Leveraging Interpolant Strength in Model Checking

Simone Fulvio Rollini[1], Ondrej Sery[1,2], and Natasha Sharygina[1]

[1] Formal Verification Lab, University of Lugano, Switzerland
{simone.fulvio.rollini,ondrej.sery,natasha.sharygina}@usi.ch
http://verify.inf.usi.ch
[2] Dept. of Distributed and Dependable Systems, Faculty of Mathematics and Physics
Charles University in Prague, Czech Republic
http://d3s.mff.cuni.cz

Abstract Craig interpolation is a well known method of abstraction successfully used in both hardware and software model checking. The logical strength of interpolants can affect the quality of approximations and consequently the performance of the model checkers. Recently, it was observed that for the same resolution proof a complete lattice of interpolants ordered by strength can be derived.
Most state-of-the-art model checking techniques based on interpolation subject the interpolants to constraints that ensure efficient verification as, for example, in transition relation approximation for bounded model checking, counterexample-guided abstraction refinement and function summarization for software update checking. However, in general, these verification-specific constraints are not satisfied by all possible interpolants.
The paper analyzes the restrictions within the lattice of interpolants under which the required constraints are satisfied. This enables investigation of the effect of the strength of interpolants on the particular techniques, while preserving their soundness. As an additional benefit, combination of this result with proof manipulation procedures allows the use of optimized solvers to generate interpolants of different strengths for various model checking techniques.

1 Introduction

Craig interpolants [4] are commonly used for abstraction in hardware and software model checking. Recently, it was shown [5] that for the same resolution proof a complete lattice of interpolants ordered by the implication relation, i.e., *strength*, can be systematically derived. The strength of the interpolants may influence speed of convergence of the model checking algorithms as well as the amount of spurious behaviors that require refinement. The result in [5] shows that there are interpolants of different strengths to choose from. However, [5] opens two new research problems. First, it is not clear how to choose the right interpolation algorithm for a particular model checking application. Second, if a concrete application puts additional constraints on the interpolants, it is not

clear if the choice among interpolants of various strengths is restricted and how much.

This paper presents a theoretical solution to the second problem. We identify two classes of common interpolation-based model checking approaches that indeed put additional requirements on the interpolants. Then we formally determine and prove the restrictions for both classes on the choice of the interpolants strength under which these requirements are satisfied.

The first class of approaches concerns simultaneous abstraction by multiple interpolants. In this scenario, we have an unsatisfiable formula in the form of a conjunction of subformulae. From the proof of unsatisfiability, we compute interpolants abstracting the different conjuncts. The additional requirement (*Req1*) is ensuring unsatisfiability of the original formula with multiple conjuncts replaced by the corresponding interpolants. A notable example of this setting is the approach presented in [8], where the abstract transition relation is iteratively refined using interpolants. The authors notice the requirement imposed on the interpolants, and observe it satisfied while implicitly assuming the use of the interpolation algorithm of [10]. However, [8] is restricted to a single interpolant generated by this algorithm. Our solution overcomes this limitation by showing formally how to generate interpolants of different strength that satisfy the requirement. Interestingly, we discovered that not all interpolants do. Another application is software update checking, where the formula represents the original program with different conjuncts representing different functions as, e.g., in [16]. When a subset of functions is updated due to code changes and fixes, this approach checks if the interpolants remain valid abstractions of the new function bodies. This is a local check to show that a formula representing the new function body still implies the corresponding interpolant. If the check succeeds, unsatisfiability of the formula with multiple conjuncts replaced by the corresponding interpolants (Req1) provides correctness of the updated system without the need to check the entire formula of the updated system again.

The second class of model checking methods is typical of counterexample-guided abstraction refinement (CEGAR) [3]. Given a spurious error trace, the goal is to annotate nodes of an abstract reachability tree with an inductive sequence of formulae that together rule out the trace. The spurious error trace is represented by an unsatisfiable path formula, constructed from the SSA form of the trace. An interpolant is computed from the prefix and suffix of the trace for every location along the error trace. Here, the additional requirement (*Req2*) is that the resulting sequence of interpolants is inductive, i.e., that for every location, the current interpolant conjuncted with the precise representation of the instruction at the location implies the next interpolant along the trace. For example, this reasoning is crucial for the refinement techniques used in Blast [1] and Impact [12]. In general, however, such a sequence is not inductive. Therefore, the authors restrict themselves to specific proof systems to ensure this property [7] ruling out not only the choice of interpolants of varying strength but also the possibility of using state-of-the-art solvers. Other authors even require multiple solver calls to ensure a similar property [6].

Contribution. The main contribution is a theoretical formulation and proof of the constraints within the lattice of interpolants, such that Req1 and Req2 are satisfied.

In particular, building on [5], this work analyzes a whole family of interpolation procedures from which the lattice is generated. The analysis yields two interesting results: (1) we prove that every member of the family that produces interpolants stronger than the ones given by Pudlák's algorithm [14] complies with Req1, and (2) we identify a subset, within the family of procedures (by means of the logical constraints that characterize it), that satisfies Req2. These results allow for the systematic study of how interpolants strength affects state-of-the-art model checking techniques, while preserving their soundness.

Moreover, since our results are not limited to the use of an ad-hoc proof system, any state-of-the-art solver can be chosen to generate proofs (if needed, post-processed by, e.g., the techniques of [2]) from which the interpolants are computed. Additionally, proof manipulation procedures, as in [15], can be applied to alter the size and the strength of interpolants in the various model checking applications.

2 Preliminaries

2.1 Craig Interpolation

Craig interpolants [4], since the seminal work by McMillan [9], have been extensively applied in SAT-based model checking and predicate abstraction [11]. Formally, given an unsatisfiable conjunction of formulae $A \wedge B$, an interpolant I is a formula that is implied by A (i.e., $A \rightarrow I$), is unsatisfiable in conjunction with B (i.e., $B \wedge I \rightarrow \bot$) and is defined on the common language of A and B. The interpolant I can be thought of as an over-approximation of A that still conflicts with B.

Several state-of-the art approaches exist to generate interpolants in an automated manner; the most successful techniques derive an interpolant for $A \wedge B$ (in certain proof systems) from a proof of unsatisfiability of the conjunction. This approach grants two important benefits: the generation can be achieved in linear time w.r.t. the proof size, and interpolants themselves only contain information relevant to determine the unsatisfiability of $A \wedge B$. In particular, Pudlák [14] investigates interpolation in the context of resolution systems for propositional logic, while McMillan [10] addresses both propositional logic and a quantifier free combination of the theories of uninterpreted functions and linear arithmetic. All these techniques adopt recursive algorithms, which initially set *partial interpolants* for the axioms. Then, following the proof structure, they deduce a partial interpolant for each conclusion from those of the premises. The partial interpolant of the overall conclusion is the interpolant for the formula.

2.2 Resolution Proofs

Assuming a finite set of propositional variables, a literal is a variable, either with positive (p) or negative ($\overline{p}$) polarity. A clause C is a finite disjunction of literals;

a formula φ in conjunctive normal form (CNF) is a finite conjunction of clauses. A *resolution proof of unsatisfiability* (or *refutation*) of a formula ϕ in CNF is a tree such that the leaves are the clauses of ϕ, the root is the empty clause $\bot$ and the inner nodes are clauses generated by means of the *resolution rule*:

$$\frac{C^+ \vee p \qquad C^- \vee \overline{p}}{C^+ \vee C^-}$$

where $C^+ \vee p$ and $C^- \vee \overline{p}$ are the *antecedents*, $C^+ \vee C^-$ the *resolvent* and p is the *pivot* of the resolution step.

2.3 Strength of Interpolants

D'Silva et al. [5] generalize the algorithms by Pudlák [14] and McMillan [10] (as well as the approach dual to McMillan's, which we will call McMillan′) for propositional resolution systems by introducing the notion of *labeled interpolation system*, focusing on the concept of *interpolant strength* (a formula ϕ is stronger than χ whenever $\phi \rightarrow \chi$). They present an analysis and a comparison of the systems corresponding to the three algorithms, together with a method to combine labeled systems in order to obtain weaker or stronger interpolants from a given proof of unsatisfiability. Throughout the paper we will adopt the notation of [5], adapted as necessary.

Given a refutation of a formula $A \wedge B$, a variable p can appear as literal only in A, only in B or in both conjuncts; p is respectively said to have *class* A, B or AB. The authors define a *labeling* L as a mapping that assigns a *color* among $\{a, b, ab\}$ independently to each variable in each clause (since a variable cannot have two occurrences in a clause, this is equivalent to coloring literals). The set of possible labelings is restricted by ensuring that class A variables receive color a and class B variables receive color b; freedom is left for AB variables to be colored either a, b or ab.

In [5], a *labeled interpolation system* is defined as a procedure Itp (shown in Table 1) that, given A, B, a refutation R of $A \wedge B$ and a labeling L, outputs a partial interpolant $Itp_L(A, B, R, C)$ for any clause C in R; this depends on the clause being in A or B (if leaf) and on the color of the pivot associated with the resolution step (if inner node). $Itp_L(A, B, R)$ represents the interpolant for $A \wedge B$, that is $Itp_L(A, B, R) \triangleq Itp_L(A, B, R, \bot)$[3]. We will omit the parameters whenever clear from the context.

In Table 1, $C \downharpoonright \alpha$ denotes the restriction of a clause C to the literals of color α. $p : \alpha$ indicates that variable p has color α. By $C[I]$ we represent that clause C has a partial interpolant I. I^+, I^- and I are the partial interpolants respectively associated with the two antecedents and the resolvent of a resolution step: $I^+ \triangleq Itp_L(C^+ \vee p)$, $I^- \triangleq Itp_L(C^- \vee \overline{p})$, $I \triangleq Itp_L(C^+ \vee C^-)$.

An operator $\sqcup$ allows to determine the color of a pivot p, taking into account that p might have different colors α and β in the two antecedents: $\sqcup$ is idempotent, symmetric and defined by $a \sqcup b \triangleq ab$, $a \sqcup ab \triangleq ab$, $b \sqcup ab \triangleq ab$.

[3] As customary, we use $\triangleq$ to characterize a definition, while $\equiv$ a correspondence.

Table 1: Labeled interpolation system Itp_L

Leaf: $C\,[I]$
$I = \begin{cases} C\lfloor b & \text{if } C \in A \\ \neg(C\lfloor a) & \text{if } C \in B \end{cases}$

Inner node: $\dfrac{C^+ \vee p : \alpha\,[I^+] \qquad C^- \vee \overline{p} : \beta\,[I^-]}{C^+ \vee C^-\,[I]}$
$I = \begin{cases} I^+ \vee I^- & \text{if } \alpha \sqcup \beta = a \\ I^+ \wedge I^- & \text{if } \alpha \sqcup \beta = b \\ (I^+ \vee p) \wedge (I^- \vee \overline{p}) & \text{if } \alpha \sqcup \beta = ab \end{cases}$

Table 2: Pudlák's interpolation system Itp_P

Leaf: $C\,[I]$
$I = \begin{cases} \bot & \text{if } C \in A \\ \top & \text{if } C \in B \end{cases}$

Inner node: $\dfrac{C^+ \vee p : \alpha\,[I^+] \qquad C^- \vee \overline{p} : \alpha\,[I^-]}{C^+ \vee C^-\,[I]}$
$I = \begin{cases} I^+ \vee I^- & \text{if } \alpha = a \\ I^+ \wedge I^- & \text{if } \alpha = b \\ (I^+ \vee p) \wedge (I^- \vee \overline{p}) & \text{if } \alpha = ab \end{cases}$

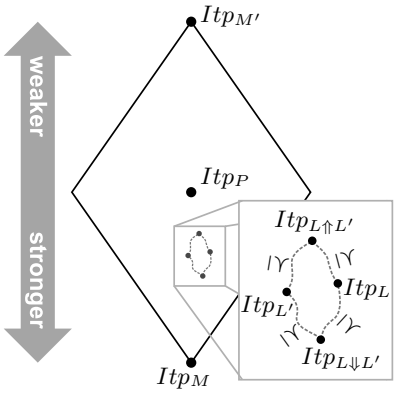

Figure 1: Lattice of labeled interpolation systems

The systems corresponding to McMillan, Pudlák and McMillan′'s interpolation algorithms will be referred to as Itp_M, Itp_P, $Itp_{M'}$. Itp_L subsumes Itp_M, Itp_P and $Itp_{M'}$, obtained as special cases by coloring all the occurrences of AB variables with b, ab and a, respectively (compare, for example, Tables 1 and 2).

A total order $\preceq$ is defined over the colors as $b \preceq ab \preceq a$, and extended to a partial order over labeled systems: $Itp_L \preceq Itp_{L'}$ if, for every clause C and variable p in C, $L(p, C) \preceq L'(p, C)$. This allows the authors to directly compare the logical strength of the interpolants produced by two systems. In fact, for any refutation R of a formula $A \wedge B$ and labelings L, L' such that $L \preceq L'$, we have: $Itp_L(A, B, R) \rightarrow Itp_{L'}(A, B, R)$ and we say that Itp_L is *stronger* than $Itp_{L'}$.

Two interpolation systems Itp_L and $Itp_{L'}$ can generate new systems $Itp_{L \Uparrow L'}$ and $Itp_{L \Downarrow L'}$ by combining the labelings L and L' in accordance with $\preceq$: $(L \Uparrow L')(p, C) \triangleq \max_{\preceq}\{L(p, C), L'(p, C)\}$ and $(L \Downarrow L')(p, C) \triangleq \min_{\preceq}\{L(p, C), L'(p, C)\}$. The authors remark that the collection of labeled systems over a refutation, together with the order $\preceq$ and the operators $\Uparrow, \Downarrow$, represent a *complete lattice*,

where Itp_M is the greatest element and $Itp_{M'}$ is the least, with Itp_P being in between (see Fig. 1).

3 Simultaneous Abstraction with Interpolation

This section analyzes simultaneous abstraction of the conjuncts of an unsatisfiable formula by means of multiple interpolants. The requirement (Req1) is to guarantee that the formula obtained by replacing the conjuncts with the respective interpolants remains unsatisfiable[4]. We formally describe the problem, proving that the requirement is satisfied if the interpolants are generated using Pudlák's interpolation system, and later generalize the result to any interpolation system which is stronger than Pudlák's. We conclude by illustrating the applications to model checking.

3.1 Problem Description

As input, we assume an unsatisfiable formula ϕ in CNF, such that $\phi \triangleq \phi_1 \wedge \ldots \wedge \phi_n$ and each ϕ_i (a *partition*) is a conjunction of clauses. Given a refutation R of ϕ and a sequence of labeled interpolation systems $Itp_{L_1}, \ldots, Itp_{L_n}$, we compute a sequence of interpolants $I_1, \ldots, I_n$ from R. Viewing ϕ as an unsatisfiable conjunction of the form $A \wedge B$, each I_i is obtained by setting ϕ_i to A and all the other ϕ_j to B ($I_i \triangleq Itp_{L_i}(\phi_i, \phi_1 \wedge \ldots \wedge \phi_{i-1} \wedge \phi_{i+1} \wedge \ldots \wedge \phi_n)$). These n ways of splitting the formula ϕ into A and B will be referred to as *configurations*. We prove that $I_1 \wedge \ldots \wedge I_n \rightarrow \bot$ (requirement Req1), if for each i: $Itp_{L_i} \equiv Itp_P$[5], and then generalize to any sequence of interpolation systems stronger than Itp_P.

3.2 Proof for Pudlák's System

Pudlák's system is symmetric, i.e., $Itp_P(\phi_1, \phi_2) = \neg Itp_P(\phi_2, \phi_1)$. Thus for $n = 2$, $I_1 = \neg I_2$; it follows that $I_1 \wedge I_2 \rightarrow \bot$. We will prove that Req1 holds for $n = 3$ and can be extended to an arbitrary number of partitions n. Table 3 shows the class and the color that a variable p assumes in the three configurations, depending on presence of p in the three partitions.

Lemma 1. *For Pudlák's interpolation system,* $I_1 \wedge I_2 \wedge I_3 \rightarrow \bot$.

Proof (by structural induction). We show that for any clause C in the refutation, the conjunction of its partial interpolants in the three configurations is unsatisfiable. In accordance with Tables 1, 2, we refer to the partial interpolants of the antecedents as I^+ and I^- with a subscript i to identify the corresponding configuration.

[4] In [8], a notion of *symmetric interpolant* overlapping with Req1 is used. We avoid this name for its easy confusion with symmetry of interpolants, a known property of the interpolants generated by Itp_P.

[5] Recall that Itp_P is applied to n distinct configurations, so it may be associated with different labelings for different values of i.

Table 3: Variables coloring in Itp_P for $n = 3$

p in ?	Variable *class*, *color* for each configuration		
	$A \triangleq \phi_1$, $B \triangleq \phi_2 \wedge \phi_3$	$A \triangleq \phi_2$, $B \triangleq \phi_1 \wedge \phi_3$	$A \triangleq \phi_3$, $B \triangleq \phi_1 \wedge \phi_2$
ϕ_1	A, a	B, b	B, b
ϕ_2	B, b	A, a	B, b
ϕ_3	B, b	B, b	A, a
ϕ_1, ϕ_2	AB, ab	AB, ab	B, b
ϕ_1, ϕ_3	AB, ab	B, b	AB, ab
ϕ_2, ϕ_3	B, b	AB, ab	AB, ab
ϕ_1, ϕ_2, ϕ_3	AB, ab	AB, ab	AB, ab

Base case (leaf). A clause C can belong either to ϕ_1, ϕ_2 or ϕ_3. In each case the clause belongs to A in one configuration and to B in the other two configurations; this implies that the conjunction of its partial interpolants contains one $\bot$ element (see Table 2), which makes the conjunction unsatisfiable.

Inductive step (inner node). The inductive hypothesis (i.h.) consists of $I_1^+ \wedge I_2^+ \wedge I_3^+ \rightarrow \bot$, $I_1^- \wedge I_2^- \wedge I_3^- \rightarrow \bot$. A pivot p can either be local to a partition or shared by at least two partitions. If local, it has color a in one configuration and b in all the others; let us assume w.l.o.g. that p is local to ϕ_1. In Itp_P the partial interpolants for the three configurations are $I_1^+ \vee I_1^-$, $I_2^+ \wedge I_2^-$, $I_3^+ \wedge I_3^-$:

$$(I_1^+ \vee I_1^-) \wedge I_2^+ \wedge I_2^- \wedge I_3^+ \wedge I_3^- \leftrightarrow$$
$$(I_1^+ \wedge I_2^+ \wedge I_2^- \wedge I_3^+ \wedge I_3^-) \vee (I_1^- \wedge I_2^+ \wedge I_2^- \wedge I_3^+ \wedge I_3^-) \rightarrow^{i.h.} \bot$$

If shared, p has color b in (at most) one configuration and ab in the other ones. Let us assume w.l.o.g. that p is shared between ϕ_2 and ϕ_3. The three partial interpolants are $I_1^+ \wedge I_1^-$, $(I_2^+ \vee p) \wedge (I_2^- \vee \overline{p})$, $(I_3^+ \vee p) \wedge (I_3^- \vee \overline{p})$:

$$I_1^+ \wedge I_1^- \wedge (I_2^+ \vee p) \wedge (I_2^- \vee \overline{p}) \wedge (I_3^+ \vee p) \wedge (I_3^- \vee \overline{p}) \rightarrow^{\vee\text{introduction}}$$
$$(I_1^+ \vee p) \wedge (I_1^- \vee \overline{p}) \wedge (I_2^+ \vee p) \wedge (I_2^- \vee \overline{p}) \wedge (I_3^+ \vee p) \wedge (I_3^- \vee \overline{p}) \leftrightarrow$$
$$(p \vee (I_1^+ \wedge I_2^+ \wedge I_3^+)) \wedge (\overline{p} \vee (I_1^- \wedge I_2^- \wedge I_3^-)) \rightarrow^{\text{resolution}}$$
$$(I_1^+ \wedge I_2^+ \wedge I_3^+) \vee (I_1^- \wedge I_2^- \wedge I_3^-) \rightarrow^{\text{i.h.}} \bot$$

□

Lemma 1 can be extended to an arbitrary number of partitions:

Theorem 1. *For Pudlák's interpolation system,* $I_1 \wedge \ldots \wedge I_n \rightarrow \bot$.

Proof. Proof by structural induction as in Lemma 1.

Base case (leaf). Exactly as in the proof of Lemma 1, with n partitions instead of 3.

Inductive step (inner node). If the pivot p is local to a partition, the same argumentation of the proof of Lemma 1 holds. If p is shared, it might assume colors b (possibly in several configurations) or ab. Multiple applications of the $\vee$introduction rule and one resolution step yield the result. □

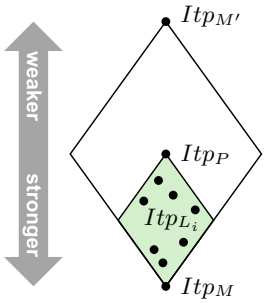

Figure 2: The interpolation systems stronger than Itp_P (colored area)

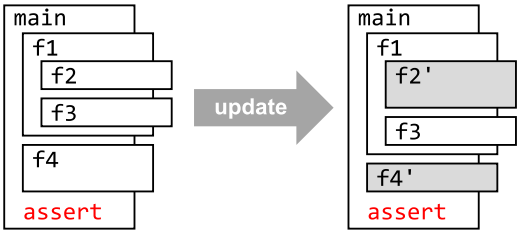

Figure 3: Update check of a program with functions `f2`, `f4` changed to `f2'`, `f4'`

3.3 Proof Generalization

Now we generalize Theorem 1 to a family of sequences of interpolation systems.

Theorem 2. *For any sequence of interpolation systems* $Itp_{L_1}, \ldots, Itp_{L_n}$*, s.t. every* Itp_{L_i} *is stronger than* Itp_P*,* $I_1 \wedge \ldots \wedge I_n \rightarrow \bot$*. (see Figure 2).*

Proof. Let $\tilde{I}_i$ be $Itp_P(\phi_i, \phi_1 \wedge \ldots \wedge \phi_{i-1} \wedge \phi_{i+1} \wedge \ldots \wedge \phi_n)$; we have that $I_i \rightarrow \tilde{I}_i$ (recall the partial order on systems defined in §2.3). This implies $I_1 \wedge \ldots \wedge I_n \rightarrow \tilde{I}_1 \wedge \ldots \wedge \tilde{I}_n$, which in turn implies $\bot$. □

The result does not necessary hold for systems weaker than Itp_P. For example, let us consider $Itp_{M'}$. A simple counterexample shows that $I_1 \wedge \ldots \wedge I_n \rightarrow \bot$ does not hold even for a trivial formula with only two partitions; let $\phi_1 \triangleq (p \vee \overline{q}) \wedge r$, $\phi_2 \triangleq (\overline{p} \vee \overline{r}) \wedge q$:

Configuration $A \triangleq \phi_1$, $B \triangleq \phi_2$:

$$\dfrac{\dfrac{\dfrac{p \vee \overline{q}\,[\bot] \qquad \overline{p} \vee \overline{r}\,[p \wedge r]}{\overline{q} \vee \overline{r}\,[p \wedge r]} \qquad r\,[\bot]}{\overline{q}\,[p \wedge r]} \qquad q\,[\overline{q}]}{\bot\,[(p \wedge r) \vee \overline{q}]}$$

Configuration $A \triangleq \phi_2$, $B \triangleq \phi_1$:

$$\dfrac{\dfrac{\dfrac{p \vee \overline{q}\,[\overline{p} \wedge q] \qquad \overline{p} \vee \overline{r}\,[\bot]}{\overline{q} \vee \overline{r}\,[\overline{p} \wedge q]} \qquad r\,[\overline{r}]}{\overline{q}\,[(\overline{p} \wedge q) \vee \overline{r}]} \qquad q\,[\bot]}{\bot\,[(\overline{p} \wedge q) \vee \overline{r}]}$$

Clearly, the interpolants $(p \wedge r) \vee \overline{q}$ and $(\overline{p} \wedge q) \vee \overline{r}$ are not mutually unsatisfiable: a partial model is $\overline{q}, \overline{r}$.

3.4 Application to Model Checking

We provide two examples of model checking algorithms, where the above setting occurs. In [8], the authors present an algorithm for iterative refinement of an abstraction of a transition relation. Initially, a coarse abstraction of the transition relation $\hat{T}_0 \equiv \mathit{true}$ is used. The abstraction $\hat{T}_i$ is used to check reachability of

error states represented by a predicate ψ from initial states represented by a predicate U. If error states are unreachable using the abstract transition relation, the system is safe. Otherwise, we have a trace from an initial state to an error state in $\hat{T}_i$ of length n, for some n. Then, reachability of the error states in n steps is checked using the precise transition relation T. For this purpose, a precise bounded model checking formula is constructed and checked for satisfiability[6]:

$$U^{\langle 0\rangle} \wedge T^{\langle 0\rangle} \wedge T^{\langle 1\rangle} \wedge \ldots \wedge T^{\langle n-1\rangle} \wedge \psi^{\langle n\rangle}$$

If satisfiable, a real error is found and the error trace is extracted from the satisfying assignment. If unsatisfiable, the corresponding interpolants are extracted from the refutation and used to strengthen the abstraction:

$$\hat{T}_{i+1} \triangleq \hat{T}_i \wedge I_1^{\langle 0\rangle} \wedge I_2^{\langle -1\rangle} \wedge \ldots \wedge I_n^{\langle -n+1\rangle}$$

The fact that $\hat{T}_{i+1} \rightarrow I_j^{\langle -j+1\rangle}$, for $1 \leq j \leq n$, and the requirement Req1 yield:

$$U^{\langle 0\rangle} \wedge \hat{T}_{i+1}^{\langle 0\rangle} \wedge \hat{T}_{i+1}^{\langle 1\rangle} \wedge \ldots \wedge \hat{T}_{i+1}^{\langle n-1\rangle} \wedge \psi^{\langle n\rangle} \rightarrow \bot$$

So the new transition relation $\hat{T}_{i+1}$ does not contain any error trace of length n and it is a tighter abstraction than $\hat{T}_i$. For this reason, the algorithm terminates for finite state systems if Req1 holds. Otherwise, termination is not guaranteed.

Another example concerns software update checking. Figure 3 depicts a situation where a program is being updated. Under a suitable encoding (e.g., as in [16]), safety of the program (w.r.t. assertion violation) is equivalent to unsatisfiability of a formula of the form $\phi_{main} \wedge \phi_{f1} \wedge \phi_{f2} \wedge \phi_{f3} \wedge \phi_{f4}$, where each conjunct represents one of the functions `main`, `f1`, `f2`, `f3`, `f4`. If the original program is safe, the formula is unsatisfiable and we can generate interpolants I_{main}, I_{f1}, I_{f2}, I_{f3}, I_{f4}. The requirement Req1 yields $I_{main} \wedge I_{f1} \wedge I_{f2} \wedge I_{f3} \wedge I_{f4} \rightarrow \bot$ and thus also $\phi_{main} \wedge \phi_{f1} \wedge I_{f2} \wedge \phi_{f3} \wedge I_{f4} \rightarrow \bot$. Now, to prove safety of the updated program, it suffices to show that $\phi_{f2'} \rightarrow I_{f2}$ and $\phi_{f4'} \rightarrow I_{f4}$. In other words, that the abstractions I_{f2} and I_{f4} of functions `f2` and `f4` are still valid abstractions for the changed functions `f2'` and `f4'`. Note that this is a local and thus computationally cheap check. Without the requirement Req1, the whole formula for the entire updated program would have to be constructed and checked again, which could require many more computational resources.

Theorem 2 offers the choice of interpolation systems generating interpolants of different strength satisfying Req1. In this second example, the benefit of a stronger interpolant is a tighter abstraction, i.e., the interpolant more closely reflects the actual behavior of the corresponding function. On the other hand, a weaker interpolant is more permissive. So it is more likely to remain a valid abstraction, when the corresponding function gets updated.

[6] In accordance with [8], we expect the transition relation to be a relation over state variables and their primed versions for the next state values and we use superscript $^{\langle i\rangle}$ to indicate addition of i primes (or removal if i is negative).

4 Inductive Sequence of Interpolants

This section analyzes the generation of a sequence of interpolants from the conjuncts of an unsatisfiable formula; the requirement (Req2) is to guarantee that the sequence is inductive [7]. We formally describe the problem, proving that the requirement is satisfied if the interpolants are produced using Pudlák's interpolation system, and later generalize the result to a particular family of systems in the lattice. We conclude by illustrating the applications to model checking.

4.1 Problem Description

As input, we assume an unsatisfiable formula ϕ in CNF, such that $\phi \triangleq \phi_1 \wedge \ldots \wedge \phi_n$ and each ϕ_i is a conjunction of clauses. Given a refutation R of ϕ and a sequence of labeled interpolation systems $Itp_{L_0}, \ldots, Itp_{L_n}$, we compute a sequence of interpolants $I_0, I_1, \ldots, I_n$ from R; I_i is obtained by setting $\phi_1 \wedge \ldots \wedge \phi_i$ to A and $\phi_{i+1} \wedge \ldots \wedge \phi_n$ to B ($I_i \triangleq Itp_{L_i}(\phi_1 \wedge \ldots \wedge \phi_i, \phi_{i+1} \wedge \ldots \wedge \phi_n)$), in particular $I_0 \equiv Itp_{L_0}(\top, \phi) \equiv \top$ and $I_n \equiv Itp_{L_n}(\phi, \top) \equiv \bot$. These ways of splitting the formula ϕ into A and B will be referred to as *configurations*.

We prove that $I_0, I_1, \ldots, I_n$ is an inductive sequence of interpolants: for every i, $I_i \wedge \phi_{i+1} \rightarrow I_{i+1}$ holds (requirement Req2) if, for every i, $Itp_{L_i} \equiv Itp_P$ (as in the previous setting, Itp_P can be associated with different labelings for different values of i). Then we generalize to a family of sequences of interpolation systems.

Notice that, for a given i, only two configurations need to be taken into account, the first associated with I_i ($A \triangleq \phi_1 \wedge \ldots \wedge \phi_i$, $B \triangleq \phi_{i+1} \wedge \ldots \wedge \phi_n$), the second with I_{i+1} ($A \triangleq \phi_1 \wedge \ldots \wedge \phi_{i+1}$, $B \triangleq \phi_{i+2} \wedge \ldots \wedge \phi_n$); ϕ_{i+1} is the only subformula shared between A and B.

Since the proof is independent of i, to simplify the notation we will represent $\phi_1 \wedge \ldots \wedge \phi_i$ as X, ϕ_{i+1} as S, $\phi_{i+2} \wedge \ldots \wedge \phi_n$ as Y (so that the formula is $X \wedge S \wedge Y$), I_i as I, I_{i+1} as J and $I_i \wedge \phi_{i+1} \rightarrow I_{i+1}$ as $I \wedge S \rightarrow J$.

4.2 Proof for Pudlák's System

Theorem 3. *For Pudlák's interpolation system, $I \wedge S \rightarrow J$.*

Proof. By the above definitions, $I \equiv Itp(X, S \wedge Y)$ and $J \equiv Itp(X \wedge S, Y) = \neg Itp(Y, X \wedge S)$ (by symmetry of Pudlák's system). Denoting $K \triangleq Itp(S, X \wedge Y)$, Lemma 1 states that $I \wedge K \wedge \neg J \rightarrow \bot$. Since $S \rightarrow K$, then $I \wedge S \wedge \neg J \rightarrow \bot$, that is $I \wedge S \rightarrow J$. □

4.3 Proof Generalization

We will now prove that $I \wedge S \rightarrow J$ holds in all the sequences of interpolation systems that comply with particular coloring restrictions. As shown in Table 4, two configurations are to be considered, which share the conjunct S. By $C\lfloor_{1,\sigma}$ and $C\lfloor_{2,\sigma}$ we denote the restriction of a clause C to the literals of color σ according to the labeling of configurations 1 and 2, respectively.

Table 4: Variables coloring for Definition 1

p in ?	Variable *class*, *color* for each configuration	
	$A \triangleq X$, $B \triangleq S \wedge Y$	$A \triangleq X \wedge S$, $B \triangleq Y$
X	A, a	A, a
S	A, a	B, b
Y	B, b	B, b
X, S	$AB, \alpha \in \{a, b, ab\}$	A, a
S, Y	B, b	$AB, \beta \in \{a, b, ab\}$
X, Y	$AB, \gamma_1 \in \{a, b, ab\}$	$AB, \gamma_2 \in \{a, b, ab\}$
X, S, Y	$AB, \delta_1 \in \{a, b, ab\}$	$AB, \delta_2 \in \{a, b, ab\}$

To simplify the proofs we initially enforce a set of constraints, so that the color taken by the occurrence of a variable in a clause in the two configurations is consistent; we will later show that the result still holds if the constraints are relaxed.

Definition 1 (Coloring constraints). *We define a set of coloring constraints (*CC*) over Table 4 as follows:* $\alpha = a$, $\beta = b$, $\gamma_1 = \gamma_2$, $\delta_1 = \delta_2$.

Lemma 2. $I \wedge S \rightarrow J$, *assuming the coloring constraints of Definition 1.*

Proof (by structural induction). We prove that, for any clause C in a refutation of $X \wedge S \wedge Y$, $f_C \wedge I(C) \wedge S \wedge \neg J(C) \rightarrow \bot$, where f_C is an additional constraint (to be determined), dependent on C, that becomes empty at the end of the proof ($f_\bot \equiv \top$). For simplicity we drop the parameter C in I, J. I^+, I^- are defined as in the previous setting, similarly J^+ and J^-.

Base case (leaf). Case splitting on C (refer to Table 4):

$C \in X$: $I \equiv C\lfloor_{1,b}$ and $J \equiv C\lfloor_{2,b}$
$C \in S$: $I \equiv \neg(C\lfloor_{1,a})$ and $J \equiv C\lfloor_{2,b}$
$C \in Y$: $I \equiv \neg(C\lfloor_{1,a})$ and $J \equiv \neg(C\lfloor_{2,a})$

We construct f_C in order to simultaneously satisfy the following conditions:

$C \in X$: $f_C \wedge C\lfloor_{1,b} \wedge S \wedge \neg(C\lfloor_{2,b}) \rightarrow \bot$
$C \in S$: $f_C \wedge \neg(C\lfloor_{1,a}) \wedge S \wedge \neg(C\lfloor_{2,b}) \rightarrow \bot$
$C \in Y$: $f_C \wedge \neg(C\lfloor_{1,a}) \wedge S \wedge C\lfloor_{2,a} \rightarrow \bot$

The CC constraints yield $C\lfloor_{1,b} \wedge \neg(C\lfloor_{2,b}) \rightarrow \bot$ and $\neg(C\lfloor_{1,a}) \wedge C\lfloor_{2,a} \rightarrow \bot$; as for $\neg(C\lfloor_{1,a}) \wedge \neg(C\lfloor_{2,b})$, it "counteracts" the literals of S with variables in X, S and S, Y and X, S, Y(colored a or b). The literals left are those whose variables are in S (denoted by $C\lfloor^S$) and those in X, S, Y colored ab (denoted by $C\lfloor_{ab}^{XSY}$); it is thus sufficient to set $f_C \triangleq \neg(C\lfloor^S \vee C\lfloor_{ab}^{XSY})$.

Inductive step (inner node). The inductive hypothesis (i.h.) consists of $f_{(C^+ \vee p)} \wedge I^+ \wedge S \wedge \neg J^+ \rightarrow \bot$, $f_{(C^- \vee \overline{p})} \wedge I^- \wedge S \wedge \neg J^- \rightarrow \bot$.

Now $f_C \equiv f_{(C^+ \vee C^-)}$, that is $\neg((C^+ \vee C^-)\lfloor^S) \wedge \neg((C^+ \vee C^-)\lfloor_{ab}^{XSY})$.
We have:

$$f_C \rightarrow f_{C^+} \wedge f_{C^-} \qquad (1)$$

since $\neg((C^+ \vee C^-)\lfloor^S) \leftrightarrow \neg(C^+\lfloor^S) \wedge \neg(C^-\lfloor^S)$ and $(C^+\lfloor_{ab}^{XSY}) \vee (C^-\lfloor_{ab}^{XSY}) \rightarrow ((C^+ \vee C^-)\lfloor_{ab}^{XSY})$ [7].
Case splitting based on the presence of the pivot p in $X/S/Y$ (see Table 4):

Case 1 (p in X).

$$\begin{aligned}
f_C \wedge S \wedge I \wedge \neg J \leftrightarrow \\
f_C \wedge S \wedge (I^+ \vee I^-) \wedge \neg(J^+ \vee J^-) \leftrightarrow \\
f_C \wedge S \wedge (I^+ \vee I^-) \wedge \neg J^+ \wedge \neg J^- \leftrightarrow \\
(f_C \wedge S \wedge I^+ \wedge \neg J^+ \wedge \neg J^-) \vee (f_C \wedge S \wedge I^- \wedge \neg J^- \wedge \neg J^+) \rightarrow \\
(f_C \wedge S \wedge I^+ \wedge \neg J^+) \vee (f_C \wedge S \wedge I^- \wedge \neg J^-) \rightarrow^{(1)} \\
(f_{C^+} \wedge S \wedge I^+ \wedge \neg J^+) \vee (f_{C^-} \wedge S \wedge I^- \wedge \neg J^-) \rightarrow^{(2)} \\
(f_{(C^+ \vee p)} \wedge S \wedge I^+ \wedge \neg J^+) \vee (f_{(C^- \vee \overline{p})} \wedge S \wedge I^- \wedge \neg J^-) \rightarrow^{\text{i.h.}} \bot
\end{aligned}$$

where (2) holds since p is restricted.

Case 2 (p in S).

$$\begin{aligned}
f_C \wedge S \wedge I \wedge \neg J \leftrightarrow \\
f_C \wedge S \wedge (I^+ \wedge I^-) \wedge \neg(J^+ \vee J^-) \leftrightarrow \\
f_C \wedge S \wedge I^+ \wedge I^- \wedge \neg J^+ \wedge \neg J^- \leftrightarrow \\
(f_C \wedge S \wedge I^+ \wedge \neg J^+) \wedge (f_C \wedge S \wedge I^- \wedge \neg J^-) \rightarrow^{(1)} \\
(f_{C^+} \wedge S \wedge I^+ \wedge \neg J^+) \wedge (f_{C^-} \wedge S \wedge I^- \wedge \neg J^-) \rightarrow^{(2)} \\
(f_{C^+} \wedge S \wedge I^+ \wedge \neg J^+ \wedge \overline{p}) \vee (f_{C^-} \wedge S \wedge I^- \wedge \neg J^- \wedge p) \leftrightarrow^{(3)} \\
(f_{(C^+ \vee p)} \wedge S \wedge I^+ \wedge \neg J^+) \vee (f_{(C^- \vee \overline{p})} \wedge S \wedge I^- \wedge \neg J^-) \rightarrow^{\text{i.h.}} \bot
\end{aligned}$$

where (2) holds since $\phi \wedge \chi \rightarrow (\phi \wedge p) \vee (\chi \wedge \overline{p})$, (3) since $f_{C_i} \wedge \overline{p} \leftrightarrow f_{(C_i \vee p)}$.

Case 3 (p in Y). Dual to Case 1.

Case 4 (p in X, S). As for Case 1.

Case 5 (p in S, Y). As for Case 3.

[7] Notice that the inverse of the last implication does not hold; in fact, a variable in X, S, Y could have, e.g., color ab in $C^+ \vee C^-$ because it has color a in C^+ and color b in C^- (recall the definition of $\sqcup$ in §2.3), which means that it appears in $(C^+ \vee C^-)\lfloor_{ab}^{XSY}$ but gets restricted both in $C^+\lfloor_{ab}^{XSY}$ and in $C^-\lfloor_{ab}^{XSY}$.

Case 6 (p in X, Y). As for Case 1 or Case 3 if color is either a or b. If ab:

$$\begin{aligned}
f_C \wedge S \wedge I \wedge \neg J \leftrightarrow \\
f_C \wedge S \wedge ((I^+ \vee p) \wedge (I^- \vee \overline{p})) \wedge \neg((J^+ \vee p) \wedge (J^- \vee \overline{p})) \leftrightarrow \\
f_C \wedge S \wedge (I^+ \vee p) \wedge (I^- \vee \overline{p}) \wedge (\neg(J^+ \vee p) \vee \neg(J^- \vee \overline{p})) \rightarrow \\
(f_C \wedge S \wedge (I^+ \vee p) \wedge \neg(J^+ \vee p)) \vee (f_C \wedge S \wedge (I^- \vee \overline{p}) \wedge \neg(J^- \vee \overline{p})) \leftrightarrow \\
(f_C \wedge S \wedge (I^+ \vee p) \wedge \neg J^+ \wedge \overline{p}) \vee (f_C \wedge S \wedge (I^- \vee \overline{p}) \wedge \neg J^- \wedge p) \rightarrow \\
(f_C \wedge S \wedge I^+ \wedge \neg J^+) \vee (f_C \wedge S \wedge I^- \wedge \neg J^-) \rightarrow^{(1)} \\
(f_{C^+} \wedge S \wedge I^+ \wedge \neg J^+) \vee (f_{C^-} \wedge S \wedge I^- \wedge \neg J^-) \rightarrow^{(2)} \\
(f_{(C^+ \vee p)} \wedge S \wedge I^+ \wedge \neg J^+) \vee (f_{(C^- \vee \overline{p})} \wedge S \wedge I^- \wedge \neg J^-) \rightarrow^{\text{i.h.}} \bot
\end{aligned}$$

where (2) holds since p is restricted.

Case 7 (p in X, S, Y). As for Case 1 or Case 3 if color is either a or b, since p is restricted. As for Case 6 if color is ab, but last three lines are replaced by:

$$\begin{aligned}
(f_C \wedge S \wedge (I^+ \vee p) \wedge \neg J^+ \wedge \overline{p} \wedge \overline{p}) \vee (f_C \wedge S \wedge (I^- \vee \overline{p}) \wedge \neg J^- \wedge p \wedge p) \rightarrow \\
(f_C \wedge S \wedge I^+ \wedge \neg J^+ \wedge \overline{p}) \vee (f_C \wedge S \wedge I^- \wedge \neg J^- \wedge p) \rightarrow^{(1)} \\
(f_{C^+} \wedge S \wedge I^+ \wedge \neg J^+ \wedge \overline{p}) \vee (f_{C^-} \wedge S \wedge I^- \wedge \neg J^- \wedge p) \rightarrow^{(2)} \\
(f_{(C^+ \vee p)} \wedge S \wedge I^+ \wedge \neg J^+) \vee (f_{(C^- \vee \overline{p})} \wedge S \wedge I^- \wedge \neg J^-) \rightarrow^{\text{i.h.}} \bot
\end{aligned}$$

where (2) holds since $f_{C_i} \wedge \overline{p} \leftrightarrow f_{(C_i \vee p)}$. □

Lemma 3. *The CC constraints of Definition 1 can be relaxed as follows:* $\alpha \preceq a$, $b \preceq \beta$, $\gamma_1 \preceq \gamma_2$, $\delta_1 \preceq \delta_2$.

Proof. Let I, J be interpolants generated using interpolation systems according to the constraints CC in Def. 1. Let us use primed variables to represent the relaxed constraints of Lemma 3 as $\alpha' \preceq \alpha = a$ (i.e., any α'), $b = \beta \preceq \beta'$ (i.e., any β'), $\gamma_1' \preceq \gamma_1 = \gamma_2 \preceq \gamma_2'$, $\delta_1' \preceq \delta_1 = \delta_2 \preceq \delta_2'$ and let I', J' represent the corresponding interpolants. Recalling the order $b \preceq ab \preceq a$ over colors, which induces a partial order over interpolation systems (see §2.3), we get $I' \rightarrow I$ and $J \rightarrow J'$. Thus, the relaxed constraints yield: $I' \wedge S \wedge \neg J' \rightarrow I \wedge S \wedge \neg J \rightarrow \bot$. □

The above considerations lead to the following result:

Theorem 4. *For any sequence of interpolation systems* $Itp_{L_0}, \ldots, Itp_{L_n}$*, which respects the relaxed constraints of Lemma 3,* $I_0, I_1, \ldots, I_n$ *is an inductive sequence of interpolants, i.e.,* $I_i \wedge \phi_{i+1} \rightarrow I_{i+1}$*, for* $0 \leq i < n$.

4.4 Application to Model Checking

The above setting occurs, for example, during counterexample-guided abstraction refinement. Given a spurious error trace, the goal is to annotate nodes of the abstract reachability tree with an inductive sequence of formulae that together

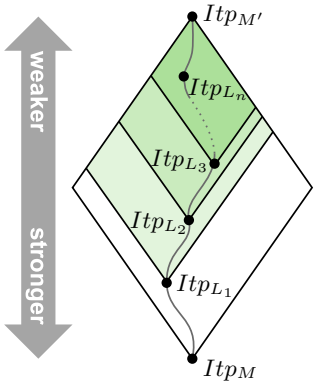

Figure 4: Weakening interpolation systems

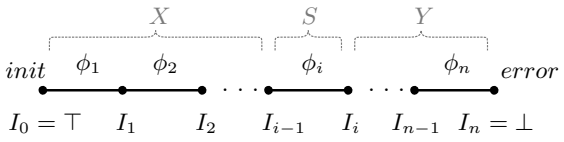

Figure 5: Spurious error trace annotated by interpolants

rule out the trace. The trace is represented as a formula $\phi_1 \wedge \ldots \wedge \phi_n$, which is constructed from the SSA form of the trace and which is unsatisfiable if and only if the error trace is infeasible. If so, a sequence of interpolants I_i is created by means of a sequence of interpolation systems as $I_i \triangleq Itp_{L_i}(\phi_1 \wedge \ldots \wedge \phi_i, \phi_{i+1} \wedge \ldots \wedge \phi_n)$, $I_0 \equiv \top$, and $I_n \equiv \bot$, as depicted in Fig. 5 (along with the partitioning of the path formula into X, S, and Y for the purposes of Lemma 2). In addition, Req2 requires the sequence of interpolants to be inductive, i.e., $I_i \wedge \phi_{i+1} \rightarrow I_{i+1}$. In which case, the error trace is removed from the refined abstraction. However, such a sequence of interpolants is not inductive in general and thus the same error trace may remain also in the refined abstraction, should Req2 be violated. As already mentioned, refinement phases of tools like Blast [1] and Impact [12] rely on this fact.

Theorem 4 ensures that Req2 is satisfied by interpolants derived using interpolation systems weakening towards the end of the error trace (depicted in Fig. 4). It is thus possible to choose an interpolation system depending on the instruction at the current position along the error trace (i.e., the current S in the language of Lemma 2). As an example, some instructions in the error trace may trigger generation of weaker interpolants (i.e., a more coarse abstraction). In practice, this would affect the speed of convergence of the refinement loop.

5 Related Work

In model checking, interpolation is a common means for abstraction. Interpolation is used as an abstract post-image operator in hardware bounded model checking [9]; the interpolant is generated from the proof of unsatisfiability of a bounded model checking formula so that it represents a superset of states reachable from the initial states. Faster convergence of the model checking algorithm applying this method of abstraction is observed during experiments. Interpola-

tion is also used in concolic execution to propagate reasons of trace infeasibility backward towards the start of the program [13]. This allows discarding infeasible traces as early as possible and thus saving the effort of evaluating them. In software bounded model checking, function summaries can be created using interpolation [16]; these are employed during analysis of different properties to represent a function body without the need to process its whole call tree. Interpolation also proves to be very useful in refining predicate abstraction based on spurious counterexamples [7]. Here, interpolation is used to derive new predicates that rule out the spurious error traces. The listed works describe applications of interpolation in model checking; see [11] for a comprehensive list. Typically, the authors limit themselves to either Pudlák's or McMillan's algorithms without considering further variation in the strength of interpolants. We believe that all these techniques would benefit from choosing among interpolants of appropriate strengths. The results of this paper provide safe boundaries for such a choice.

Other related work concerns the actual generation of interpolants. Pudlák [14] shows that interpolants can be derived in linear time with respect to the given refutation. McMillan [10] proposes a different algorithm that produces logically stronger interpolants and addresses both propositional logic and a quantifier free combination of the theories of uninterpreted functions and linear arithmetic. In [2], local proof transformations are presented that (by reordering proofs and removing so called ab-mixed predicates) can change a refutation produced by a standard SMT-solver so that it becomes suitable for interpolant generation. Authors of [5] provide a generalized algorithm for interpolation that subsumes both Pudlák's and McMillan's algorithms. They also show that a complete lattice of interpolants ordered by the implication relation can be systematically derived from a given refutation. However, they do not study the limits with regard to the actual application of interpolants of differing strength in model checking. Building upon [5], our work provides this missing connection and defines and proves these boundaries in the particular model checking settings.

In this paper, we consider two classes of model checking approaches that put additional requirements on the resulting interpolants. These requirements were previously formulated in the literature (Req1 in [8] and Req2 in [7]). Until now, however, the conditions under which they hold have not been thoroughly studied, in particular in the context of different interpolation systems of [5]. Novelty of our work lies in the fact that we provide constraints on the complete lattice of interpolants that, when obeyed, ensure satisfaction of the requirements.

6 Conclusion

Interpolants are not unique and may vary in strength. The effects of using interpolants of different strength in model checking can be substantial and are yet to be properly studied. However, common applications of interpolation in model checking put additional requirements that (as we show) are not satisfied in general, specifically when interpolants of various strengths are generated by different interpolation systems. In this paper, for two classes of model checking

techniques employing interpolation, we showed the safe boundaries for varying the strength of interpolants, proving the limitations under which the requirements are satisfied. Our theoretical result enables study of the effects of the interpolants strength on the model checking algorithms. Since our result is not limited to an ad-hoc proof system, any state-of-the-art solver can be used to generate proofs used for interpolation. Strength and size of interpolants can be also affected by proof manipulation procedures as shown in [15]. We intend to address the above questions in our future work.

References

1. Beyer, D., Henzinger, T.A., Jhala, R., Majumdar, R.: The Software Model Checker Blast: Applications to Software Engineering. Int. J. STTT 9, 505–525 (2007)
2. Bruttomesso, R., Rollini, S.F., Sharygina, N., Tsitovich, A.: Flexible Interpolation with Local Proof Transformations. In: ICCAD '10. pp. 770–777. IEEE (2010)
3. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-Guided Abstraction Refinement. In: CAV '00. LNCS, vol. 1855, pp. 154–169. Springer (2000)
4. Craig, W.: Three Uses of the Herbrand-Gentzen Theorem in Relating Model Theory and Proof Theory. J. of Symbolic Logic pp. 269–285 (1957)
5. D'Silva, V., Kroening, D., Purandare, M., Weissenbacher, G.: Interpolant Strength. In: VMCAI '10. LNCS, vol. 5944, pp. 129–145. Springer (2010)
6. Heizmann, M., Hoenicke, J., Podelski, A.: Nested Interpolants. In: POPL '10. pp. 471–482. ACM (2010)
7. Henzinger, T.A., Jhala, R., Majumdar, R., McMillan, K.L.: Abstractions from Proofs. In: POPL '04. pp. 232–244. ACM (2004)
8. Jhala, R., McMillan, K.L.: Interpolant-Based Transition Relation Approximation. Logical Methods in Computer Science 3(4) (2007)
9. McMillan, K.L.: Interpolation and SAT-Based Model Checking. In: CAV '03. LNCS, vol. 2725, pp. 1–13. Springer (2003)
10. McMillan, K.L.: An Interpolating Theorem Prover. In: TACAS '04. LNCS, vol. 2988, pp. 16–30. Springer (2004)
11. McMillan, K.L.: Applications of Craig Interpolation in Model Checking. In: TACAS '05. LNCS, vol. 3440, pp. 1–12 (2005)
12. McMillan, K.L.: Lazy Abstraction with Interpolants. In: CAV '06. LNCS, vol. 4144, pp. 123–136. Springer (2006)
13. McMillan, K.L.: Lazy Annotation for Program Testing and Verification. In: CAV' 10. LNCS, vol. 6174, pp. 104–118. Springer (2010)
14. Pudlák, P.: Lower Bounds for Resolution and Cutting Plane Proofs and Monotone Computations. Journal of Symbolic Logic 62(3), 981–998 (1997)
15. Rollini, S.F., Bruttomesso, R., Sharygina, N.: An Efficient and Flexible Approach to Resolution Proof Reduction. In: HVC '10. LNCS, vol. 6504, pp. 182–196. Springer-Verlag (2010)
16. Sery, O., Fedyukovich, G., Sharygina, N.: Interpolation-based Function Summaries in Bounded Model Checking. In: HVC '11. LNCS (2011), to appear