

Proof Tree Preserving Sequence Interpolation of Quantified Formulas in the Theory of Equality

(work in progress)

Elisabeth Henkel, Jochen Hoenicke and Tanja Schindler

Department of Computer Science, University of Freiburg, Germany

Abstract

Interpolation of SMT formulas is difficult, especially in the presence of quantifiers since quantifier instantiations introduce mixed terms, i.e., terms containing symbols that belong to different partitions of the input formula. Existing interpolation algorithms for quantified formulas require proof modifications or syntactical restrictions on the generated proof trees. We present a non-restrictive, proof tree preserving approach to compute inductive sequences of interpolants for quantified formulas in the theory of equality with uninterpreted functions using a single proof tree.

Keywords

SMT, interpolation, quantified formulas, sequence interpolants, proof tree preserving interpolation

1. Introduction

We present an interpolation procedure for quantified formulas in the theory of equality with uninterpreted functions. Interpolants have various applications in model checking, for example, during abstraction refinement or invariant generation [1, 2, 3, 4]. For the analysis of sequential programs, sequence interpolants [2] that are computed from infeasible error paths, are of particular interest. For each point of a program path, a sequence interpolant provides a reason why an error is unreachable from this point on. Often, the interpolant is general enough to exclude several paths at once.

Many verification problems require the ability to reason about quantifiers, which are, among others, needed in user-added assertions, for the modeling of unsupported theories or memory modeling [5, 6]. However, the support of interpolation in the presence of quantified formulas is limited. A major challenge are mixed terms that are introduced in the context of quantifier instantiations. Most interpolation algorithms are either restricted to quantifier-free input formulas [7, 8, 9, 10, 11, 12, 13], require proof modifications [14], or require syntactical restrictions on the generated proof trees [15, 16, 17].


Our method is based on the proof tree preserving interpolation approach [12, 13] which allows for interpolation of quantifier-free formulas in the presence of mixed literals without the need for proof modifications or solver restrictions. In particular, sequence or tree interpolants

SMT 2021, 19th International Workshop on Satisfiability Modulo Theories, July 18 - 19, 2021

✉ henkele@informatik.uni-freiburg.de (E. Henkel); hoenicke@informatik.uni-freiburg.de (J. Hoenicke); schindle@informatik.uni-freiburg.de (T. Schindler)



© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

can be computed from one single proof of unsatisfiability. We extend this procedure to handle instantiation-based resolution proofs of quantified formulas similar to [14]. However, we simplify the presented quantifier introduction step and perform all necessary modifications only virtually such that the proof tree preserving character is restored.

The contributions of this paper are a proof tree and satisfiability preserving scheme for the virtual modification of mixed terms, a scheme to interpolate instantiation clauses generated by a quantifier solver, and modifications of interpolation rules for theory lemmas and resolvents to generate valid inductive sequences of interpolants from a single proof tree.

Related Work. Interpolation goes back to Craig [18] who proved that for a valid implication $A \rightarrow B$ in first-order logic, there always exists an interpolant (Craig called it “intermediate”), i.e., a formula I that is implied by A and implies B , and that contains only symbols that A and B share. Since then, several methods for computing interpolants from proofs of unsatisfiability have been presented both for quantifier-free and quantified problems. We focus our presentation on some of the most recent and some of the closest related works, and refer to [15, 19] for an overview of existing work before 2015. Most work on interpolation in the presence of quantified formulas has been done in the context of automated theorem proving.

Gleiss et al. [17] present an interpolation method based on splitting the proof of unsatisfiability into local subproofs. Their method can be applied to arbitrary first-order theories, but requires local proofs, i.e., proofs where each inference can be assigned to one partition as all occurring symbols lie in one partition. Each subproof that only uses inferences assigned to A is then summarized in an *intermediant*. For local proofs, the conjunction of these intermediants is an interpolant. The method is implemented in the Vampire theorem prover [20].

Bonacina and Johansson [15] present a method for interpolation in first-order logic with equality in the context of the superposition calculus which first computes *provisional interpolants* and then quantifies over constants that are not shared between A and B . Their method does not require local proofs, but only yields interpolants if the only non-shared symbols in the provisional interpolants are constants. The authors also discuss how the method can be used in the DPLL($\Gamma + \mathcal{T}$) calculus [21], but to our knowledge, there does not exist an implementation.

The method presented by Kovács and Voronkov [16] does not require local proofs either. They also follow a two-step approach, first computing *relational interpolants* and then quantifying over non-shared symbols. In contrast to [15], their method can deal with non-shared function symbols in the relational interpolants, but is restricted to logic without equality.

A basis for our work is the interpolation method by Christ and Hoenicke [14] for instantiation-based proofs in SMT solvers. It relies on proof transformations to obtain a *purified* proof where all instances of quantified formulas are local. This is achieved by introducing variables and auxiliary equations relating the variables to the terms they replace. To obtain an interpolant, these variables are bound by quantifiers once the terms they replace are resolved from the proof.

Finally, Christ et al. [12] present a framework that allows for computing quantifier-free interpolants for quantifier-free problems from a given proof of unsatisfiability without modifying the proof. In particular, it deals with *mixed* literals containing non-shared terms from both A and B by auxiliary variables and more complex interpolation rules. We describe this method in more detail in Section 3.

2. Notation

We assume the usual notions of first-order logic. A *theory* \mathcal{T} is defined by a signature that contains interpreted and uninterpreted constant, function, and predicate symbols, and by a set of axioms that settle the meaning for its interpreted symbols. We consider the theory of equality (with uninterpreted functions), whose axioms state reflexivity, symmetry, and transitivity for the single interpreted symbol, the equality predicate $=$, and congruence for each uninterpreted function symbol.

As usual, a *term* is a variable, a constant, or an application of an n -ary function symbol to n terms. An *atom* is the formula that is always true (\top), an equality between two terms, or the application of an n -ary predicate to n terms. A *literal* is an atom or its negation. A *clause* is a disjunction of literals, and a formula is in *conjunctive normal form* (CNF) if it is a conjunction of clauses. We assume that formulas are in CNF where each clause can be preceded by universal quantifiers only. This can be accomplished by Skolemization.

We usually denote functions by f, g, h , terms by a, b, s, t , variables by v, x, y, z , literals by ℓ , clauses by C , the empty clause by \perp , and formulas by ϕ, ψ, A, B, I . For two formulas ϕ and ψ , we use $\phi \models \psi$ to denote that ψ is a logical consequence of ϕ in the theory of equality. Formula ϕ is unsatisfiable if and only if $\phi \models \perp$. We denote the equivalence of two formulas by $\phi \equiv \psi$. A formula is said to be *ground* if it does not contain any variable.

A *substitution* maps variables to terms. We denote the result of substituting the terms \bar{t} for the variables \bar{x} in a formula $\phi(\bar{x})$ by $\phi(\bar{t})$. We call a substitution *ground* if the terms \bar{t} are ground. By $\phi[\ell]$ we denote a formula in negation normal form where the literal ℓ only occurs positively, and by $\phi[\psi]$ the same formula where all occurrences of ℓ are replaced by the formula ψ .

3. Preliminaries

The basis for our interpolation procedure is an instantiation-based resolution proof for the unsatisfiability of a given formula. We shortly describe the structure of such proofs and how interpolants can be computed from ground resolution proofs.

Instantiation-Based Resolution Proofs. An *instance* of a quantified formula $\forall \bar{x}.\phi(\bar{x})$ is the result of applying a ground substitution for the variables \bar{x} to the body ϕ , i.e., the formula $\phi(\bar{t})$ for ground terms \bar{t} . Most SMT solvers treat problems containing quantified formulas by successively adding instances of the quantified formulas to the ground part which is then solved in the usual way (e.g., with the DPLL(\mathcal{T})/CDCL algorithm). This approach is mainly suitable to show unsatisfiability of a formula, but it can also show satisfiability in some restricted fragments.

A *resolution proof* for the unsatisfiability of a formula in CNF is a derivation of the empty clause \perp using the rule

$$\frac{C_1 \vee \ell \quad C_2 \vee \neg \ell}{C_1 \vee C_2}$$

where C_1 and C_2 are clauses, and ℓ is a literal called the *pivot* (literal). A resolution proof can be represented by a tree. For a formula in a theory \mathcal{T} that contains quantified subformulas, the tree has three types of leaves: *input clauses*, *theory lemmas*, i.e., clauses that are valid in

the theory \mathcal{T} , and *instantiation clauses* of the form $\neg(\forall \bar{x}.\phi(\bar{x})) \vee \phi(\bar{t})$. Note that we treat the quantified formula as a literal which also occurs positively as an input clause. The inner nodes are clauses obtained by resolution, and the unique root node is the empty clause \perp .

Interpolation. A *Craig interpolant* [18] for an unsatisfiable conjunction $A \wedge B$ is a formula I that is implied by A , contradicts B , and contains only symbols that occur in both A and B .¹ The notion of a Craig interpolant extends to formulas in a theory \mathcal{T} , in this case, symbols interpreted by the theory may occur in the interpolant. The notion of binary interpolants can be generalized to sequences of formulas. Given such a sequence of formulas ϕ_1, \dots, ϕ_n that together are unsatisfiable, a *sequence interpolant* [2] I_0, \dots, I_n has the following properties:

1. the formulas I_0 and I_n are \top and \perp , respectively,
2. the sequence of interpolants is inductive, i.e., for $i = 0, \dots, n$, the interpolant I_i together with the formula ϕ_{i+1} implies the next interpolant I_{i+1} , and
3. for $i = 1, \dots, n$, each interpolant I_i contains only symbols that occur both in some of the formulas ϕ_1, \dots, ϕ_i and in some of the formulas $\phi_{i+1}, \dots, \phi_n$.

We call the formulas ϕ_i the *partitions*² of the interpolation problem. If a symbol occurs in two partitions ϕ_j and ϕ_k , we can assume that it also occurs in all partitions between j and k without changing the symbol condition of the sequence interpolants. Therefore, we say a symbol *virtually occurs* in ϕ_i if it occurs in some ϕ_j with $j \leq i$ and in some ϕ_k with $i \leq k$.

Each interpolant I_i in a sequence interpolant is also a binary interpolant of $A_i := \phi_1 \wedge \dots \wedge \phi_i$ and $B_i := \phi_{i+1} \wedge \dots \wedge \phi_n$. Note that not every sequence of binary interpolants obtained by such partitioning is a sequence interpolant. A literal is *mixed* if there is no partition ϕ_i where all symbols of the literal virtually occur. All literals of the input problem are obviously not mixed, but mixed literals can be introduced by Nelson–Oppen equality propagation.

Interpolants can be computed from resolution proofs, e.g., with the interpolation systems by Pudlák [7] or by McMillan [8]. The general idea is to compute so-called partial interpolants for every formula proved in the intermediate steps. For resolution proofs such a formula is a clause C . The conjunction $\neg C$ is split and added to the input formulas to obtain a sequence interpolation problem. First, each term and literal is given one or more *colors* that correspond to the partitions where it occurs. A term that occurs in multiple partitions can be given multiple colors, or a color can be chosen arbitrarily. If a term has multiple colors, we require that the set of colors forms an interval $[i, j]$. We define the *projection* $\neg C \upharpoonright \phi_i$ as the conjunction of all literals in $\neg C$ that have color i . In particular, $\ell \upharpoonright \phi_i$ is ℓ if ℓ has color i , and \top otherwise. Then the *partial interpolant* of C is a sequence interpolant of $\phi_1 \wedge \neg C \upharpoonright \phi_1, \dots, \phi_n \wedge \neg C \upharpoonright \phi_n$. The interpolation systems of Pudlák and McMillan only differ in the way they define the coloring: for Pudlák a shared literal is colored with all partitions it occurs in, for McMillan a shared literal is colored with only the last partition it occurs in. Both systems require that each literal has at

¹As mentioned in the related work, Craig originally defined an interpolant for a valid implication $A \rightarrow B$, i.e., as a formula implied by A and contradictory to B . The notion we use is also known as *reverse interpolant* [22].

²The word partition can be used in English for the sections, or the boundaries, or the whole partitioning. In this paper we use the word partition for one section of the partitioning.

least one color and that all symbols in the literal occur in the corresponding partition. Mixed literals require a different interpolation scheme where projections $\ell \downarrow \phi_i$ are defined differently.

Proof Tree Preserving Interpolation. The proof tree preserving interpolation framework allows for computing binary [12] and sequence or tree interpolants [13] in the presence of mixed equality literals without modifying a given proof tree. It works with equality interpolating theories [9, 23].

A *mixed equality literal* $a = b$ or $a \neq b$ is a literal where a is colored with $[j', j]$ and b is colored with $[k, k']$ where these intervals are disjoint. W.l.o.g., we assume that $\phi_j < \phi_k$. We also say that this literal is j, k -mixed. The basic idea is to use a fresh auxiliary variable x for each mixed equality literal. When projecting a j, k -mixed literal, we split it into two literals using the auxiliary variable.

$$\begin{aligned} (a = b) \downarrow \phi_j &::= (a = x) & (a = b) \downarrow \phi_k &::= (x = b) & (a = b) \downarrow \phi_i &::= \top \text{ for } i \neq j, k \\ (a \neq b) \downarrow \phi_j &::= EQ(x, a) & (a \neq b) \downarrow \phi_k &::= \neg EQ(x, b) & (a \neq b) \downarrow \phi_i &::= \top \text{ for } i \neq j, k \end{aligned}$$

Here, EQ is an uninterpreted predicate that forces the interpolant to have a certain shape. The interpolation procedure treats EQ and x as shared symbols, which may occur in the interpolant. Note that $\ell \downarrow \phi_j \wedge \ell \downarrow \phi_k$ is equisatisfiable to ℓ and has the same models for the symbols in ℓ .

Again, we define the partial interpolants of a clause C as the sequence interpolant for the sequence $\phi_1 \wedge \neg C \downarrow \phi_1, \dots, \phi_n \wedge \neg C \downarrow \phi_n$. If the negation $\neg C$ of a clause contains a mixed equality $a = b$, the partial interpolants I_i for C with $j \leq i < k$ may contain the auxiliary variable x . If $\neg C$ contains $a \neq b$, we require that the variable x in the interpolant I_i only occurs in atoms of the form $EQ(x, s_l)$ which must occur positively in I_i . This is not really a restriction as interpolants naturally have this shape. The s_l are shared terms, which are closely related to equality-interpolating terms in the sense of [9].

As mentioned earlier, an interpolant is computed by annotating each clause C in the proof tree, starting with the leaves, by a partial interpolant I_0, \dots, I_n . We denote this by $C : I_1, \dots, I_{n-1}$ (we omit $I_0 \equiv \top$ and $I_n \equiv \perp$). Resolution combines the partial interpolants of the involved clauses. To resolve on a mixed literal $a = b$, we combine the partial interpolants of the input clauses using the following function that replaces all occurrences of $EQ(x, s_l)$ in I by a copy of I' where all occurrences of x are replaced by s_l .

$$ruleEq(a = b, I[EQ(x, s_1)] \dots [EQ(x, s_m)], I'(x)) ::= I[I'(s_1)] \dots [I'(s_m)]$$

The following rule is used to compute the partial interpolants for the resolvent of a resolution rule from the partial interpolants of the antecedents.

$$\frac{C_1 \vee \ell : I_1, \dots, I_{n-1} \quad C_2 \vee \neg \ell : I'_1, \dots, I'_{n-1}}{C_1 \vee C_2 : I''_1, \dots, I''_{n-1}}$$

with $I''_i ::= \begin{cases} I_i \vee I'_i & \text{if } \ell \text{ has only colors } j \leq i, \\ I_i \wedge I'_i & \text{if } \ell \text{ has only colors } j > i, \\ ruleEq(\ell, I_i, I'_i) & \text{if } \ell \text{ is } j, k\text{-mixed, } j \leq i < k, \\ (I_i \vee \ell) \wedge (I'_i \vee \neg \ell) & \text{if } \ell \text{ has colors } [j, k] \text{ with } j \leq i < k. \end{cases} \quad (\text{resolution})$

Example 1 (Mixed Literals). Consider the following two (tautological) clauses

$$C := a = b \vee a \neq t \vee t \neq b \quad C' := a \neq b \vee s \neq f(a) \vee s = f(b),$$

where a is ϕ_1 -local, b is ϕ_2 -local, and the remaining symbols are shared. A partial interpolant of C is the formula $EQ(x, t)$, since it is an interpolant of

$$\neg C \upharpoonright \phi_1 \equiv EQ(x, a) \wedge a = t \quad \text{and} \quad \neg C \upharpoonright \phi_2 \equiv \neg EQ(x, b) \wedge b = t.$$

This interpolant can be obtained by using well-known interpolation techniques for the theory of equality. Note that $EQ(x, t)$ captures the equality-interpolating term t of $a = b$ in the sense of [9]. For the second clause C' , the formula $s = f(x)$ is a partial interpolant, i.e., it is an interpolant of

$$\neg C' \upharpoonright \phi_1 \equiv a = x \wedge s = f(a) \quad \text{and} \quad \neg C' \upharpoonright \phi_2 \equiv x = b \wedge s \neq f(b).$$

The interpolant for the resolvent of C and C' can now be obtained using the resolution rule:

$$\frac{a = b \vee a \neq t \vee t \neq b : EQ(x, t) \quad a \neq b \vee s \neq f(a) \vee s = f(b) : s = f(x)}{a \neq t \vee s \neq f(a) \vee t \neq b \vee s = f(b) : s = f(t)}$$

The interested reader is invited to check that the annotated partial interpolant is correct.

4. Mixed Terms

One obstacle to use proof tree preserving interpolation in the presence of quantifiers are mixed terms. To illustrate the problem, let us consider the following example.

Example 2 (Running Example). Take the sequence of formulas ϕ_1, ϕ_2, ϕ_3 with

$$\phi_1 : \forall x. g(h(x)) = x \quad \phi_2 : \forall y. f(g(y)) \neq b \quad \phi_3 : \forall z. f(z) = z.$$

The conjunction of the three formulas is unsatisfiable, which can be seen as follows. Instantiating ϕ_1 with b gives $g(h(b)) = b$. Instantiating ϕ_2 with $h(b)$ gives $f(g(h(b))) \neq b$. Substituting $g(h(b))$ using the first equality yields $f(b) \neq b$, which is clearly a contradiction to ϕ_3 .

A *mixed term* is a term that uses symbols from at least two different partitions. In the example above, a mixed term is $h(b)$, because h only occurs in ϕ_1 and b only occurs in ϕ_2 . Mixed terms may be generated during the instantiation process of quantified formulas. They can thus appear in a resolution proof within instantiation clauses and theory lemmas but not in input clauses. A resolution node may contain a mixed term if it was already contained in one of its antecedents. Interpolation algorithms based on resolution proofs usually rely upon non-mixed terms. We do not want to restrict the instantiation procedure to local terms as this would negatively effect the performance of the solver. Transforming the proofs into localized proofs in the context of sequence interpolation can break the inductivity property if not done in a consistent manner. So it would require to either localize the proof for all partitions at once or to compute the interpolants by repeated binary interpolation, which requires a new proof for each interpolant.

We propose a method to handle mixed terms, simply by coloring them with some input partition. This partition does not contain all symbols occurring in the term. Thus, we replace subterms of the wrong color by fresh variables to purify the term. Each variable is defined by adding an auxiliary equation to all partitions where the subterm's outer function symbol occurs.

4.1. Purification of Mixed Literals

We proceed by first assigning a color to each term, and purify mixed terms afterwards. As a symbol may appear in several partitions, we can assign one or multiple of these partitions as its color. However, this decision may affect the number of quantifiers within the final interpolant. We use the following coloring scheme to obtain as few quantifiers as possible. To color a term, we start by coloring its subterms from bottom up. The color of a constant symbol corresponds to the partitions it occurs in. For a function application $f(t_1, \dots, t_n)$, we compute the intersection of the partitions where f occurs and the colors of the terms t_1, \dots, t_n . If the intersection is non-empty, it is used to color the term. Otherwise, if it is empty, meaning that the term contains symbols for which there is no common partition. We color the term by all partitions in which the function symbol f occurs. The whole literal is then colored as if the top-level terms occurred in the partitions corresponding to the terms' colors.

If the literal ℓ colored with partition i contains a mixed term, it contains some constant c or function application $f(t_1, \dots, t_n)$ where c or f does not occur in the partition i . In the projection $\ell \downarrow \phi_i$, these constants and function applications are replaced by fresh variables $v_{f(t_1, \dots, t_n)}$. Each fresh variable is set equal to the term it stands for by an auxiliary equation of the form $v_{f(t_1, \dots, t_n)} = f(v_{t_1}, \dots, v_{t_n})$. The arguments of the function application are also replaced by fresh variables and the corresponding auxiliary equations are added to their corresponding partitions, i.e., the term is flattened.

Example 3 (Purification). Take the sequence of formulas ϕ_1, ϕ_2, ϕ_3 from the running example (Example 2). Consider the literal $\ell \equiv g(h(b)) = b$ that stems from instantiating ϕ_1 . The subterm b occurs only in partition 2. The term $h(b)$ is mixed since h only occurs in partition 1. We assign it partition 1. The term $g(h(b))$ is now also assigned partition 1 as it is the intersection where g occurs (1, 2) and what its argument $h(b)$ was assigned. The whole literal is treated as mixed equality, since the left-hand side $g(h(b))$ of ℓ is colored 1 and the right-hand side b is colored 2.

When computing the projection of the mixed equality $\ell \downarrow \phi_i$, we split ℓ first into $g(h(b)) = x$ and $x = b$, as we explained in Section 3. In the projection, we further replace the subterm b on the left-hand side (that does not match partition 1) with a fresh variable. The auxiliary equation is added to partition 2, since this is the partition where b occurs.

$$\ell \equiv g(h(b)) = b : \quad \ell \downarrow \phi_1 \equiv g(h(v_b)) = x, \quad \ell \downarrow \phi_2 \equiv x = b \wedge v_b = b, \quad \ell \downarrow \phi_3 \equiv \top$$

Consider the literal $\ell \equiv f(g(h(b))) = b$ that stems from instantiating ϕ_2 . As before, the term $g(h(b))$ is assigned partition 1. The term $f(g(h(b)))$ is mixed as f is only in partition 2, 3. So the whole term gets assigned partitions 2, 3. Since the right-hand side b of the equality appears only in partition 2, the whole literal is assigned partition 2.

When computing the projection, we replace all subterms not occurring in partition 2 with fresh variables, as well as their subterms. The auxiliary equations are added to the partitions where the corresponding function symbol appears.

$$\ell \equiv f(g(h(b))) = b : \quad \ell \downarrow \phi_1 \equiv v_h = h(v_b), \quad \ell \downarrow \phi_2 \equiv f(g(v_h)) = b \wedge v_b = b, \quad \ell \downarrow \phi_3 \equiv \top$$

Note that if h occurred in multiple partitions, the auxiliary equation $v_h = h(v_b)$ would be added to all projections where h occurs in the partition.

4.2. Quantifier Introduction

As mentioned in Section 3, our algorithm computes for each clause C in the resolution proof a partial interpolant of $\phi_1 \wedge \neg C \downarrow \phi_1, \dots, \phi_n \wedge \neg C \downarrow \phi_n$. Interpolation of leaf clauses is performed based on the color of literals, i.e., the interpolants are computed as if the literal occurred in the partition corresponding to its color. This may introduce symbols from the wrong partition in the interpolant, i.e., the resulting interpolant may violate the symbol condition. The algorithm then replaces all terms containing symbols that are not allowed in the interpolant by their auxiliary variable. In many cases, these auxiliary variables are allowed to occur in the partial interpolant, since they satisfy the symbol condition with respect to the corresponding interpolation problem $\phi_1 \wedge \neg C \downarrow \phi_1, \dots, \phi_n \wedge \neg C \downarrow \phi_n$.

We say that a variable v is *supported* by a clause C if the term that it replaced occurs within a mixed term in C , that is, if the auxiliary equation defining v is part of one of the projections $\neg C \downarrow \phi_i$. Note that a variable may be unsupported even if the corresponding subterm occurs in the clause, e.g., v_b is not supported by $f(b) \neq b$, because the literal contains no mixed term and thus there is no auxiliary equation for v_b . A variable may only appear unbound in the partial interpolant of a clause C if it is supported by C . Thus, in a post-processing step, we bind variables that are not supported by the current clause but still occur in a partial interpolant. An existential quantifier is introduced around the interpolant I_i for each unsupported variable v_f where the head function symbol f appears in A_i . Similarly, a universal quantifier is introduced for v_f if f appears in B_i . If multiple variables become unsupported at the same time, we need to be careful to add the quantifiers in the right order. If one of the variables corresponds to a subterm of the other, then the variable for the outer term is defined using the variable of the subterm. Hence its quantifier needs to appear after the quantifier for the variable of the outer term. Since we add quantifiers from inside to outside, we need to add the quantifiers in inverse dependency order, i.e., the quantifier for the variable corresponding to the outermost term needs to be added first.

5. Interpolation for Quantified Formulas

In this section, we show how to compute partial sequence interpolants for the different clause types involved in a proof of unsatisfiability for quantified input formulas. The overall algorithm follows a simple common theme: first compute the interpolants using only the colors we assigned to literals and ignore the symbol condition. Then introduce the fresh variable v_t for every term t that violates the symbol condition. Finally, introduce quantifiers for each variable v_t that is unsupported by the current clause, as outlined in Section 4.2. In the following, we concretize this for the different kind of clauses in the resolution proof.

Input Clauses. Input clauses cannot contain mixed terms. The interpolant I_i of an input clause C is either $\neg(\neg C \setminus A_i)$ if the clause C is part of A_i , or $\neg C \setminus B_i$ if C is part of B_i . Here $\neg C \setminus A_i$ removes all literals from C that do not have any color from A_i . For Pudlák’s algorithm where each literal is colored with all partitions it occurs in, all literals are removed and the interpolants are always either \top or \perp .

Instantiation Clauses. We say that an instantiation clause originates from the partition in which its quantified formula occurs in. We extend the definition of $\neg C \setminus A_i$ to handle a j, k -mixed equality $a = b$ as follows: if both a, b are in A_i , then $(\ell \setminus A_i) \equiv \top$, if both are in B_i , then $(\ell \setminus A_i) \equiv \ell$, and if a is in A_i and b in B_i , then $(a = b \setminus A_i) \equiv x = b$ and $(a \neq b \setminus A_i) \equiv \neg EQ(x, b)$. Analogously, we define $\neg C \setminus B_i$. The partial sequence interpolant I_i is then computed in three steps.

1. Set I_i to $\neg(\neg C \setminus A_i)$ if C originates from A_i , and to $\neg C \setminus B_i$, otherwise.
2. Replace every subterm t in I_i that must not occur in I_i according to the symbol condition by its corresponding fresh variable v_t .
3. Introduce quantifiers for each variable that is not supported by C .

Theory Lemmas. Our proof tree can contain theory lemmas for transitivity and congruence.

$$t_1 \neq t_2 \vee \dots \vee t_{n-1} \neq t_n \vee t_1 = t_n \quad (\text{transitivity})$$

$$t_1 \neq t'_1 \vee \dots \vee t_n \neq t'_n \vee f(t_1, \dots, t_n) = f(t'_1, \dots, t'_n) \quad (\text{congruence})$$

Transitivity lemmas are interpolated by summarizing equality chains involving A_i -colored literals in the negation of the lemma. If the single equality of the transitivity lemma is mixed, the chain $EQ(x, a_1) \wedge a_1 = \dots = a_n$ is summarized by $EQ(x, a_n)$. As the ends of the A_i -colored chains are also B_i -colored (the boundary terms appear in B_i -colored literals), this satisfies the symbol condition except when a mixed term contains a subterm from a different partition. To fix the interpolant, all subterms t from the wrong partition are replaced by their corresponding variable v_t . Our construction guarantees that these variables are supported by C .

For congruence lemmas, the interpolant depends on the color of the equality $f(t_1, \dots, t_n) = f(t'_1, \dots, t'_n)$. If it is colored with a partition from A_i , the interpolant is $\neg(\neg C \setminus A_i)$ and if it is colored with a partition from B_i , the interpolant is $\neg C \setminus B_i$. If the equality is a mixed equality where $f(t_1, \dots, t_n)$ is from A_i , then f is shared and the interpolant is $EQ(x, f(x_1, \dots, x_n))$ where x is the variable for the projection of the mixed equality, and for $j = 1, \dots, n$ the term x_j is the auxiliary variable of $t_j = t'_j$, if that equality is mixed, or t_j if the equality is colored with B_i , or t'_j if it is colored with A_i . Again, the subterms t from the wrong partition are replaced by their corresponding variable v_t , which is supported by C .

Resolution Clauses. The partial interpolant for a resolution node is computed by the rule (resolution) in Section 3. Since the resolution rule removes a literal from both antecedent clauses, the resolvent may no longer support all variables occurring in the interpolant. So in the last step, we add quantifiers for all unsupported variables as outlined in Section 4.2.

Example 4. We illustrate the algorithm on our running example (Example 2).

$$\phi_1 : \forall x. g(h(x)) = x \quad \phi_2 : \forall y. f(g(y)) \neq b \quad \phi_3 : \forall z. f(z) = z$$

The symbol f occurs in partitions 2–3, g in 1–2, h in 1, b in 2. Our goal is to compute sequence interpolants I_1, I_2 where ϕ_1 implies I_1 , and I_1 and ϕ_2 imply I_2 , and I_2 and ϕ_3 imply \perp . Also I_1 may only contain the symbol g and I_2 only the symbol f .

Figure 1 gives an instantiation-based resolution proof for the unsatisfiability of $\phi_1 \wedge \phi_2 \wedge \phi_3$. Each clause C in the proof tree is annotated with its partial interpolant I_1, I_2 . In the following, we explain step by step how these interpolants were computed. First, we color the literals occurring in the proof. The literals $g(h(b)) = b$ and $f(g(h(b))) = b$ were already explained in Section 4.1. The literal $f(g(h(b))) = f(b)$ is colored the same way as $f(g(h(b))) = b$. Finally, the literal $f(b) = b$ is not mixed and is colored with 2. To summarize, the projection of these literals to the partitions 1–3 is as follows.

ℓ	$\ell \downarrow \phi_1$	$\ell \downarrow \phi_2$	$\ell \downarrow \phi_3$
$g(h(b)) = b$	$g(h(v_b)) = x$	$x = b \wedge v_b = b$	\top
$f(g(h(b))) = f(b)$	$v_h = h(v_b)$	$f(g(v_h)) = f(b) \wedge v_b = b$	\top
$f(g(h(b))) = b$	$v_h = h(v_b)$	$f(g(v_h)) = b \wedge v_b = b$	\top
$f(b) = b$	\top	$f(b) = b$	\top

The first input clause on the top left of the proof tree is from ϕ_1 , i.e., it is in A_1 and A_2 . By the rule for input clauses its interpolants are \perp, \perp . The projections of the negated instantiation clause $Inst$ on the top right are

$$\neg Inst \downarrow \phi_1 \equiv (\phi_1 \wedge EQ(x, g(h(v_b)))) \quad \neg Inst \downarrow \phi_2 \equiv (\neg EQ(x, b) \wedge v_b = b) \quad \neg Inst \downarrow \phi_3 \equiv \top.$$

For I_1 , the quantified formula is in A_1 , and the interpolant is first set to $\neg(\neg C \setminus A_1)$. Since $g(h(b)) = b$ is 1, 2-mixed, $(g(h(b)) \neq b \setminus A_1)$ is $\neg EQ(x, b)$ and I_1 is set to $EQ(x, b)$. In the final step, the forbidden symbol b is replaced by its shared variable v_b yielding $EQ(x, v_b)$. The variable v_b is supported by the clause and therefore not quantified. The interpolant I_2 is just \perp as it contains only literals with colors 1–2. Resolving the first two clauses builds the disjunction of the corresponding interpolants, because the literal is colored with partition 1.

The equality in the congruence lemma, in the second row in Figure 1, is colored with partition 2. Therefore, interpolant I_1 is first set to $\neg C \setminus B_1$, which is $g(h(b)) = x$. As the function symbol h may not occur in the interpolant, the term $h(b)$ is replaced by its auxiliary variable v_h . The interpolant I_2 is set to \perp since the lemma involves no literal colored with partition 2. The result is $g(v_h) = x, \perp$, which is indeed a sequence interpolant of $v_h = h(v_b) \wedge g(h(v_b)) = x, f(g(v_h)) \neq f(b) \wedge x = b \wedge v_b = b, \top$. The next resolution step involves a 1, 2-mixed literal. Therefore, we use the third case (*ruleEq*) of rule (*resolution*). The first interpolants of the antecedents are combined by replacing $EQ(x, v_b)$ with $g(v_h) = v_b$, yielding $g(v_h) = v_b$. All literals in the transitivity lemma (third row) are colored with partition 2. Therefore the resulting interpolants are \top, \perp . The resolution step combines the interpolants with conjunction or disjunction, respectively, as the literal is colored with partition 2.

The second instantiation clause is completely colored with partition 2 as is the corresponding input clause. The interpolants are just \top, \perp and resolving them with the other branch does not change the previous interpolants. However, since the literal $f(g(h(b))) \neq b$ gets removed in the resolution step, v_h and v_b are no longer supported. Therefore, quantifiers are introduced: the inner existential quantifier for the 1-colored outermost term $v_h = h(v_b)$ and the outer universal quantifier for the 2-colored inner term $v_b = b$. Note that although the clause still contains b in the literal $f(b) \neq b$, this literal is not mixed and therefore does not support v_b .

The last instantiation clause is originating from ϕ_3 , hence I_1 is set to $\neg C \setminus B_1$ (which is \top) and I_2 is set to $\neg C \setminus B_2$. Since $f(b) \neq b$ is colored with partition 2, the second interpolant is

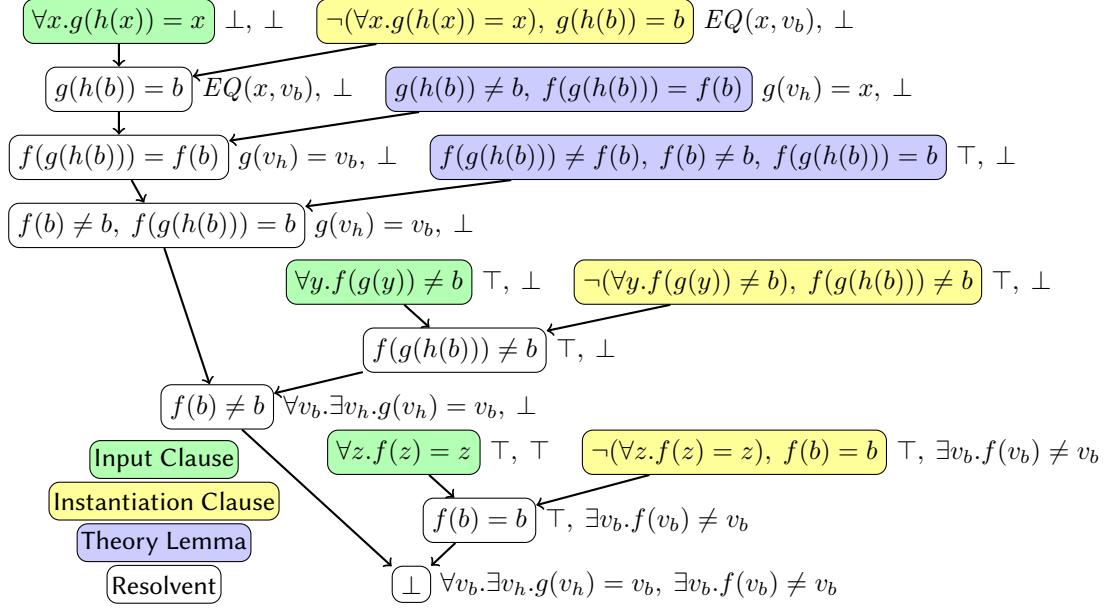


Figure 1: Refutation by resolution for Example 4 annotated by partial sequence interpolants.

$f(b) \neq b$. In the next step, b is replaced by its auxiliary variable v_b and since this is not supported by the clause, an existential quantifier must be introduced. So the partial sequence interpolant is $\top, \exists v_b.f(v_b) \neq v_b$. This is indeed a valid interpolant for the sequence $\top, f(b) \neq b, \forall z.f(z) = z$.

In the last two resolution steps, the interpolants are combined with conjunction or disjunction. Since for each resolution step one of the interpolants is \top or \perp , the other interpolant is not changed. The final sequence interpolant, i.e., $\forall v_b.\exists v_h.g(v_h) = v_b, \exists v_b.f(v_b) \neq v_b$, can be found in the annotation of the empty clause.

6. Conclusion and Future Work

We presented an algorithm to obtain inductive sequences of interpolants from instantiation-based refutation proofs of quantified formulas in the theory of equality. In contrast to most existing algorithms that allow for interpolation of quantified formulas, our approach works on non-local proofs. It neither requires to restrict the inference done by the SMT solver, nor does it require transformations of the proof tree. Mixed terms occurring in a proof tree due to quantifier instantiations are directly treated using virtual purification. To obtain (partial) interpolants satisfying the symbol condition, local terms are replaced by auxiliary variables that are eventually bound by quantifiers.

Our algorithm computes sequence interpolants from a single proof of unsatisfiability instead of repeatedly performing binary interpolation on several proof trees. Implementation of the algorithm in SMTInterpol [24] is ongoing work. We plan to extend our algorithm to tree interpolants as well as to other theories including linear arithmetic.

Acknowledgments

This work was partially supported by the German Research Council (DFG) under HO 5606/1-2.

References

- [1] T. A. Henzinger, R. Jhala, R. Majumdar, K. L. McMillan, Abstractions from proofs, in: N. D. Jones, X. Leroy (Eds.), Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, January 14-16, 2004, ACM, 2004, pp. 232–244. URL: <https://doi.org/10.1145/964001.964021>. doi:10.1145/964001.964021.
- [2] K. L. McMillan, Lazy abstraction with interpolants, in: T. Ball, R. B. Jones (Eds.), Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings, volume 4144 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 123–136. URL: https://doi.org/10.1007/11817963_14. doi:10.1007/11817963_14.
- [3] K. L. McMillan, Quantified invariant generation using an interpolating saturation prover, in: C. R. Ramakrishnan, J. Rehof (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings, volume 4963 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 413–427. URL: https://doi.org/10.1007/978-3-540-78800-3_31. doi:10.1007/978-3-540-78800-3_31.
- [4] M. Heizmann, J. Hoenicke, A. Podelski, Nested interpolants, in: POPL, ACM, 2010, pp. 471–482.
- [5] M. N. Seghir, A. Podelski, T. Wies, Abstraction refinement for quantified array assertions, in: J. Palsberg, Z. Su (Eds.), Static Analysis, 16th International Symposium, SAS 2009, Los Angeles, CA, USA, August 9-11, 2009. Proceedings, volume 5673 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 3–18. URL: https://doi.org/10.1007/978-3-642-03237-0_3. doi:10.1007/978-3-642-03237-0_3.
- [6] R. Blanc, A. Gupta, L. Kovács, B. Kragl, Tree interpolation in Vampire, in: K. L. McMillan, A. Middeldorp, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings, volume 8312 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 173–181. URL: https://doi.org/10.1007/978-3-642-45221-5_13. doi:10.1007/978-3-642-45221-5_13.
- [7] P. Pudlák, Lower bounds for resolution and cutting plane proofs and monotone computations, *J. Symb. Log.* 62 (1997) 981–998. URL: <https://doi.org/10.2307/2275583>. doi:10.2307/2275583.
- [8] K. L. McMillan, An interpolating theorem prover, in: K. Jensen, A. Podelski (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings,

- volume 2988 of *Lecture Notes in Computer Science*, Springer, 2004, pp. 16–30. URL: https://doi.org/10.1007/978-3-540-24730-2_2. doi:10.1007/978-3-540-24730-2_2.
- [9] G. Yorsh, M. Musuvathi, A combination method for generating interpolants, in: R. Nieuwenhuis (Ed.), *Automated Deduction - CADE-20*, 20th International Conference on Automated Deduction, Tallinn, Estonia, July 22-27, 2005, Proceedings, volume 3632 of *Lecture Notes in Computer Science*, Springer, 2005, pp. 353–368. URL: https://doi.org/10.1007/11532231_26. doi:10.1007/11532231_26.
- [10] A. Cimatti, A. Griggio, R. Sebastiani, Efficient interpolant generation in satisfiability modulo theories, in: C. R. Ramakrishnan, J. Rehof (Eds.), *Tools and Algorithms for the Construction and Analysis of Systems*, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings, volume 4963 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 397–412. URL: https://doi.org/10.1007/978-3-540-78800-3_30. doi:10.1007/978-3-540-78800-3_30.
- [11] A. Fuchs, A. Goel, J. Grundy, S. Krstic, C. Tinelli, Ground interpolation for the theory of equality, in: S. Kowalewski, A. Philippou (Eds.), *Tools and Algorithms for the Construction and Analysis of Systems*, 15th International Conference, TACAS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings, volume 5505 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 413–427. URL: https://doi.org/10.1007/978-3-642-00768-2_34. doi:10.1007/978-3-642-00768-2_34.
- [12] J. Christ, J. Hoenicke, A. Nutz, Proof tree preserving interpolation, in: TACAS, volume 7795 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 124–138.
- [13] J. Christ, J. Hoenicke, Proof tree preserving tree interpolation, *J. Autom. Reasoning* 57 (2016) 67–95.
- [14] J. Christ, J. Hoenicke, Instantiation-based interpolation for quantified formulae, in: N. Bjørner, R. Nieuwenhuis, H. Veith, A. Voronkov (Eds.), *Decision Procedures in Software, Hardware and Bioware*, 18.04. - 23.04.2010, volume 10161 of *Dagstuhl Seminar Proceedings*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2010. URL: <http://drops.dagstuhl.de/opus/volltexte/2010/2735/>.
- [15] M. P. Bonacina, M. Johansson, On interpolation in automated theorem proving, *J. Autom. Reason.* 54 (2015) 69–97. URL: <https://doi.org/10.1007/s10817-014-9314-0>. doi:10.1007/s10817-014-9314-0.
- [16] L. Kovács, A. Voronkov, First-order interpolation and interpolating proof systems, in: T. Eiter, D. Sands (Eds.), *LPAR-21*, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017, volume 46 of *EPiC Series in Computing*, EasyChair, 2017, pp. 49–64. URL: <https://easychair.org/publications/paper/r2j>.
- [17] B. Gleiss, L. Kovács, M. Suda, Splitting proofs for interpolation, in: L. de Moura (Ed.), *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction*, Gothenburg, Sweden, August 6-11, 2017, Proceedings, volume 10395 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 291–309. URL: https://doi.org/10.1007/978-3-319-63046-5_18. doi:10.1007/978-3-319-63046-5_18.
- [18] W. Craig, Three uses of the Herbrand-Gentzen theorem in relating model theory and

- proof theory, *J. Symb. Log.* 22 (1957) 269–285. URL: <https://doi.org/10.2307/2963594>. doi:10.2307/2963594.
- [19] M. P. Bonacina, M. Johansson, Interpolation systems for ground proofs in automated deduction: a survey, *J. Autom. Reason.* 54 (2015) 353–390. URL: <https://doi.org/10.1007/s10817-015-9325-5>. doi:10.1007/s10817-015-9325-5.
- [20] L. Kovács, A. Voronkov, First-order theorem proving and Vampire, in: N. Sharygina, H. Veith (Eds.), *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 1–35. URL: https://doi.org/10.1007/978-3-642-39799-8_1. doi:10.1007/978-3-642-39799-8_1.
- [21] L. M. de Moura, N. Bjørner, Engineering DPLL(T) + saturation, in: A. Armando, P. Baumgartner, G. Dowek (Eds.), *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008. Proceedings*, volume 5195 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 475–490. URL: https://doi.org/10.1007/978-3-540-71070-7_40. doi:10.1007/978-3-540-71070-7_40.
- [22] L. Kovács, A. Voronkov, Interpolation and symbol elimination, in: R. A. Schmidt (Ed.), *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings*, volume 5663 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 199–213. URL: https://doi.org/10.1007/978-3-642-02959-2_17. doi:10.1007/978-3-642-02959-2_17.
- [23] R. Bruttomesso, S. Ghilardi, S. Ranise, Quantifier-free interpolation in combinations of equality interpolating theories, *ACM Trans. Comput. Log.* 15 (2014) 5:1–5:34. URL: <https://doi.org/10.1145/2490253>. doi:10.1145/2490253.
- [24] J. Christ, J. Hoenicke, A. Nutz, SMTInterpol: An interpolating SMT solver, in: *SPIN*, volume 7385 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 248–254.