neq p}$ are omitted here for clarity and are covered w.l.o.g. by the variables of $T_1^{\neq p}$.

Example 3 (Variable Sharing in Counter Monitor). Reconsider Equation (2), the two-step BMC unrolling of the counter monitor (cf. Example 2). The variable partitioning of that unrolling in the Craig interpolation case is: $\bar{a} = \{\alpha_2, \beta_2\}$, $\bar{b} = \{\alpha_0, \alpha_1, \beta_0, \beta_1, \gamma_0, \gamma_1, \gamma_2\}$ and $\bar{c} = \{\delta_0, \delta_1, \delta_2\}$.

The partitions are more fine-grained in the component-oriented interpolation case. If we would take $p = 1$ and $i = 1$ on our running example, we would get the following arrangement out of Equation (2):

$$\underbrace{I_0^1 \wedge T_1^1 \wedge \wedge T_2^1 \wedge (\gamma_0 \vee \gamma_1 \vee \gamma_2)}_{\neg B} \wedge \underbrace{I_0^2 \wedge T_1^2}_{T_1^{\neq 1}} \wedge \underbrace{\wedge T_2^2}_{T_{\neq 1}^{\neq 1}}$$

The partitioning of Figure 1b then applies, resulting in the following partitioning:

$\bar{a} = \{\delta_0, \delta_1\}$ $\bar{d} = \{\}$ $\bar{g} = \{\alpha_2, \beta_2\}$

$\bar{b} = \{\alpha_0, \beta_0, \gamma_0\}$ $\bar{e} = \{\delta_2\}$

$\bar{c} = \{\gamma_1\}$ $\bar{f} = \{\alpha_1, \beta_1, \gamma_2\}$

(end of example)

We use the finer grained notion of variable sharing in Figure 1b to construct component-oriented interpolants by traversing the resolution refutation graph of Equation (1) for each step i of component p's environment:

Definition 1 (Component-Oriented Interpolant Construction). *Let us consider step $i \leq k$ of a component p's environment. Furthermore, let $G = (V, E)$ be the resolution refutation graph of Equation* (1) *and partition the occurring variables into disjoint sets according to Figure 1b. For each non-root vertex $v \in V$,*

let v_1 and v_2 be its predecessors and x its pivot variable. Then, with each $v \in V$ we associate a Boolean formula C_v^i given as follows

$$C_v^i = \begin{cases} \bot & \text{if } v \in T_i^{\neq p} \text{ and } v \text{ is root} \\ \top & \text{if } v \in T_{\neq i}^{\neq p} \cup I^{\neq p} \cup B \text{ and } v \text{ is root} \\ (\neg x \wedge C_{v_1}^i) \vee (x \wedge C_{v_2}^i) & \text{if } x \in \bar{b} \cup \bar{c},\ x \in v_1,\ \neg x \in v_2, \text{ and } v \text{ is non-root} \\ C_{v_1}^i \vee C_{v_2}^i & \text{if } x \in \bar{a} \cup \bar{d} \text{ and } v \text{ is non-root} \\ C_{v_1}^i \wedge C_{v_2}^i & \text{if } x \in \bar{g} \cup \bar{f} \cup \bar{e} \text{ and } v \text{ is non-root} \end{cases}$$

If Definition 1 is applied starting from the leaf $\bot$, one gets a component-oriented interpolant for $T_i^{\neq p}$.

Example 4 (Component-Oriented Interpolation on Counter Monitor). Consider Figure 2. As it is an unrolling for two timesteps, there are partial interpolants for the first and second timestep, i.e. respectively C_v^1 and a C_v^2. Take for example the upper-left three-node subtree. That is $v = \neg\gamma_1$ and its two predecessors as $v_1 = \alpha_0 \vee \neg\gamma_1$ and $v_2 = \neg\alpha_0$. The pivot is therefore α_0. As we determined earlier in Example 3 that α_0 is in $\bar{b}$, the partial interpolant of v for transition step 1 becomes $(\neg\alpha_0 \wedge C_{v_1}^1) \vee (\alpha_0 \wedge C_{v_2}^1)$. Since $C_{v_1}^1 = \bot$ and $C_{v_2}^1 = \top$, this is simplified to $C_v^1 = \alpha_0$. *(end of example)*

This interpolant is weak enough to preserve the over-approximation from Craig interpolation, i.e. $T_i^{\neq p} \implies C_\bot^i$. This is captured by the following lemma:

Lemma 1 (Over-Approximation by Component-Oriented Interpolant). *Let σ be a valuation such that $\sigma(v) = \bot$ for any $v \in V$ in Definition 1. For any $1 \leq i \leq k$, the following holds:*

$$\sigma(C_v^i) = \bot \implies \exists a \in T_i^{\neq p} :: \sigma(a) = \bot \tag{3}$$

Intuitively, this means that whenever the partial interpolant C_v^i evaluates to false for a particular valuation, a clause of $T_i^{\neq p}$ evaluates to false as well for the same valuation, causing the whole formula (see Equation (1)) to evaluate to false.

Proof. Due to paper size constraints, we only provide a proof sketch. The full proof is by induction on the structure of C_v^i and follows the reasoning in [21]. The base case is trivial to show. For the inductive step, there are three cases, namely that the pivot variable x of vertex v is either in $\bar{b} \cup \bar{c}$ or in $\bar{a} \cup \bar{d}$ or in $\bar{g} \cup \bar{f} \cup \bar{e}$. See Figure 1.

By the definition of the resolution refutation graph, each non-root vertex has two predecessors v_1 and v_2. It can be shown that regardless of each of the three cases, whenever $\sigma(C_v^i) = \bot$ either predecessor branch evaluates to false for both the intermediate component-oriented interpolant and the predecessor vertex, e.g. $\sigma(v_1) = \bot$ and $\sigma(C_{v_1}^i) = \bot$. Then by induction, Equation (3) can be concluded. □

Contrary to Craig interpolation, we cannot conclude $\sigma(C_v^i) = \top \implies \exists b \in B :: \sigma(b) = \bot$ and thus preserve the unsatisfiability of Equation (1) using the

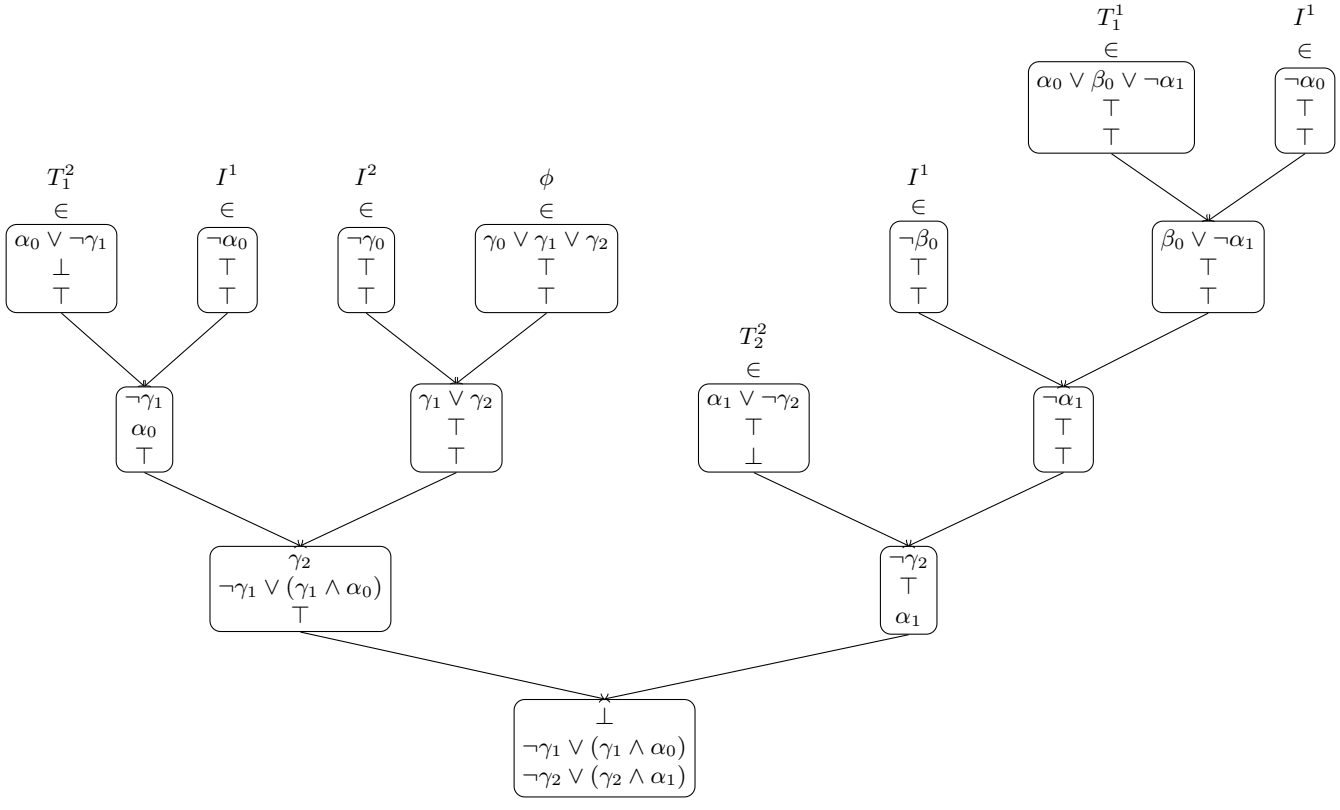


Fig. 2: An annotated resolution refutation graph from our two-step BMC unrolling of Equation (2). Each node contains three lines. The first line identifies the vertex $v \in V$. The second and third lines are respectively C_v^1 and C_v^2 obtained through the application of Definition 1. The root nodes are annotated with $\in$, meaning that the clause identifying the vertex is part of the formula, e.g. $\neg\alpha_0 \in I^1$.

component-oriented interpolant. It could also be that whenever the component-oriented interpolant evaluates to true, a clause in $T_{\neq i}^{\neq p}$ evaluates to false. Or that a clause in $I^{\neq p}$ evaluates to false. In that sense, the component-oriented interpolant is significantly weaker than a Craig interpolant. It is however strong enough for our practical purposes, as is demonstrated later in Section 4.

The component-oriented interpolant $C_\perp^i$ only derives an interpolated environment for transition step i, i.e. $T_i^{\neq p} \implies C_\perp^i$. By substitution of the occurring variables to current and successor-state variables, it can be used as a transition function for the unbounded case. This holds for each $1 \leq i \leq k$. So in general, the following definition and theorem are applicable:

Definition 2 (Interpolated Environment). *Let the component-oriented interpolants $C_\perp^1, \ldots, C_\perp^k$ be derived from the resolution refutation graph of Equation* (1) *using Definition 1. The component-oriented interpolated environment transition function, defined as E^p, can be derived as such:*

$$E^p = \bigwedge_{i=1}^{k} C_\perp^i(\bar{r}_{i-1}, \bar{r}_i)[\bar{r}_{i-1}/\bar{s}, \bar{r}_i/\bar{s}']$$

Theorem 2 (Over-Approximation by Interpolated Environment). *Let E^p be given according to Definition 2. It then follows that*

$$T^{\neq p} \implies E^p \tag{4}$$

Proof. This follows from Lemma 1 which shows that $T_i^{\neq p} \implies C_\perp^i$. As $var(C_\perp^i) \subseteq var(T_i^{\neq p})$, it follows that $T^{\neq p} \implies C_\perp^i[\bar{r}_{i-1}/\bar{s}, \bar{r}_i/\bar{s}']$. By composition of implications of each i, Equation (4) follows. □

Example 5 (Transition Function from Component-Oriented Interpolants). Let us apply Definition 2 on the component-oriented interpolants $C_\perp^1$ and $C_\perp^2$ from our running example (cf. Figure 2). That is, we substitute the occurring timed variables into current and next-state variables:

$$\begin{aligned} E^p &= C_\perp^1[\alpha_0/\alpha, \gamma_1/\gamma'] \wedge C_\perp^2[\alpha_1/\alpha, \gamma_2/\gamma'] \\ &= (\neg\gamma_1 \vee (\gamma_1 \wedge \alpha_0))[\alpha_0/\alpha, \gamma_1/\gamma'] \wedge (\neg\gamma_2 \vee (\gamma_2 \wedge \alpha_1))[\alpha_1/\alpha, \gamma_2/\gamma'] \\ &= \neg\gamma' \vee (\gamma' \wedge \alpha) \\ &= \alpha \vee \neg\gamma' \end{aligned}$$

Since we partitioned $T^{\neq p} = T^2$ in Equation (2), it follows from Equation (4) that $T^2 \implies \alpha \vee \neg\gamma'$. This is clearly evident from the definition of T^2 in Example 2, where the interpolated environment is in fact the third clause. Component-oriented interpolation therefore reduces the original transition function to $1/6^{\text{th}}$ of the amount of clauses. *(end of example)*

Applications Theorem 2 can be applied in several ways. We elaborate on a few possible applications in the following.

Manual inspection for example becomes more feasible. Models as large and complex as the one mentioned in Section 1 are labor-intensive to analyze manually, yet this is often the pragmatical approach by industry for verifying/validating involved requirements. The interpolated environment of Theorem 2 can support this. Assume one is intimate with a particular (set of) component(s), e.g. the power system. The remainder components can be viewed as a rather unfamiliar environment that can be abstracted in terms of variables shared with the power system. Such an abstraction is significantly smaller and thus eases manual inspection. The abstraction is cheap to compute, as it can be obtained for a bound as small as $k = 1$, although a larger k is preferable since this possibly strengthens the accuracy of the environment.

It can also be used as an abstraction method in model checking. Consider the invariant checking case and assume a tractable bound k for $M \models_k \phi$. Yet it is unclear whether it holds beyond the tractable bound k. One can pick a component p and use Theorem 2 to over-approximate the remainder to E^p. Heuristically it is wise to include at least the component directly referred by ϕ as p, as they directly affect the property of interest. Then the smaller model $(I^p, T^p \wedge E^p)$ can be subjected to unbounded model checking to verify $M \models \phi$. An example of such an algorithm is discussed later in Section 4. Note that the transition function $T^p \wedge E^p$ could be too weak. Thus, if a counterexample is found during unbounded model checking, one has to distinguish whether it is a false-negative due to over-approximation of E^p, or whether it is a counterexample that also occurs in the original model. Techniques from CEGAR (counterexample guided abstraction refinement) [10, 7] can be utilized for this. Theorem 2 can also supplement existing CEGAR techniques, as it can generate computationally cheap abstractions.

Partial model reverification is also a suitable application. In monolithic model checking, refinements or changes of the model require a full reverification round. Theorem 2 can speed this up. Assume only a part of the model is changed, for example component p. The unchanged environment can be interpolated from previous verifications. The resulting interpolated environment is smaller in size. Instead of reverifying the full model, the modified component p and the interpolant of the unchanged environment E^p can be used. Since reverification with the smaller model $(I^p, T^p \wedge E^p)$ is likely to be faster, as less variables are present, it can be used less reluctantly upon changes to component p, thus providing more direct and continuous feedback during the construction of the model. Note that here over-approximation might cause false counterexamples as well and therefore warrant the use of CEGAR techniques.

4 Evaluation on Satellite Case Studies

We developed a prototype implementation utilizing Theorem 2 in NuSMV 2.5.4 and applied it to industrially-sized models of satellite case studies reported in

[11]. The resulting data provides an indication of the quality of the abstraction, as well as its effectiveness when used for manual inspection or (re)verification.

Tool Implementation The prototype implementation is an extension of NuSMV 2.5.4 [4]. We reused NuSMV's data structures and functionality for representing and handling propositional formulae. The SAT-solving was performed by MiniSAT 1.14p. We deliberately chose version 1.14p over newer versions, as it is – at the moment – the only publicly available version that can generate resolution refutation graphs upon unsatisfiability. Additionally, NuSMV has a preexisting integration with MiniSAT which we extended for handling those graphs. The models are expressed in SLIM, a formalized dialect of AADL. We used the SLIM-to-SMV translator built in the COMPASS toolset for obtaining their SMV representations [2].

Case Description We ran our evaluation with two large industrially-sized models. They are system-software models based on design data of Earth-orbiting satellites in development.

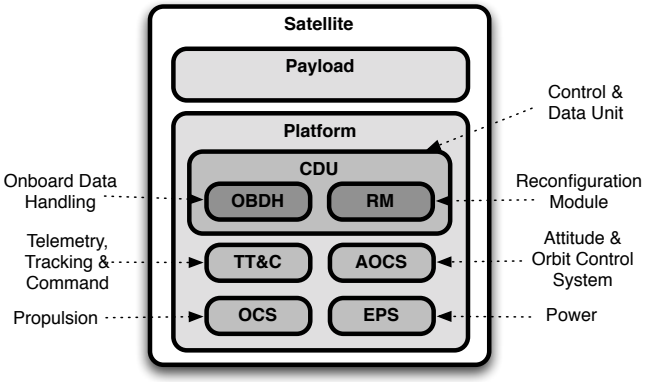


Fig. 3: Decomposition of the PDR satellite model.

The first model is from the case study reported in [11]. We call it the PDR satellite platform model. It was constructed from the design data available during the satellite's preliminary design review (PDR). Its global decomposition into subsystems is shown in Figure 3. The OCS consists of a series of controllable thrusters for orbital corrections. The AOCS is a control system consisting of several kinds of sensors for acquiring and maintaining a correct attitude and orbit. The CDU is the main computer. The EPS consists of solar arrays and batteries for powering the satellite and the TT&C is the radio communication interface for ground control on Earth. The focus of the PDR model is the relation of the system's nominal behavior, its erroneous behavior (e.g. faults) and the subsequent behaviors resulting from the fault tolerance strategies by the fault

management design. Its nominal state space is roughly 48 million states. This number multiplies rapidly when failures are injected, thus activating failure modes and the associated fault management strategies. The case is modeled in our AADL dialect and comprises 3,831 lines of code, not counting comments.

The second model is a refinement of the PDR model. We call it the CDR model. It was crafted from the design data available during the critical design review (CDR) of the same satellite mission. During the CDR, more design details have been decided upon. It is estimated that the amount of design data increased twofold. The CDR model's nominal behavior state space nevertheless counts 2,341 states thanks to the effective modeling lessons learned from the PDR case study. The CDR model is however more detailed, more complex and more difficult to analyze. Akin to the PDR model, its state space multiplies once failures are injected. It is composed of 6,357 lines of AADL code, not counting comments. A more detailed report of this model is currently being prepared for publication.

We considered several configurations of the PDR and the CDR models. The final configurations outlined below are known to require a bound $k > 1$ for proving or disproving the invariant property of interest [11]. The first two configurations are from the PDR model, whereas the remaining three are from the CDR model. Note that the models are based on proprietary designs. Their details are therefore not publicly available.

Model	Fault Injections	Property
PDR-1	Earth sensor failure	fail-operational flag is set
PDR-2	Propulsion failure	AOCS status flags are consistent
CDR-3	Various platform failures	not in safe mode
CDR-4	(none, i.e. nominal behaviour)	solar voltage level is consistent
CDR-5	(none, i.e. nominal behaviour)	not in safe mode

Comparison Factors All experiments were run on a Linux 64-bits machine equipped with a 2.33 GHz multi-core CPU and 32 GB RAM. We set the maximum computation time to 900 seconds. Our implementation is however single-threaded. The exploitation of the multiple cores in a multi-threaded fashion is future work.

We intended to use NuSMV's BDD-based verification as the baseline. We however quickly learned that the BDDs were ineffective on both the PDR and CDR model. BDD-based verification was a magnitude slower on PDR configurations than the other techniques we considered (see Table 1). On CDR configurations, the time for constructing the transition functions exceeded the 900 seconds by far, thus leaving no time for the verification. We therefore omit BDD-verification data and decided upon another technique as the baseline.

We used McMillan's interpolation-based unbounded model checking technique for invariants [20] instead. It starts by k-bounded model checking the property. Then it (Craig) interpolates the first transition step $C \implies I \wedge T_1$. This interpolant is a weakened characterization of one-step successor states $\bar{s}_1$. These states are added to I by variable substitution, i.e. $I \leftarrow I \vee C[\bar{s}_1/\bar{s}_0]$. The new I is a characterization of the original initial states and the one-step successor states. It

is then used to bounded model check the property up to bound k, thus reaching a search depth of $k+1$. This is repeated until a fixpoint is reached. A sketch of the algorithm can be found in [20]. It is furthermore also part of Algorithm 1 from lines 5 to 8, which we shall explain shortly after. We implemented the algorithm in NuSMV as there was no pre-existing implementation. The interpolation scheme we implemented is by McMillan as well [19] and it has been studied thoroughly for use in this setting [10].

The component-oriented interpolation technique has been casted into a verification scheme. We heuristically chose the components p by selecting those directly referred in the property. Given this, the remaining procedure is shown in Algorithm 1. Intuitively, it obtains an interpolated environment (line 3), which is then used in an inner reachability analysis (lines 5-8) until a fixpoint is encountered (line 8), meaning that the property holds. Otherwise, the bound is increased in the hope for a stronger interpolated environment (line 9). The overall algorithm can terminate in two ways: either a concrete (and real) counterexample is eventually found at depth k while executing line 2, or reachability analysis on the over-approximated model reaches a fixpoint without violation of the property (line 8). Note that even though any inner reachability algorithm could be used, we employed McMillan's interpolant-based invariant checking algorithm here. This is mainly for efficiency reasons of staying in a SAT-based context. If for example BDD-based reachability techniques were used, we would have to convert E^p, I^p and T^p to BDDs, resulting in additional overhead.

Algorithm 1 Component-Oriented Interpolation-based Invariant Checking.

1: $k \leftarrow 1$
2: **while** $\neg\phi \wedge I_0^p \wedge T_1^p \wedge \cdots \wedge T_k^p \wedge I_0^{\neq p} \wedge T_1^{\neq p} \wedge \cdots \wedge T_k^{\neq p}$ is unsatisfiable **do**
3: $\quad E^p \leftarrow$ component-oriented interpolant of $I_0^{\neq p} \wedge T_1^{\neq p} \wedge \cdots \wedge T_k^{\neq p}$
4: $\quad R \leftarrow I^p$
5: $\quad$ **while** $R \wedge T_1^p \wedge E_1^p \wedge \cdots \wedge T_k^p \wedge E_k^p \neg\phi$ is unsatisfiable **do**
6: $\quad\quad C \leftarrow$ Craig interpolant of $R \wedge T_1^p \wedge E_1^p$
7: $\quad\quad$ **if** $C \wedge \neg R$ is satisfiable **then** $R \leftarrow R \vee C$
8: $\quad\quad$ **else**[no new states explored] **return** ϕ holds
9: $\quad k \leftarrow k+1$
10: **return** counterexample extracted from the satisfying assignment

Experiment Data and Discussion A summary of the experiment data is presented in Table 1. We kept track of the depth needed to determine whether the property holds or whether there exists a counterexample. This depth k is the column "Bound" in Table 1. A smaller bound indicates a faster convergence of the abstraction.

The results indicate that the CDR model has a higher complexity than the PDR model. This was expected due to the doubling of design details in the CDR design data. The results furthermore indicate that the verification by the

Table 1: Summary of verification outcome, needed bound k, verification time and peak memory consumption for McMillan's interpolation-based invariant checking (MCM) and the component-oriented interpolation-based invariant checking (COMP).

Case	Technique	Outcome	Bound	Time (sec)	Mem (Mb)
PDR-1	MCM	counterexample	3	2.42	95.9
	COMP	counterexample	3	3.52	111.9
PDR-2	MCM	counterexample	2	1.77	92.0
	COMP	counterexample	2	2.28	100.4
CDR-3	MCM	counterexample	11	486.06	651.0
	COMP	counterexample	11	338.56	865.5
CDR-4	MCM	holds	4	7.10	125.7
	COMP	holds	3	7.00	138.0
CDR-5	MCM	holds	7	69.20	171.5
	COMP	holds	3	8.10	137.0

component-oriented interpolation method is competitive. This is in particular visible for CDR-3 and CDR-5, where the computation time is significantly better. The reason for this is the needed bound k. A small k appears to suffice for a quality abstraction. Note that these measures cannot be trivially generalized. Timings depend heavily on the used SAT-solver, in particular on the heuristics it employs, the possibly imposed randomness influenced by the order of clauses, or by the choice of the target system. These factors are inherent to the nature of current-day SAT-solvers. The numbers should therefore be interpreted as indications.

While the experiment data indicate a positive influence of component-oriented interpolation, we suspect that the way it is used in this evaluation suffers from double abstraction. Observe that in Algorithm 1 two abstraction techniques are jointly used. The first comes from component-oriented interpolation which is a possible source of false counterexamples. The second comes from inner reachability analysis (line 6 of Algorithm 1), which may add further false counterexamples. Each abstract counterexample of the inner reachability analysis turns the while condition in line 5 of Algorithm 1 to false, leading to an unnecessary increase of k. An exact, rather than an approximative, inner reachability would resolve this. We are however not aware of any exact unbounded SAT-based reachability techniques. BDD-based techniques might work, but we suspect that the repeated conversion from SAT-based data structures to BDDs would add too much overhead to be competitive. Hence, we leave further optimization in this area as future work.

As elaborated in Section 3, there are other applications for the component-oriented interpolated environment, like manual inspection or partial model reverification. Their algorithms look slightly different from Algorithm 1, but the essential computational steps are there. For manual inspection, the emphasis is on the

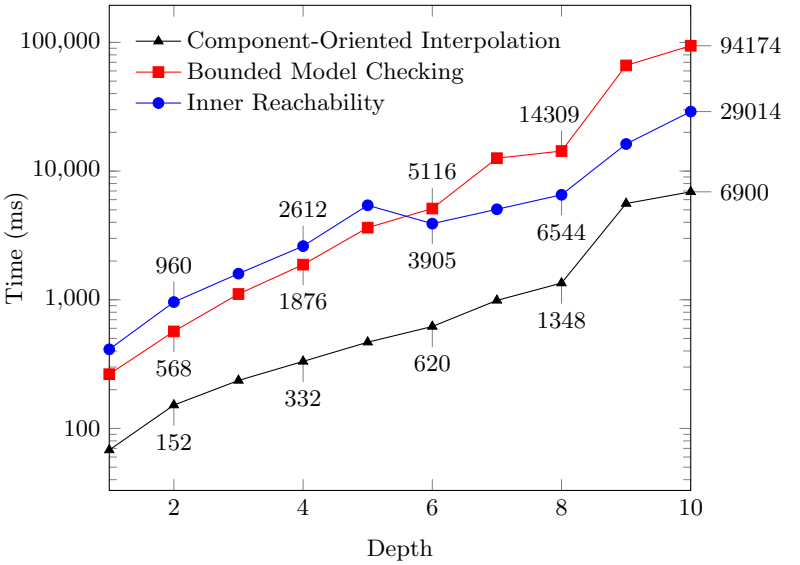


Fig. 4: Plots of time spent in milliseconds at each depth k for bounded model checking, component-oriented interpolation and inner reachability (respectively lines 2, 3 and 5-8 of Algorithm 1) on experiment configuration CDR-3 (checking avoidance of safe mode in the presence of various platform failures).

bounded model checking step (line 2) and component-oriented interpolation (line 3). For reverification on the other hand, the emphasis lies on inner reachability (lines 5-10). To extrapolate the effectiveness of component-oriented interpolation for those two applications, we logged the time spent on parts of Algorithm 1 at each step k. A summary is shown in Figure 4 for experiment configuration CDR-3, which is representative for the other experiment configurations. Note that the y-axis has a logarithmic scale. The bottom line is that the step for constructing the interpolated environment has little impact on the overall running time, as it only takes a fraction of the time spent on bounded model checking and inner reachability. Note that the (most time-costly) bounded model checking step is avoided for partial model reverification, whereas inner reachability is omitted for manual inspection. This is where time is saved for the overall analysis.

5 Related Work

There is a huge body of work on compositional analysis in literature. In the introduction, Section 1, we have briefly explained the main differences between our work and what has been reported in literature. We continue this discussion in this section.

Contrary to many works from the Owicki-Gries paradigm, which often make distinctions between global and local variables, our work fits the *shared variables*

paradigm. Global variables are typical to parallel systems, where multiple (identical) concurrent processes are active at the same time and interact with each others through a small set of global variables. The technique by [7] essentially abstracts and redefines the concurrent processes in terms of those global variables and calls this notion the split invariant. The technique by [13] is similar to that, as well as the technique by [18]. Our work expresses environments in terms of those variables that are *shared* with other components, thus not requiring a model structure in which global variables are explicitly defined by the user. The shared variable paradigm is therefore a generalization of the global/local variables paradigm.

In this work, we show, so far, how the environments are interpolated using the unsatisfiability proof from bounded model checking of an invariant. This can be extended to a larger class of properties. Akin to [8], where the notion of a process environment is described, the interpolated environments, as it currently is formalized, can also be used to verify safety and liveness properties. In fact, our notion of the interpolated environment was inspired by that. We however foresee that the interpolated environment is not strong enough for verifying liveness properties and that techniques from CEGAR are a necessity here. Particular techniques in this setting are by [8] and [17]. As these works were carried out for the global/local variables paradigm, it requires further investigation how these techniques are extendable to the shared variables paradigm. For the moment, Algorithm 1 naively increases the bound k as a refinement step without checking whether an abstract counterexample is a false-positive or not.

With regard to rely/assume guarantee/provide reasoning, we remarked in the introduction that the automated approaches are closely related to ours. In [14], a technique based on automata determinization is described to generate weakest assumptions. In subsequent work [5], assumptions are learned using an automaton learning algorithm, like L*. Our work is a twist on this, as we describe a method using Craig interpolation.

Rely/assume guarantee/provide reasoning has also been applied to AADL models, like for example [6]. The scope and semantic base of [6] differs from ours. Our satellite models for example are expressed in a formalized dialect of AADL by [2]. It is designed to provide a rigorous and coherent semantics for a system's nominal, erroneous and degraded behavior. The work by [6] appears to focus only on the nominal behavior. Furthermore, their approach does not generate assumptions nor guarantuees as we do, but rather provides a tool-supported reasoning framework over them once provided by the user.

Interpolation in model checking has become an active field since the pioneering work by [19]. It has been further studied since, covering applications such as a monolithic abstraction of the transition relation [16], or more theoretical investigations studying the differences in strength of interpolants as a consequence of a chosen interpolation generation scheme [10]. We were inspired by [20] and devised a modified interpolation scheme that is suitable for compositional reasoning that is reported in this paper.

6 Conclusions

We have described and experimentally evaluated a technique for deriving an abstract environment transition condition from a component-oriented model using a Craig interpolation-inspired method. We call it the component-oriented interpolant. It particularly fits models where highly cohesive components communicate through shared variables, which is a generalization over the global/local variables setting. To our knowledge it is the first application of interpolation-like techniques to exploit a model's composition of components.

Through our work, we identified several open points for future work. In particular a study of the component-oriented interpolant's strength would be interesting. We know from Lemma 1 that the component-oriented interpolant over-approximates, but we do not know how strong it is such that the property still holds up to bound k. This is in contrast to classical Craig interpolation, where its interpolant does have this property. It requires further study to understand how and whether the component-oriented interpolant can be strengthened. Inspiration can be drawn from the strengthening techniques for classical interpolation, where the reordering of vertices in the resolution refutation proof and asymmetric interpolation schemes have been studied for this purpose [10].

Studying the strength of the component-oriented interpolant also benefits its suitability for verifying more expressive properties, like safety and liveness properties. We estimate that the component-oriented interpolation scheme of Definition 2 overapproximates too much for that purpose and thus the straightforward usage of the interpolated environment to safety/liveness properties would yield too many false-positive counterexamples.

Furthermore, we indicated in Section 4 that Algorithm 1 is rudimentary. It suffers from double abstraction, because it does not perform an exact inner reachability analysis using the interpolated environment. Exact methods would alleviate that. Especially ones that work in a SAT-based context are preferable, because that would avoid the overhead of converting the used data-structures. This is open for further investigation.

We have made our implementation available on `http://www-i2.informatik.rwth-aachen.de/~nguyen/coi/` under the LGPL open source license.

Acknowledgements This work was partially supported by ESA/ESTEC (contract no. 4000100798), Thales Alenia Space (contract no. 1520014509/01) and EU FP7-ICT D-MILS (reference 318772).

References

1. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic Model Checking without BDDs. In: Proc. of Tools and Algorithms for Construction and Analysis of Systems (TACAS), LNCS, vol. 1384, pp. 193–207. Springer (1999)
2. Bozzano, M., Cimatti, A., Katoen, J.-P., Nguyen, V.Y., Noll, T., Roveri, M.: Safety, Dependability and Performance Analysis of Extended AADL Models. Computer Journal 54(5), 754–775 (2011)

3. Buss, S.R.: Propositional Proof Complexity. Computational Logic (1997)
4. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In: Proc. of Computer Aided Verification (CAV), LNCS, vol. 2404, pp. 359–364. Springer (2002)
5. Cobleigh, J.M., Giannakopoulou, D., Păsăreanu, C.S.: Learning Assumptions for Compositional Verification. In: Proc. of Tools and Algorithms for the Construction and Analysis of Systems (TACAS), LNCS, vol. 2619, pp. 331–346. Springer (2003)
6. Cofer, D.D., Gacek, A., Miller, S.P., Whalen, M.W., LaValley, B., Sha, L.: Compositional Verification of Architectural Models. In: Proc. of NASA Formal Methods Symposium (NFM), LNCS, vol. 7226, pp. 126–140. Springer (2012)
7. Cohen, A., Namjoshi, K.S.: Local Proofs for Global Safety Properties. Formal Methods in System Design 34(2), 104–125 (2009)
8. Cohen, A., Namjoshi, K.S.: Local Proofs for Linear-Time Properties of Concurrent Programs. In: Proc. of Computer Aided Verification (CAV), LNCS, vol. 5123, pp. 149–161. Springer (2008)
9. Craig, W.: Three Uses of the Herbrand-Gentzen Theorem in Relating Model Theory and Proof Theory. The Journal of Symbolic Logic 22(3), 269–285 (1957)
10. D'Silva, V., Kroening, D., Purandare, M., Weissenbacher, G.: Interpolant Strength. In: Proc. of Verification, Model Checking, and Abstract Interpretation (VMCAI), LNCS, vol. 5944, pp. 129–145. Springer (2010)
11. Esteve, M.-A., Katoen, J.-P., Nguyen, V.Y., Postma, B., Yushtein, Y.: Formal Correctness, Safety, Dependability and Performance Analysis of a Satellite. In: Proc. of Software Engineering (ICSE), pp. 1022–1031. IEEE (2012)
12. Feiler, P.H., Gluch, D.P.: *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. Addison-Wesley (2012)
13. Flanagan, C., Qadeer, S.: Thread-Modular Model Checking. In: Proc. of Model Checking Software (SPIN), LNCS, vol. 2648, pp. 213–224. Springer (2003)
14. Giannakopoulou, D., Pasareanu, C.S., Barringer, H.: Assumption Generation for Software Component Verification. In: Proc. of Automated Software Engineering (ASE), pp. 3–12. IEEE (2002)
15. Gupta, A., Popeea, C., Rybalchenko, A.: Threader: A Constraint-Based Verifier for Multi-Threaded Programs. In: Proc. of Computer Aided Verification (CAV), LNCS, vol. 6806, pp. 412–417. Springer (2011)
16. Jhala, R., McMillan, K.L.: Interpolant-Based Transition Relation Approximation. In: Proc. of Computer Aided Verification (CAV), LNCS, vol. 3576, pp. 39–51. Springer (2005)
17. Malkis, A., Podelski, A., Rybalchenko, A.: Thread-Modular Counterexample Guided Abstraction Refinement. In: Proc. of Static Analysis (SAS), LNCS, vol. 6337, pp. 356–372. Springer (2010)
18. Malkis, A., Podelski, A., Rybalchenko, A.: Thread-modular Verification is Cartesian Abstract Interpretation. In: Proc. of Theoretical Aspects of Computing (ICTAC), LNCS, vol. 4281, pp. 183–197. Springer (2006)
19. McMillan, K.: Interpolation and SAT-Based Model Checking. In: Proc. of Computer Aided Verification (CAV), LNCS, vol. 2725, pp. 1–13. Springer (2003)
20. McMillan, K.L.: Applications of Craig Interpolants in Model Checking. In: Proc. of Tools and Algorithms for the Construction and Analysis of Systems (TACAS), LNCS, vol. 3440, pp. 1–12. Springer (2005)
21. Nguyen, V.Y. "Trustworthy Spacecraft Design Using Formal Methods". PhD thesis. RWTH Aachen University, Germany, 2012