Extending Proof Tree Preserving Interpolation to Sequences and Trees (Work In Progress)

Jürgen Christ
Albert-Ludwigs-Universität Freiburg
christj@informatik.uni-freiburg.de

Jochen Hoenicke
Albert-Ludwigs-Universität Freiburg
hoenicke@informatik.uni-freiburg.de

Abstract

We present ongoing work to extend proof tree preserving interpolation to inductive sequences and tree interpolants. We present an algorithm to compute tree interpolants inductively over a resolution proof. Correctness of the algorithm is justified by the concept of partial tree interpolants and the appropriate definition of a projection function for conjunctions of literals onto nodes of the tree. We observe great similarities between the interpolation rules for binary interpolation and those for tree interpolation.

1 Introduction

Craig interpolation is widely used in model checking [6, 13, 15]. Instead of binary interpolation [5], these techniques use inductive sequences or trees of interpolants. Tree interpolants arise from model-checking recursive and concurrent programs in a natural way. An execution of the program with procedures can be represented as a nested trace, where the statement after a procedure call has two predecessors, the return statement of the called procedure and the procedure call itself. To reason about correctness in a modular way requires combining the function summary with the intermediate assertion before the procedure call. This leads naturally to a tree-like structure [11, 12]. Tree interpolants are also useful to approximate function summaries for incremental update checking [17].

Similarly modular reasoning about concurrent programs need interference free proofs or assume-guarantee reasoning. The proof of an intermediate assertions can depend on the previous assertion of the same thread and the guarantees provided by the other threads. Thus, an unfolding of the parallel program has again a tree-like shape. Other uses of interpolants in model-checking are data-flow graph based method [8] which compute tree interpolants for an unfolded data-flow tree.

Although tree interpolants are widely used, only a few tools are able to produce them without the need for repeated applications of binary interpolation to different interpolation problems. The techniques used by these tools to ensure correctness of inductive sequences and trees of interpolants is not well documented.

In this paper, we extend the recently proposed technique of proof tree preserving interpolation [3, 4] to compute inductive sequences and trees of interpolants. The key idea of this technique is to define partial interpolants in the context of mixed literals that cannot be assigned to a partition of the input problem. This is achieved by introducing auxiliary variables and defining a projection function that splits mixed literals using auxiliary variables. We extend the projection function to trees of formulae and the notion of partial interpolants to partial tree interpolants. This allows us to compute tree interpolants inductively over the proof tree and is an essential step towards the correctness proof of the algorithm.

We show that tree interpolants can be computed for every node separately provided that some precautions are met for leaf interpolation. We give a rule that allows for inductively computing partial tree interpolants over a given proof. We observe that this rule for tree interpolation is identical to applying the rule from [3] for binary interpolation for every node separately.

Related Work. Only a few publications describe how to compute tree interpolants. Gupta et al. [10] describe how to solve a set of recursion-free Horn clauses over the theories of uninterpreted functions and linear real arithmetic. This corresponds directly to the tree interpolation problem for a conjunctive formula that does not contain negated equalities. They have stricter syntactic restrictions for the partial solutions and a rule for combining these, which is similar to our combination rule for partial interpolants. Our algorithm computes the same solutions when working on this fragment, however, we allow more input problems and our method is complete even for linear integer arithmetic.

The interpolating version of Z3 (iZ3) [1] can extract tree interpolants although there is no publication describing how it computes tree interpolants. iZ3 poses additional restrictions on the occurrence of symbols in the input and treats every non-constant function symbol as global symbol. In contrast to the method presented in this paper, iZ3 cannot interpolate linear real arithmetic and is incomplete on linear integer arithmetic.

2 Notation

We assume the usual notation and terminology of many sorted first-order logic. We consider the quantifier-free fragments of the theory of uninterpreted functions with atoms of the form $s_1 = s_2$ for two terms s_1 and s_2, and linear arithmetic over integer and reals. To allow for quantifier-free interpolation of integer arithmetic, we extend the signature with the functions $\left\lfloor \frac{\cdot}{k} \right\rfloor$ for all integers $k \geq 2$. By $\mathbb{Q}_\varepsilon$ we denote the rational numbers including an infinitesimal part [7], i. e., $\mathbb{Q}_\varepsilon = \mathbb{Q} \cup \{c - \varepsilon \mid c \in \mathbb{Q}\}$. We assume that the literals of linear arithmetic are normalised to the form $\sum_i c_i a_i \leq c$ where a_i are constant symbols, $c_i \in \mathbb{Z}$, and $c \in \mathbb{Z}$ (for integers) or $c \in \mathbb{Q}_\varepsilon$ (for reals). We use $t \leq c - \varepsilon$ to denote $t < c$. Note that in linear arithmetic the negation of an atom $\neg t \leq c$ can be expressed as $-t \leq -c - \varepsilon$.

For formulae we use the symbols F and I (for interpolants). By $symb(F)$ we denote the set of non-logical symbols occurring in F. We denote constant symbols by a, b, terms by s, t, numerical constants by c, variables by x, and set-valued variables by X. By $I[F]$ we denote a formula that contains the subformula F only positively. By $I(t)$ we denote a formula that contains a term t. Given two clauses $\ell \vee C_1$ and $\neg\ell \vee C_2$ (called antecedents), the resolution rule

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

concludes $C_1 \vee C_2$. In the context of SMT, a resolution proof is a derivation of the empty clause $\bot$ from the input clauses, theory lemmas, and theory combination clauses using only the resolution rule.

3 Proof Tree Preserving Interpolation

A binary interpolation problem consists of a pair of formulae (A, B). An interpolant exists if $A \wedge B$ is unsatisfiable and the theory admits interpolation. An interpolant I satisfies (i) $A \models I$, (ii) $B \wedge I$ is unsatisfiable, and (iii) $symb(I) \subseteq symb(A) \cap symb(B)$. In this section we briefly review proof tree preserving interpolation [4]. This technique extends the interpolation algorithms given by Pudlák [16] and McMillan [14] to mixed literals, i. e., literals containing symbols occurring only in formula A and symbols occurring only in formula B. For a given binary interpolation problem (A, B), proof tree preserving interpolation algorithms define two projection functions for every literal ℓ. The first projection, $\ell \lfloor A$, projects the literal onto A, the second, $\ell \lfloor B$ projects onto B. The algorithms from Pudlák and McMillan define the projection functions for non-mixed literals and require the invariant $\ell \leftrightarrow (\ell \lfloor A) \wedge (\ell \lfloor B)$. Usually, one of the projections is ℓ and the other is $\top$. The projection is extended to conjunctions of literals.

If ℓ is mixed, we cannot split the literal into two conjuncts such that the first conjunct only contains symbols occurring in A and the second conjunct only contains symbols occurring in B. We extended the projection function to mixed literals by introducing *auxiliary variables*. The invariant satisfied by our projection function is the existential closure $\ell \leftrightarrow \exists x.\ ((\ell \lfloor A) \wedge (\ell \lfloor B))$ of the invariant above. For example, the mixed literal $a \leq b$ may be split into $a \leq x$ and $x \leq b$ using the auxiliary variable x. The auxiliary variable is used to connect the projections to the A and B part, similarly (but orthogonal) to Nelson-Oppen theory combination.

To compute an interpolant from a resolution proof, we compute *partial interpolants* for every node in the resolution proof: Given a clause C in the resolution proof of $A \wedge B$, a partial interpolant I_C is an interpolant for $(A \wedge (\neg C \lfloor A), B \wedge (\neg C \lfloor B))$. Computation of partial interpolants differs between input clauses, theory lemmas, and results of resolution steps.

Since input clauses do not contain mixed literals, we can use the usual syntactic rules to compute partial interpolants [14]. Unfortunately, for theory lemmas the situation is different. These lemmas are the source of mixed literals in SMT proofs and, hence, need special procedures to compute partial interpolants. We compute an interpolant for $(\neg C \lfloor A, \neg C \lfloor B)$, i. e., an interpolant of the theory conflict (the negation of the theory lemma) projected onto A and B. The conflict is interpolated using a theory specific interpolator. Note that the auxiliary variables introduced during projection of mixed literals may occur in the interpolant.

The algorithms from Pudlák and McMillan give rules to compute a partial interpolant for the consequence of a resolution step given partial interpolants for the antecedents. The resulting partial interpolant is either a conjunction, a disjunction, or a multiplexer depending on whether the pivot literal occurs only in A, only in B, or in both. We extend these rules to handle mixed literals. The partial interpolants for clauses containing mixed literals contain the auxiliary variables introduced by the projection function only in specific syntactically restricted sub-formulae. The structure of these formulae makes it possible to remove the auxiliary variable once the corresponding mixed literal is the pivot of a resolution step. Details are out of the scope of this paper and can be found in [3, 4]. In this paper we will give slightly different syntactic restrictions in Section 5.4 and define the interpolation rules in Section 5.5.

4 Tree Interpolation

A tree interpolation problem is specified by a (directed) tree $T = (V, E)$ where V is a set of nodes, $E \subseteq V \times V$ is a set of edges (pointing from child to parent node), and $L : V \to \mathit{Formula}$ is a labelling function that assigns a formula to every node in the tree. With $st(v) := \{w \mid (w, v) \in E^*\}$ (where E^* is the reflexive transitive closure of E) we denote the set of nodes in the subtree with the root v. A solution to the tree interpolation problem exists if the conjunction $\bigwedge_{v \in V} L(v)$ is unsatisfiable and the theories involved support interpolation. A labelling function $I : V \to \mathit{Formula}$ is called a *tree interpolant* [1] for T and L if the following properties hold:

1. $I(v_r) \equiv \bot$ where v_r is the root of T,
2. $(\bigwedge_{(w,v) \in E} I(w)) \wedge L(v) \models I(v)$ for all $v \in V$,
3. for every node v, all symbols in $I(v)$ occur both inside the subtree rooted at v and outside this subtree, i. e., $\mathit{symb}(I(v)) \subseteq (\bigcup_{w \in st(v)} \mathit{symb}(L(w))) \cap (\bigcup_{w' \notin st(v)} \mathit{symb}(L(w')))$.

By $lca(v, w)$, we denote the least common ancestor of v and w, i. e., the first common node in the tree that is encountered when traversing the tree towards the root from v and w. Obviously, for any pair of nodes v and w, the least common ancestor $lca(v, w)$ is unique. Every non-empty set of nodes has a unique least common ancestor, which can be computed by repeatedly applying the binary lca function.

We define for every node $v \in V$ the set of symbols $symb(v)$, that we could add to $L(v)$ without changing the symbol condition in the definition of tree interpolants. For a symbol a occurring in two nodes $v_1, v_2 \in V$, i. e., $a \in symb(L(v_1)) \cap symb(L(v_2))$, we add a to $symb(v)$ for every node v on the path from v_1 or v_2 to their least common ancestor $lca(v_1, v_2)$.

Lemma 1. *Replacing $symb(L(w))$ with $symb(w)$ does not change Condition 3. of tree interpolants. In particular*

$$\bigcup_{w \in st(v)} symb(L(w)) = \bigcup_{w \in st(v)} symb(w) \quad \text{and} \quad \bigcup_{w' \notin st(v)} symb(L(w')) = \bigcup_{w' \notin st(v)} symb(w').$$

Inductive Sequences and Tree Interpolation

Given a sequence of n formulae $F_1, \ldots, F_n$ such that $\bigwedge_{i=1}^{n} F_i$ is unsatisfiable, an inductive sequence of interpolants is a sequence of $n+1$ formulae $I_0, \ldots, I_n$ such that (i) $I_0 \equiv \top$, (ii) $I_n \equiv \bot$, (iii) $I_{i-1} \wedge F_i \models I_i$ for $1 \leq i \leq n$, and (iv) $symb(I_i) \subseteq (\bigcup_{j=1}^{i} symb(F_j)) \cap (\bigcup_{j=i+1}^{n} symb(F_j))$ for $1 \leq i \leq n$. Such a sequence can be computed either by repeatedly computing interpolants according to condition (iii), or by carefully extracting all $n+1$ interpolants for this sequence from one proof.

Theorem 1 (Sequence Interpolation is Tree Interpolation)**.** *Inductive sequences of interpolants are a special case of tree interpolants.*

Since we can recast a sequence interpolation problem as a tree interpolation problem we will only extend proof tree preserving interpolation to tree interpolation. The extension to sequences is left to the reader.

5 Adapting Proof Tree Preserving Interpolation to Tree Interpolation

To adapt proof tree preserving interpolation to tree interpolation we have to adapt the projection function used in binary interpolation to trees. Furthermore, we have to show that the interpolating resolution rules are still valid, i. e., that the interpolants computed by these rules satisfy the properties of partial tree interpolants. Throughout this section, let $T = (V, E)$ and L be a tree interpolation problem and $v, v_c, v_p \in V$ be nodes in the tree.

The projection function $\ell \downarrow v$ projects a literal ℓ onto the node $v \in V$ of the tree defining the interpolation problem. As in [4], we introduce auxiliary variables x for mixed literals ℓ. For tree interpolation, we introduce a fresh variable for each node $v \in V$ where the literal is mixed. The auxiliary variables introduced for a node are shared with the parent node; for each edge $(v_c, v_p) \in E$, the projection $\ell \downarrow v_p$ also contains the auxiliary variables of node v_c. The projection of a literal ℓ with the auxiliary variables $\vec{x}$ must satisfy two conditions. First, the conjunction of the projections of a literal ℓ onto every node in the tree $\bigwedge_{v \in V} \ell \downarrow v$ is equivalent to ℓ, i. e.,

$$\ell \iff \exists \vec{x}. \bigwedge_{v \in V} \ell \downarrow v \text{ where } \vec{x} \text{ is the set of auxiliary variables introduced for } \ell.$$

Second, $\ell \downarrow v$ must only contain theory symbols, symbols from $symb(v)$, and the auxiliary variables introduced for ℓ and v or the children of v.

Our algorithm computes a tree interpolant from the resolution proof by computing a partial tree interpolant for every clause C occurring in the proof tree. Partial tree interpolants are defined using the projection function as follows.

Definition 1 (Partial Tree Interpolant)**.** A *partial tree interpolant* for a clause C is a tree interpolant for T and L' where $L'(v) = L(v) \wedge (\neg C \downarrow v)$ for $v \in V$.

Obviously, a partial tree interpolant of the empty clause is a tree interpolant of T and L. Note that for an intermediate clause C a partial tree interpolant I may contain the auxiliary variables of the literals occurring in the clause C. To be more precise, for a node $v \in V$ the interpolant $I(v)$ may contain the auxiliary variables introduced for ℓ and v.

5.1 Adapting Propositional Interpolation Algorithms

In this section we show how the rules for propositional interpolation [16, 14] can be adapted to tree interpolation. Since we only consider the propositional case, we assume no literal is mixed. In this case we do not introduce any auxiliary variables. The projection $\ell \downarrow v$ is either ℓ or $\top$ and there must be at least one node $v \in V$ with $\ell \downarrow v = \ell$. These conditions guarantee $\ell \iff \bigwedge_{v \in V} \ell \downarrow v$. McMillan's and Pudlák's algorithm only differ in the projection function. For Pudlák's algorithm we set $\ell \downarrow v = \ell$ if and only if $\ell \in \mathit{symb}(v)$. For McMillan's algorithm we set $\ell \downarrow v = \ell$ only for the least common ancestor of the nodes $v \in V$ with $\ell \in \mathit{symb}(v)$.

Tree interpolants will be computed recursively over the resolution proof of the conjunction of the labels of the tree. As in the binary case, we devise special rules to compute partial tree interpolants for leaves of the proof tree. We compute partial tree interpolants for the resolution steps using the following rule.

$$\frac{\ell \vee C_1 : I_1 \qquad \neg\ell \vee C_2 : I_2}{C_1 \vee C_2 : I_3}, \text{ where } I_3(v) = \begin{cases} I_1(v) \vee I_2(v) & \text{if } \ell \downarrow v' = \top \text{ for all } v' \notin \mathit{st}(v) \\ I_1(v) \wedge I_2(v) & \text{if } \ell \downarrow v' = \top \text{ for all } v' \in \mathit{st}(v) \\ (I_1(v) \vee \ell) \wedge & \\ (I_2(v) \vee \neg\ell) & \text{otherwise} \end{cases}$$

The rule above can be interpreted as applying Pudlák's resp. McMillan's algorithm for each node separately. The condition $\ell \downarrow v' = \top$ for all $v' \notin \mathit{st}(v)$ means that the literal does not occur outside the subtree of v, i. e., the literal is A-local if we see the subtree of v as the A partition of a binary interpolation problem. Likewise the condition $\ell \downarrow v' = \top$ for $v' \in \mathit{st}(v)$ means that the literal is B-local. If neither is the case, the literal is shared.

Lemma 2. *The rule above is correct, i. e., if I_1 is a partial tree interpolant of $\ell \vee C_1$ and I_2 a partial tree interpolant of $\neg\ell \vee C_2$, then I_3 is a partial tree interpolant of $C_1 \vee C_2$.*

5.2 Occurrences of Symbols and Scope of Mixed Literals

An SMT proof may involve literals that are not in the original input formulae. These literals may be mixed, i. e., they contain symbols from different nodes. Let ℓ be a literal with $\mathit{symb}(\ell) = \{a_1, \ldots, a_n\}$. If for a node v, some symbols occur only inside the subtree rooted at v and some symbols occur only outside the subtree, we say that the literal ℓ *is mixed in* v. We denote with $\mathsf{mixed}(\ell)$ the set of all nodes v such that ℓ is mixed in v.

For a symbol a we overload *lca* and denote with $\mathit{lca}(a)$ the least common ancestor of all nodes $v \in V$ with $a \in \mathit{symb}(v)$. By the definition of $\mathit{symb}(v)$ for $v \in V$ we have $a \in \mathit{symb}(\mathit{lca}(a))$ and $a \notin \mathit{symb}(v)$ for $v \notin \mathit{st}(\mathit{lca}(a))$. For every symbol a, $\mathit{lca}(a)$ is the unique node such that all occurrences of a are in the subtree of this node and a occurs in the node itself. Having $a \in \mathit{symb}(\mathit{lca}(a))$ is the main reason why we defined $\mathit{symb}(v)$ in this way.

If a literal ℓ is mixed in some nodes we denote by $\mathsf{mixedparent}(\ell)$ the least common ancestor of nodes that have a child where ℓ is mixed. Then we can exactly characterise the set $\mathsf{mixed}(\ell)$ as follows:

Lemma 3. *Let ℓ be a literal that is mixed in some nodes and contains the symbols $a_1, \ldots, a_n$. Then*

$$\mathsf{mixed}(\ell) = \{v \in V \mid \exists i.\ 1 \leq i \leq n.\, lca(a_i) \in st(v) \text{ and } \mathsf{mixedparent}(\ell) \text{ is a proper ancestor of } v\}$$

5.3 Extending Projection

We now extend the projection functions to cope with mixed literals. For every literal ℓ and every node $v_j \in \mathsf{mixed}(\ell)$, an auxiliary variable x_j is introduced. The projection $\ell \downarrow v$ is chosen such that it is correct with respect to the following definition.

Definition 2. Let $\downarrow$ be a projection function. The projection function is *correct*, iff for all literals ℓ:

$$\ell \iff \exists \vec{x}.\ \bigwedge_{v \in V} \ell \downarrow v$$

where $\vec{x} = \{x_j \mid v_j \in \mathsf{mixed}(\ell)\}$ is the set of all auxiliary variables introduced for the literal ℓ.

5.3.1 Mixed Equalities

We start by giving the projection function for an equality literal $\ell :\equiv a_1 = a_2$. By Lemma 3, every node $v_p \in \mathsf{mixed}(\ell)$ lies on a path between $lca(a_i)$ and $\mathsf{mixedparent}(\ell)$ (for some $i \in \{1,2\}$). The a_i is unique, since v_p is not mixed if $lca(a_i) \in st(v_p)$ for both $i = 1,2$. For each node $v_p \in \mathsf{mixed}(\ell)$ we introduce an auxiliary variable x_p that captures the value of this a_i. The projection of ℓ achieves this by fixing the value x_p of a mixed node v_p to the value x_c of the (uniquely defined) child v_c that lies on the path to the unique $lca(a_i)$, or to the value of a_i if $v_p = lca(a_i)$. The projection of the node $\mathsf{mixedparent}(\ell)$ ensures that $a_1 = a_2$ by making the auxiliary variables of the corresponding children equal.

$$a_1 = a_2 \downarrow v_p = \begin{cases} x_{c_1} = x_{c_2} & \text{if } (v_{c_1}, v_p), (v_{c_2}, v_p) \in E \text{ and } v_{c_1}, v_{c_2} \in \mathsf{mixed}(\ell) \\ a_i = x_c & \text{if } (v_c, v_p) \in E,\ v_c \in \mathsf{mixed}(\ell), \text{ and } lca(a_i) = v_p \\ x_c = x_p & \text{if } (v_c, v_p) \in E,\ v_c, v_p \in \mathsf{mixed}(\ell) \\ a_i = x_p & \text{if } lca(a_i) = v_p,\ v_p \in \mathsf{mixed}(\ell) \text{ for some } i \in \{1,2\} \\ \top & \text{otherwise} \end{cases}$$

In the first two cases, we observe that a_1, a_2 occur inside the subtree of v_p. Hence, v_p is not mixed but has at least one mixed child. By Lemma 3, $v_p = \mathsf{mixedparent}(\ell)$. Usually, this means that there are exactly two child nodes v_{c_1} and v_{c_2} in which ℓ is mixed, one an ancestor of $lca(a_1)$ and one an ancestor of $lca(a_2)$ (first case). However, it is also possible that $v_p = lca(a_i)$ for one of the two symbols a_1, a_2 (second case). In both cases, the corresponding projection ensures that $a_1 = a_2$.

When ℓ is mixed in v_p, the third or the fourth case applies. Then, for exactly one $i \in \{1,2\}$, $lca(a_i)$ occurs in the subtree of v_p. If already $v_p = lca(a_i)$, we are in the forth case. Otherwise, the third case applies and v_c is the child containing $lca(a_i)$. Both projections ensure that $x_p = a_i$ for the value a_i that occurs in the subtree of v_p. The last case only applies if ℓ is not mixed in v_p and $v_p \neq \mathsf{mixedparent}(\ell)$. The correctness of this projection is proved in the appendix.

The projection of a disequality $a_1 \neq a_2$ is tricky. Instead of a plain auxiliary variable x_p we introduce a set-valued auxiliary variable X_p for every node v_p where the literal is mixed. For such a node v_p one a_i ($i = 1,2$) occurs only in the subtree of v_p and the other only outside the subtree. The projections of the literal enforce that X_p contains the value a_i that occurs in the subtree of node v_i and does not contain the other value. It may contain other values different from a_1 and a_2 when the value of a_i cannot be

expressed using only symbols shared between the subtree of v_i and its complement. The projections of $a_1 \neq a_2$ are defined as follows.

$$a_1 \neq a_2 \downarrow v_p = \begin{cases} X_{c_1} \cap X_{c_2} = \emptyset & \text{if } (v_{c_1}, v_p), (v_{c_2}, v_p) \in E \text{ and } v_{c_1}, v_{c_2} \in \mathsf{mixed}(\ell) \\ a_i \notin X_c & \text{if } (v_c, v_p) \in E,\ v_c \in \mathsf{mixed}(\ell), \text{ and } lca(a_i) = v_p \\ X_c \subseteq X_p & \text{if } (v_c, v_p) \in E,\ v_c, v_p \in \mathsf{mixed}(\ell) \\ a_i \in X_p & \text{if } lca(a_i) = v_p,\ v_p \in \mathsf{mixed}(\ell) \text{ for some } i \in \{1,2\} \\ \top & \text{otherwise} \end{cases}$$

Although the formulae are different, the cases are exactly the same as for equality. The fourth and third formulae ensure that X_p contains the value a_i that occurs in the subtree of v_p. With this property, each of the first two formulae ensures that $a_1 \neq a_2$.

Despite the definition of the projection function, we do not need set-theoretic reasoning in our solver. The projections are only used to prove the correctness of the resolution rule and the theory specific interpolation rules. The theory specific interpolation algorithm is specialised to conflicts arising from the Congruence Closure algorithm. Such conflicts may only contain a single disequality $a_1 \neq a_2$ and a chain of equalities that force the value of a_1 and a_2 to be equal. For each node v_p where this literal is mixed, we use the usual algorithm [9] to summarise equality chains originating from the subtree of v_p, which gives us a formula of the form $a_i = s_1 \wedge s_2 = s_3 \wedge \cdots \wedge s_{n-1} = s_n$ where a_i is the symbol that occurs in the subtree of v_p. The projection of the mixed literal to the subtree of v_p has the form $a_i \in X_{c_1} \wedge \cdots \wedge X_{c_k} \subseteq X_p$, which can be summarised by $a_i \in X_p$. The interpolant returned by our algorithm is $s_1 \in X_p \wedge s_2 = s_3 \cdots \wedge s_{n-1} = s_n$. Note, that if the conflict also contains mixed *equalities*, the plain auxiliary variable x_p introduced by that equality may occur in a shared terms s_i.

The syntactic restriction we pose on the partial invariants is that X_p occurs only in a literal $s \in X_p$, where s is an arbitrary term (not containing a set-valued variable). In particular, $s \in X_p$ may occur only positively. To get a similar notation as in our previous paper [3], we define $EQ(X_p, s) :\equiv s \in X_p$. On the other hand, a variable x_p introduced by a mixed equality may occur anywhere in the partial interpolant, even under a function application or in the s-part of an $EQ(X_p, s)$ term.

5.3.2 Mixed Inequalities

A linear inequality $\ell :\equiv c_1 a_1 + \ldots + c_n a_n \leq c$ is a comparison between a sum of n constant symbols a_i multiplied by a constant $c_i \in \mathbb{Z}$ and a constant $c \in \mathbb{Q}_\varepsilon$ for linear real arithmetic or $c \in \mathbb{Z}$ for linear integer arithmetic. We introduce an auxiliary variable x_j for every $v_j \in \mathsf{mixed}(\ell)$. We define a helper projection for the sum.

$$c_1 a_1 + \cdots + c_n a_n \downarrow v_p = \sum_{c \mid (v_c, v_p) \in E \wedge v_c \in \mathsf{mixed}(\ell)} x_c + \sum_{i \mid lca(a_i) = v_p} c_i a_i$$

Thus, the projection of the sum to v_p is the sum of all terms that occur in v_p for the last time and the sum of all auxiliary variables for all mixed child nodes. The projection of ℓ to the nodes v is defined as follows.

$$\ell \downarrow v_p = \begin{cases} (c_1 a_1 + \cdots + c_n a_n \downarrow v_p) \leq c & \text{if } v_p = \mathsf{mixedparent}(\ell) \\ (c_1 a_1 + \cdots + c_n a_n \downarrow v_p) \leq x_p & \text{if } v_p \in \mathsf{mixed}(\ell) \\ \top & \text{otherwise} \end{cases}$$

Again, the introduced auxiliary variable is shared between the node where it was introduced and its parent node. It is allowed to occur in the partial interpolant of its node but only in the special pattern $LA(s, k, F) :\equiv F$ which must occur positively in the interpolant. Here s is an affine sum of shared terms and auxiliary variables. Every variable x occurring in F must also appear in s with a positive coefficient

and F must be monotone in x, i. e., $x \geq x' \implies F(x) \to F(x')$. Finally we require that $F = \bot$ for $s > 0$ and $F = \top$ for $s < -k$.

The algorithm we present here is a slight improvement of the algorithm in [3]. We could use the same algorithm but the new one will compute slightly smaller interpolants. The basic difference is that $LA(s,k,F)$ was defined as $s \leq 0 \wedge (s \geq -k \to F)$, while in our new definition we assume that F is already false for $s > 0$ and true for $s < -k$. Also the monotonicity condition of F simplifies the correctness proof by avoiding the weak and strong interpolant that was needed in our previous paper.

5.4 Interpolation of Theory Lemmas

Our algorithm uses the Congruence Closure algorithm to produce conflicts in the theory of equality. We can compute the partial tree interpolant separately for every node of the tree. However, we must carefully assign every literal to a unique node of the tree. Then for every node the A part of the interpolation problem consists of all literals assigned to a node in the subtree and the B part consists of all other literals.

Usually an interpolant is computed as the summary of the A paths built from the conflict. In the presence of function congruence a more elaborated algorithm is needed [9]. This algorithm also works for mixed equality literals if they are split into their projections. The auxiliary variables can only occur in the interpolants of the nodes for which the variables were introduced.

However, for a mixed *disequality* it is not feasible to replace it by the projections of the literals as it involves new predicates and universally quantified formulae. Instead, they need to be treated separately. A conflict always involves exactly one disequality that contradicts an equality path. If this disequality is mixed, there is always an A path of equalities that start at the A part of the mixed disequality and ends at a shared symbol s. Instead of adding a summary equality, we add the literal $X(s)$, where X is the mixed predicate that was added for the mixed disequality in the current partition.

For inequalities we apply the algorithm of [3] for each partition to compute the tree interpolant. Mixed inequalities are replaced by their projections on the nodes of the tree. This sums up all inequalities that occur in the A part of the interpolation problem, i. e., the subtree of the node v for which we compute the interpolant. Again we need to ensure that every literal is assigned to a unique node.

Conjecture 1. *The computed partial interpolant will fulfil the invariants of partial tree interpolants.*

5.5 Extended Interpolation Rules

The interpolation rules for tree interpolants are a straight-forward extension of the interpolation rules for binary interpolation presented in Section 3. For every resolution step in the resolution proof of unsatisfiability, we compute for every node $v \in V$ an interpolant. If $\ell \vee C_1$ has partial tree interpolant I_1 and $\neg\ell \vee C_2$ has partial tree interpolant I_2, we can compute a tree interpolant I_3 for $C_1 \vee C_2$ by node-wise computing partial interpolants. For $v \in V$, $I_3(v)$ is a combination of the pivot literal ℓ and the partial interpolants $I_1(v)$ and $I_2(v)$ of the antecedents of the resolution step.

The computation of $I_3(v)$ from $I_1(v)$ and $I_2(v)$ is done by the same algorithm as for binary interpolation [3]. If the literal ℓ is not mixed in v we use the definition from Section 5.1. If the literal is mixed in v, we use a special interpolation rule $\mathsf{mixcomb}(\ell, I_1(v), I_2(v))$ that takes two partial interpolants (i. e., the labels of the corresponding partial tree interpolants at the node v) and computes a new partial interpolant for the resolvent. The partial interpolants $I_1(v)$ and $I_2(v)$ may contain the auxiliary variables introduced by ℓ in v, which may not occur in the resulting partial interpolant.

$$\frac{\ell \vee C_1 : I_1 \qquad \neg\ell \vee C_2 : I_2}{C_1 \vee C_2 : I_3}, \text{ where } I_3(v) = \begin{cases} I_1(v) \vee I_2(v) & \text{if } \ell \lfloor v' = \top \text{ for all } v' \notin st(v) \\ I_1(v) \wedge I_2(v) & \text{if } \ell \lfloor v' = \top \text{ for all } v' \in st(v) \\ \mathsf{mixcomb}(\ell, I_1(v), I_2(v)) & \text{if } v \in \mathsf{mixed}(\ell) \\ (I_1(v) \vee \ell) \wedge & \\ (I_2(v) \vee \neg\ell) & \text{otherwise} \end{cases}$$

For a mixed equality, our syntactic restriction of interpolants guarantees that the first interpolant is of the form $I_1[EQ(X,s_1)]\dots[EQ(X,s_n)]$, i. e., all occurrences of the auxiliary variable X are in a literal $EQ(X,s_i)$ occurring positively in the formula. The second interpolant $I_2(x)$ has no syntactic restrictions. The combined interpolant is obtained by replacing each literal $EQ(X,s_i)$ in I_1 by $I_2(s_i)$, which is expressed as

$$\mathsf{mixcomb}(a = b, I_1[EQ(X,s_1)]\dots[EQ(X,s_n)], I_2(x)) :\equiv I_1[I_2(s_1)]\dots[I_2(s_n)].$$

For an inequality, a partial interpolant of $\ell \vee C_1$ has the shape

$$I_1[LA(c_{11}x_1 + s_{11}, k_{11}, F_{11})]\dots[LA(c_{1n}x_1 + s_{1n}, k_{1n}, F_{1n})].$$

We use $LA_{1i}(x_1)$ as short-hand for $LA(c_{1i}x_1 + s_{1i}, k_{1i}, F_{1i})$ and write the formula above as $I_1[LA_{1i}(x_1)]$. Similarly, we write $I_2[LA_{2j}(x_2)]$ for a partial interpolant $I_2[LA_{21}(x_2)]\dots[LA_{2m}(x_2)]$ of $\neg\ell \vee C_2$. For each pair LA_{1i} and LA_{2j} we compute a formula $LA_{3ij} := LA(s_{3ij}, k_{3ij}, F_{3ij})$ such that $LA_{3ij} \iff \exists x.\, LA_{1i}(x) \wedge LA_{2j}(-x)$. This is possible since the value of an $LA(s,k,F)$ is only unknown for $-k \le s(x) \le 0$. In the integer case we can enumerate all possible values of x in this interval:

$$\begin{aligned} s_{3ij} &:= c_{2j}s_{1i} + c_{1i}s_{2j} \\ k_{3ij} &:= c_{2j}k_{1i} + c_{1i}k_{2j} + c_{1i}c_{2j} \\ F_{3ij} &:\equiv \bigvee_{i=0}^{\left\lceil \frac{k_{1i}+1}{c_{1i}} \right\rceil} F_{1i}\left(\left\lfloor \frac{-s_{1i}}{c_{1i}} \right\rfloor - i\right) \wedge F_{2j}\left(i - \left\lfloor \frac{-s_{1i}}{c_{1i}} \right\rfloor\right) \end{aligned}$$

In the real case the constant k is guaranteed to be either $-\varepsilon$ or 0. Thus, there is at most one interesting value. We use the following definitions.

$$\begin{aligned} s_{3ij} &:= c_{2j}s_{1i} + c_{1i}s_{2j} \\ k_{3ij} &:= \begin{cases} k_{2j} & \text{if } k_{1i} = -\varepsilon \\ 0 & \text{if } k_{1i} = 0 \end{cases} \\ F_{3ij} &:\equiv \begin{cases} F_{2j}\left(\frac{s_{1i}}{c_{1i}}\right) & \text{if } k_{1i} = -\varepsilon \\ s_{3ij} < 0 \vee \left(F_{1i}\left(-\frac{s_{1i}}{c_{1i}}\right) \wedge F_{2j}\left(\frac{s_{1i}}{c_{1i}}\right)\right) & \text{if } k_{1i} = 0 \end{cases} \end{aligned}$$

The partial interpolant of the resolvent $C_1 \vee C_2$ in the mixed case can be expressed as

$$\mathsf{mixcomb}(t \le c, I_1[LA_{1i}], I_2[LA_{2j}]) :\equiv I_1[I_2[LA_{311}]\dots[LA_{31m}]]\dots[I_2[LA_{3n1}]\dots[LA_{3nm}]].$$

The interpolation rules are exactly the same as for the binary case. The only differences between the definition above and the one in [3] is a small simplification of the rule for linear arithmetic that would also be applicable to the binary interpolation case and the exact definition of $EQ(X,s)$, which is only needed for the correctness proof and not used in the interpolation algorithm.

Conjecture 2. *The extended interpolation rules with the definition of* mixcomb *for the equality and inequality literals given above is correct. Thus, if I_1 and I_2 are partial tree interpolants for the clauses $\ell \vee C_1$ and $\neg\ell \vee C_2$ respectively, then I_3 is a partial tree interpolant for the clause $C_1 \vee C_2$.*

6 Conclusion

We presented our ongoing work to extend proof tree preserving interpolation to tree interpolation, a generalisation of sequence interpolation. The key ingredients are the extension of the projection functions to mixed literals by introducing auxiliary variables and predicates, a syntactic restriction of the occurrence of these auxiliary symbols in the partial tree interpolants, and a set of rules to compute partial tree interpolants for resolution steps on mixed literals. The major difficulty with this technique lies in the correctness proofs that are still part of ongoing work. To our knowledge, this is the first paper to focus on the problem of extracting tree interpolants from resolution proofs produced by state-of-the-art SMT solvers. The interpolation technique is implemented in the interpolating SMT solver SMTInterpol [2].

References

[1] iZ3 documentation. `http://research.microsoft.com/en-us/um/redmond/projects/z3/old/iz3documentation.html`. Accessed: 2012-10-05.

[2] Jürgen Christ, Jochen Hoenicke, and Alexander Nutz. SMTInterpol: An interpolating SMT solver. In *SPIN'12*, pages 248–254. Springer, 2012.

[3] Jürgen Christ, Jochen Hoenicke, and Alexander Nutz. Proof tree preserving interpolation. In *TACAS'13*, pages 124–138. Springer, 2013.

[4] Jürgen Christ, Jochen Hoenicke, and Alexander Nutz. Proof tree preserving interpolation. Reports of SFB/TR 14 AVACS 89, SFB/TR 14 AVACS, February 2013. ISSN: 1860-9821, `http://www.avacs.org`.

[5] William Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *J. Symb. Log.*, 22(3):269–285, 1957.

[6] Klaus Dräger, Andrey Kupriyanov, Bernd Finkbeiner, and Heike Wehrheim. SLAB: A certifying model checker for infinite-state concurrent systems. In *TACAS'10*, pages 271–274. Springer, 2010.

[7] Bruno Dutertre and Leonardo de Moura. A fast linear-arithmetic solver for DPLL(T). In *CAV'06*, pages 81–94. Springer, 2006.

[8] Azadeh Farzan, Zachary Kincaid, and Andreas Podelski. Inductive data flow graphs. In *POPL'13*, pages 129–142. ACM, 2013.

[9] Alexander Fuchs, Amit Goel, Jim Grundy, Sava Krstic, and Cesare Tinelli. Ground interpolation for the theory of equality. In *TACAS'09*, pages 413–427. Springer, 2009.

[10] Ashutosh Gupta, Corneliu Popeea, and Andrey Rybalchenko. Solving recursion-free horn clauses over LI+UIF. In *APLAS'11*, pages 188–203. Springer, 2011.

[11] Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. Nested interpolants. In *POPL'10*, pages 471–482. ACM, 2010.

[12] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth L. McMillan. Abstractions from proofs. In *POPL'04*, pages 232–244. Springer, 2004.

[13] Kenneth L. McMillan. Interpolation and SAT-based model checking. In *CAV'03*, pages 1–13. Springer, 2003.

[14] Kenneth L. McMillan. An interpolating theorem prover. *Theor. Comput. Sci.*, 345(1):101–121, 2005.

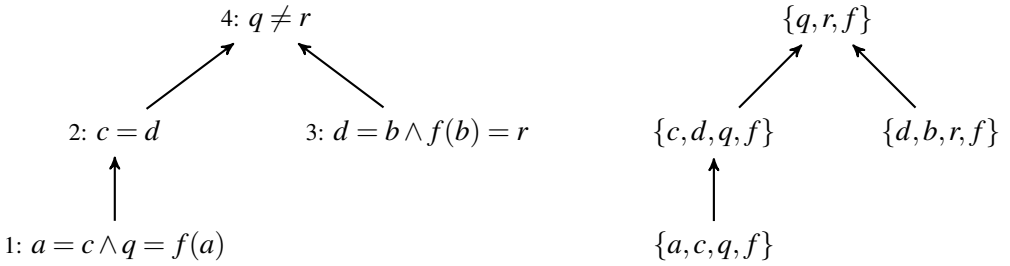
[15] Kenneth L. McMillan. Lazy abstraction with interpolants. In *CAV'06*, pages 123–136. Springer, 2006.

[16] Pavel Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *J. Symb. Log.*, 62(3):981–998, 1997.

[17] Ondrej Sery, Grigory Fedyukovich, and Natasha Sharygina. Incremental upgrade checking by means of interpolation-based function summaries. In *FMCAD'12*, pages 114–121. IEEE, 2012.

A Example

We present our algorithm on the tree interpolation problem from Figure 1(a) consisting of four nodes. In the figure we draw the arrows from children to parent. Each node contains the labelling depicted by the corresponding figure, i. e., either the input labelling, or a (partial) tree interpolant. The numbers in Figure 1(a) will be used to identify the individual nodes. The (extended) symbol set $symb(v)$ for each node is given in Figure 1(b).



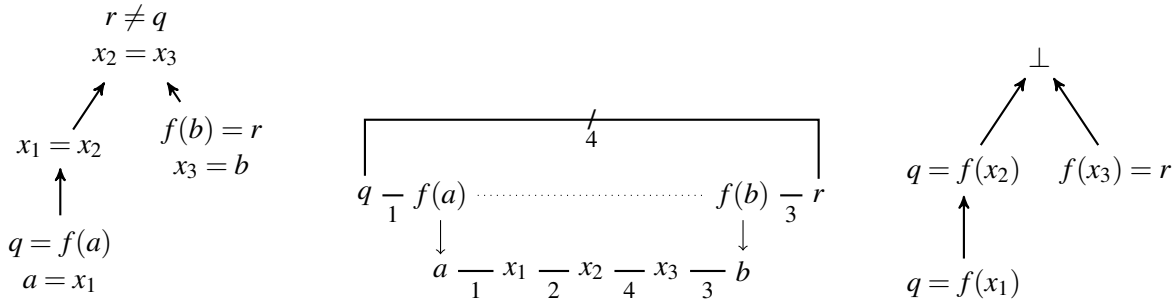
(a) Interpolation Problem (b) Symbol set $symb(v)$ for each node

Figure 1: Tree interpolation problem and symbol set used throughout this section.

A.1 Leaf Interpolation for Equality Theory

We assume the solver for the theory of equalities produces the literal $a = b$ and detects the conflicts $q = f(a) \wedge a = b \wedge f(b) = r \wedge q \neq r$ and $a = c \wedge c = d \wedge d = b \wedge a \neq b$. We show how to derive partial tree interpolants for these conflicts from the corresponding congruence graphs.

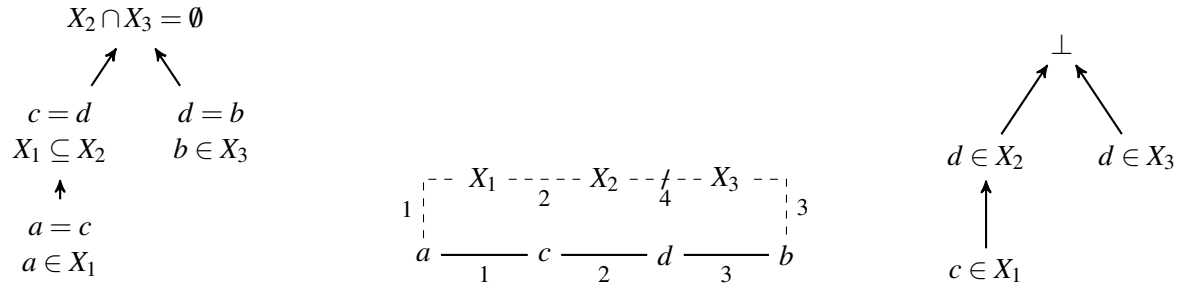
Figure 2(a) gives the projection of the conflict onto the individual nodes. For the literal $a = b$, which is mixed in nodes 1, 2, and 3, we introduce the auxiliary variables x_1, x_2, x_3. In Figure 2(b) we show the corresponding Congruence Closure graph which we already extended by the auxiliary variables. The horizontal edges denote equalities and are labelled by the node that contains the equality literal. The vertical arrows denote function application and the dotted edge is a derived congruence. The inequality is depicted by the top edge. The interpolation algorithm summarises for each node the equalities occurring in the corresponding subtree. If the equality chain crosses a function application, e. g., $q = f(a)$ and $a = x_1$, we need to lift the end-point yielding $q = f(x_1)$. For Node 4, the (dis-)equality chain spans the whole cycle and is summarised by $\bot$. The resulting partial tree interpolant is given in Figure 2(c).



(a) Interpolation Problem (b) Corresponding Congruence Graph (c) Partial Tree Interpolant

Figure 2: Interpolating the conflict $q = f(a) \wedge a = b \wedge f(b) = r \wedge q \neq r$.

The second conflict $a = c \wedge c = d \wedge d = b \wedge a \neq b$ contains the negated literal $a \neq b$, for which we introduce set-valued auxiliary variables X_1, X_2, X_3. Figure 3(a) gives the projection of the conflict onto the individual nodes. The corresponding Congruence Closure graph is given in Figure 3(b). Here, we split the disequality into the literals $a \in X_1, X_1 \subseteq X_2, X_2 \cap X_3 = \emptyset$ and $b \in X_3$. These literals are sketched in the figure by dashed edges. The interpolants are computed as usual by summarising the edges belonging to one subtree. Here, $d = c$, $c = a$, $a \in X_1$ and $X_1 \subseteq X_2$ is summarised by $d \in X_2$. The literal $X_2 \cap X_3 = \emptyset$ occurs only in the node $\mathsf{mixedparent}(a \neq b)$ and the literal is not mixed in that node. Thus, we never need to build a summary including this edge. The resulting partial tree interpolant is given in Figure 3(c).



(a) Interpolation Problem (b) Corresponding Congruence Graph (c) Partial Tree Interpolant

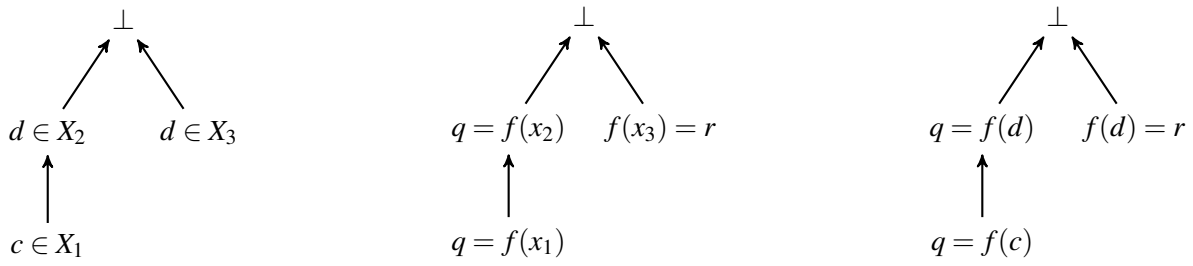
Figure 3: Interpolating the conflict $a = c \wedge c = d \wedge d = b \wedge a \neq b$.

A.2 Interpolation Rule for Resolution Proof

The theory lemma clauses corresponding to the conflicts in the previous section are combined by the resolution rule to a new clause.

$$\frac{a = b \vee a \neq c \vee c \neq d \vee d \neq b \qquad a \neq b \vee q \neq f(a) \vee f(b) \neq r \vee q = r}{a \neq c \vee c \neq d \vee d \neq b \vee q \neq f(a) \vee f(b) \neq r \vee q = r}$$

Since the pivot is mixed in nodes 1, 2, and 3, we need to apply mixcomb to combine the partial interpolants of these nodes. For equality literals the interpolants have the shape $I_1[s \in X]$ and $I_2(x)$ (in our case $I_1[F] \equiv F$). The resulting interpolant for each node is computed as $I_1[I_2(s)]$, which basically means that we just have to replace in the second interpolant x_i by the term s, where $s \in X_i$ is the first interpolant. The result is shown in Figure 4.



(a) Partial Tree Interpolant of $a = b \vee \ldots$ (b) Partial Tree Interpolant of $a \neq b \vee \ldots$ (c) Partial Tree Interpolant of resolvent

Figure 4: Applying the interpolation rule for resolution.

B Proofs, Proof Sketches, and Proof Ideas

B.1 Proof of Lemma 1

Proof. Assume we added a to $symb(w)$ for some $w \in st(v)$. For w there are $v_1, v_2 \in V$ with $a \in symb(L(v_i))$. At least one of them is a descendent of w, hence there is another node in $v_i \in st(v)$ with $a \in symb(L(v_i))$. On the other hand, if we added a to $symb(w')$ for some $w' \notin st(v)$. Then again there are $v_1, v_2 \in V$ with $a \in symb(L(v_i))$. If both of these nodes would lie in $st(v)$, then v would be a common ancestor of v_1 and v_2 which contradicts $w \notin st(v)$. Hence there is a node $v_i \notin st(v)$ with $a \in symb(L(v_i))$. □

B.2 Proof for Theorem 1

Proof. Given a sequence interpolation problem $F_1, \ldots, F_n$, construct a tree as follows. Let $T = (V, E)$ with $V = \{v_0, v_1, \ldots, v_n\}$, $E = \{(v_{i-1}, v_i) \mid 1 \le i \le n\}$, $L(v_i) = F_i$ for $1 \le i \le n$, and $L(v_0) = \top$. Given a solution I to the tree interpolation problem $T = (V, E)$ and L, a solution to the sequence interpolation problem is $I_i := I(v_i)$. □

B.3 Proof Sketch of Lemma 2

Proof Sketch. Fixing a node v_p we have to show $\bigwedge_{(v_c, v_p) \in E} I_3(v_c) \wedge L(v_p) \wedge (\neg C_1 \wedge \neg C_2) \downharpoonright v_p \models I_3(v_p)$ and can assume that a similar condition holds for I_1 and I_2. We distinguish three cases.

Case 1. There is an edge $(v_c, v_p) \in E$ with $\ell \downharpoonright v = \top$ for all $v \notin st(v_c)$. Then $I_3(v_c) = I_1(v_c) \vee I_2(v_c)$ and $I_3(v_p) = I_1(v_p) \vee I_2(v_p)$. Also for all other edges $(v'_c, v_p) \in E$, $\ell \downharpoonright v = \top$ for all $v \in st(v'_c)$, hence $I_3(v'_c) = I_1(v'_c) \wedge I_2(v'_c)$.

Assume $\bigwedge_{(v_c, v_p) \in E} I_3(v_c)$ holds, then $\bigwedge_{(v_c, v_p) \in E} I_1(v_c)$ or $\bigwedge_{(v_c, v_p) \in E} I_2(v_c)$ hold. Using the induction hypothesis for I_1 and I_2 we derive that $I_1(v_p)$ or $I_2(v_p)$ hold (note that $\ell \downharpoonright v_p = \top$ since $v_p \notin st(v_c)$). Then $I_3(v_p)$ holds.

Case 2. Assume $\ell \downharpoonright v = \top$ for all $v \in st(v_p)$. Then $I_3(v_p) = I_1(v_p) \wedge I_2(v_p)$ and $I_3(v_c) = I_1(v_c) \wedge I_2(v_c)$ for all $(v_c, v_p) \in E$. From $\bigwedge I_3(v_c)$ we conclude that $\bigwedge_{(v_c, v_p) \in E} I_1(v_c)$ and $\bigwedge_{(v_c, v_p) \in E} I_2(v_c)$ hold. Using the induction hypothesis (again $\ell \downharpoonright v_p = \top$) we derive $I_1(v_p)$ and $I_2(v_p)$ thus $I_3(v_p)$.

Case 3. Otherwise $I_3(v_c) \implies (I_1(v_c) \vee \ell) \wedge (I_2(v_c) \vee \neg \ell)$ for all $(v_c, v_p) \in E$ since we are not in Case 1. With the induction hypothesis and $\ell \implies \ell \downharpoonright v_p$ we derive from $\bigwedge I_3(v_c)$ that $(I_1(v_p) \vee \ell) \wedge (I_2(v_p) \vee \neg \ell)$ holds. Since we are not in Case 2, this implies $I_3(v_p)$. □

B.4 Proof of Lemma 3

Proof. We first show

$$\mathsf{mixed}(\ell) \subseteq \{v \in V \mid \exists i.\ 1 \le i \le n. lca(a_i) \in st(v) \text{ and } \mathsf{mixedparent}(\ell) \text{ is a proper ancestor of } v\}.$$

Let $v \in \mathsf{mixed}(\ell)$. Since ℓ is mixed in v, there is at least one symbol a_i that occurs only inside the subtree of v. Hence, $lca(a_i) \in st(v)$ for some i. Moreover, the $\mathsf{mixedparent}(\ell)$ is an ancestor of the parent of v, hence it is a proper ancestor of v.

For the other direction

$$\mathsf{mixed}(\ell) \supseteq \{v \in V \mid \exists i.\ 1 \le i \le n. lca(a_i) \in st(v) \text{ and } \mathsf{mixedparent}(\ell) \text{ is a proper ancestor of } v\}$$

take a node v, from the set on the right-hand side. Then there is an i such that $lca(a_i) \in st(v)$, i. e., a_i occurs only inside the subtree of v. It remains to show that there is another symbol that occurs only outside the subtree of v. There must be a node $w \in \mathsf{mixed}(\ell)$ such that v is not a proper ancestor of w (otherwise v would be an ancestor of $\mathsf{mixedparent}(\ell)$).

Case 1: w is an ancestor of v. There is a symbol a_j that only occurs outside of the subtree of w. Thus, this symbol occurs only outside of the subtree of v, so ℓ is mixed in v.

Case 2: w and v have disjoint subtrees. There is a symbol a_j that only occurs inside of w. Thus, this symbol occurs only outside of the subtree of v, so ℓ is mixed in v. □

B.5 Projection Function is Correct

Lemma 4 (Correctness of the Projection Function)**.** *The projection function defined in Section 5.3 is correct (in the sense of Definition 2.*

Proof Sketch. For $\ell \equiv a_1 = a_2$, show by induction on v_p that

$$\exists\{x_j | v_j \in (st(v_p) \setminus \{v_p\}) \cap \mathsf{mixed}(\ell)\}. \bigwedge_{v_j \in st(v_p)} \ell \downarrow v_j$$
$$\iff \begin{cases} \top & \text{if } v_j \notin \mathsf{mixed}(\ell) \text{ for all } v_j \in st(v_p), \\ a_i = x_p & \text{if } v_p \in \mathsf{mixed}(\ell),\ lca(a_i) \in st(v_p), \\ a_1 = a_2 & \text{if } \mathsf{mixedparent}(\ell) \in st(v_p). \end{cases}$$

For $\ell \equiv a_1 \neq a_2$, show by induction on v_p that

$$\exists\{X_j | v_j \in (st(v_p) \setminus \{v_p\}) \cap \mathsf{mixed}(\ell)\}. \bigwedge_{v_j \in st(v_p)} \ell \downarrow v_j$$
$$\iff \begin{cases} \top & \text{if } v_j \notin \mathsf{mixed}(\ell) \text{ for all } v_j \in st(v_p), \\ a_i \in X_p & \text{if } v_p \in \mathsf{mixed}(\ell),\ lca(a_i) \in st(v_p), \\ a_1 \neq a_2 & \text{if } \mathsf{mixedparent}(\ell) \in st(v_p). \end{cases}$$

For $\ell \equiv \sum c_i a_i \leq c$, show by induction on v_p that

$$\exists\{x_j | v_j \in (st(v_p) \setminus \{v_p\}) \cap \mathsf{mixed}(\ell)\}. \bigwedge_{v_j \in st(v_p)} \ell \downarrow v_j$$
$$\iff \begin{cases} \top & \text{if } v_j \notin \mathsf{mixed}(\ell) \text{ for all } v_j \in st(v_p), \\ \sum_{lca(a_i) \in st(v_p)} c_i a_i \leq x_p & \text{if } v_p \in \mathsf{mixed}(\ell), \\ \sum_{lca(a_i) \in st(v_p)} c_i a_i \leq c & \text{if } \mathsf{mixedparent}(\ell) \in st(v_p). \end{cases}$$

□

B.6 Proof Idea for Leaf Interpolation (Conjecture 1)

Proof Idea. For equality conflicts that does not contain a mixed *disequality* the A paths in the interpolant of the parent node are the summary of all literals occurring in the subtree. It is thus the summary of the literals occurring in the parent node and the equalities in the interpolants of the interpolants of the child nodes. Thus it follows from them.

For mixed disequality we also summarise the literals $s = a$, $a \in X_1$, and $X_1 \subseteq X_2$ to $s \in X_2$. When moving to a mixed parent of a mixed child it is not too difficult to see that when the equality chain is extended at the front by $s' = s$ (e. g. in another child node) and the subset chain is extended by $X_2 \subseteq X_3$, that the summary $s' \in X_3$ of the parent node can be derived from these formulae. Finally, when moving to the node $\mathsf{mixedparent}(a_1 \neq a_2)$ we summarise the child interpolants $a_1 \in X_{c_1}$ and $a_2 \in X_{c_2}$ together with $X_{c_1} \cap X_{c_2} = \emptyset$ to $a_1 \neq a_2$.

For inequality conflicts, the interpolant of the parent node is the sum of all inequalities occurring in the subtree (multiplied with their respective Farkas coefficient). This is just the sum of the interpolants of the child nodes and the inequalities occurring in the parent node. □

B.7 Proof Idea for Extended Interpolation Rule (Conjecture 2)

Proof Idea. If the pivot literal is not mixed, the correctness follows from Lemma 2.

For mixed equality literals we need to do a case split over the four cases of the projection function (two children are mixed, one child is mixed, both parent and one child is mixed, and only the parent is mixed). The proof uses the induction hypothesis for I_1 and I_2. In the induction hypothesis for I_1 the variables X_c and X_p may occur. The trick is now to instantiate the variable X_c by the set $\{x | I_2^c(x)\}$ and likewise for X_p. The remaining proof is tedious but straight-forward.

For a mixed inequality literal we need the fact that $LA_{3ij} \iff \exists x. LA_{1i}(x) \wedge LA_{2j}(-x)$ holds, which we proved in [4]. The technical difficulty of the proof lies in finding a common x that works for all terms $LA_{1i}(x)$ and $LA_{2j}(-x)$ that occur in the interpolants I_1 and I_2. This can be achieved by choosing the minimum x for i and the maximum x for j using the monotonicity of LA. Then one can show that if for a mixed child $I_1[I_2[LA_{3ij}]]$ holds, there is an x such that $I_1[LA_{1i}(x)]$ and $I_2[LA_{2j}(-x)]$ holds (except for one special case where $I_2 \equiv \bot$ that needs to be handled separately). Likewise for a mixed parent $I_1[LA_{1i}(x)]$ and $I_2[LA_{2j}(-x)]$ imply $I_1[I_2[LA_{3ij}]]$. Now the remaining proof is straight-forward using the induction hypothesis. □