Proof Tree Preserving Interpolation ⋆

Jürgen Christ, Jochen Hoenicke, and Alexander Nutz

Chair of Software Engineering, University of Freiburg

Abstract. Craig interpolation in SMT is difficult because, e. g., theory combination and integer cuts introduce mixed literals, i. e., literals containing local symbols from both input formulae. In this paper, we present a scheme to compute Craig interpolants in the presence of mixed literals. Contrary to existing approaches, this scheme neither limits the inferences done by the SMT solver, nor does it transform the proof tree before extracting interpolants. Our scheme works for the combination of uninterpreted functions and linear arithmetic but is extendable to other theories. The scheme is implemented in the interpolating SMT solver SMTInterpol.

1 Introduction

A Craig interpolant for a pair of formulae A and B whose conjunction is unsatisfiable is a formula I that follows from A and whose conjunction with B is unsatisfiable. Furthermore, I only contains symbols common to A and B. Model checking and state space abstraction [16,21] make intensive use of interpolation to achieve a higher degree of automation. This increase in automation stems from the ability to fully automatically generate interpolants from proofs produced by modern theorem provers.

For propositional logic, a SAT solver typically produces resolution-based proofs that show the unsatisfiability of an error path. Extracting Craig interpolants from such proofs is a well understood and easy task that can be accomplished, e. g., using the algorithms of Pudlák [25] or McMillan [20]. An essential property of the proofs generated by SAT solvers is that every proof step only involves literals that occur in the input.

This property does not hold for proofs produced by SMT solvers for formulae in a combination of first order theories. Such solvers produce new literals for different reasons. First, to combine two theory solvers, SMT solvers exchange (dis-)equalities between the symbols common to these two theories in a Nelson-Oppen-style theory combination. Second, various techniques dynamically generate new literals to simplify proof generation. Third, new literals are introduced in the context of a branch-and-bound or branch-and-cut search for non-convex theories. The theory of linear integer arithmetic for example is typically solved by

⋆ This work is supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center “Automatic Verification and Analysis of Complex Systems” (SFB/TR14 AVACS)

searching a model for the relaxation of the formula to linear rational arithmetic and then using branch-and-cut with Gomory cuts or *extended branches* [8] to remove the current non-integer solution from the solution space of the relaxation.

The literals produced by either of these techniques only contain symbols that are already present in the input. However, a literal produced by one of these techniques may be *mixed*[1] in the sense that it may contain symbols occurring only in A and symbols occurring only in B. These literals pose the major difficulty when extracting interpolants from proofs produced by SMT solvers.

In this paper, we present a scheme to compute Craig interpolants in the presence of mixed literals. Our interpolation scheme is based on syntactical restrictions of *partial interpolants* and specialised rules to interpolate resolution steps on mixed literals. This enables us to compute interpolants in the context of a state-of-the-art SMT solver without manipulating the proof tree or restricting the solver in any way. We base our presentation on the quantifier-free fragment of the combined theory of uninterpreted functions and linear arithmetic over the rationals or the integers. The interpolation scheme is used in the interpolating SMT solver SMTInterpol [4].

Related Work. Craig [6] shows in his seminal work on interpolation that for every inconsistent pair of first order formulae an interpolant can be derived. In the proof of the corresponding theorem he shows how to construct interpolants without proofs by introducing quantifiers in the interpolant. For Boolean circuits, Pudlák [25] shows how to construct quantifier-free interpolants from resolution proofs of unsatisfiability.

A different proof-based interpolation system is given by McMillan [20] in his seminal paper on interpolation for SMT. The presented method combines the theory of equality and uninterpreted functions with the theory of linear rational arithmetic. Interpolants are computed from partial interpolants by annotating every proof step. The partial interpolants have a specific form that carries information needed to combine the theories. The proof system is incomplete for linear integer arithmetic as it cannot deal with arbitrary cuts and mixed literals introduced by these cuts.

Brillout et al. [2] present an interpolating sequent calculus that can compute interpolants for the combination of uninterpreted functions and linear integer arithmetic. The interpolants computed using their method might contain quantifiers since they do not use divisibility predicates. Furthermore their method limits the generation of Gomory cuts in the integer solver to prevent some mixed cuts. The method presented in this paper combines the two theories without quantifiers and, furthermore, does not restrict any component of the solver.

Yorsh and Musuvathi [26] show how to combine interpolants generated by an SMT solver based on Nelson-Oppen combination. They define the concept of *equality-interpolating theories*. These are theories that can provide a shared term t for a mixed literal $a = b$ that is derivable from an interpolation problem. A troublesome mixed interface equality $a = b$ is rewritten into the conjunction

[1] Mixed literals sometimes are called *uncolourable*.

$a = t \wedge t = b$. They show that both, the theory of uninterpreted functions and the theory of linear rational arithmetic are equality-interpolating. We do not explicitly split the proof. Additionally, our method can handle the theory of linear integer arithmetic without any restriction on the solver. The method of Yorsh and Musuvathi, however, cannot deal with cuts used by most modern SMT solvers to decide linear integer arithmetic.

Cimatti et al. [5] present a method to compute interpolants for linear rational arithmetic and difference logic. The method presented in this paper builds upon their interpolation technique for linear rational arithmetic. For theories combined via delayed theory combination, they show how to compute interpolants by transforming a proof into a so-called *ie-local* proof. In these proofs, mixed equalities are close to the leaves of the proof tree and splitting them is cheap since the proof trees that have to be duplicated are small. A variant of this restricted search strategy is used by MathSAT [13] and CSIsat [1].

Goel et al. [12] present a generalisation of equality-interpolating theories. They define the class of *almost-colourable proofs* and an algorithm to generate interpolants from such proofs. Furthermore they describe a restricted DPLL system to generate almost-colourable proofs. This system does not restrict the search if convex theories are used. Their procedure is incomplete for non-convex theories like linear arithmetic over integers since it prohibits the generation of mixed branches and cuts.

Recently, techniques to transform proofs gained a lot of attention. Bruttomesso et al. [3] present a framework to lift resolution steps on mixed literals into the leaves of the resolution tree. Once a subproof only resolves on mixed literals, they replace this subproof with the conclusion removing the mixed inferences. The newly generated lemmas however are mixed between different theories and require special interpolation procedures. Even though these procedures only have to deal with conjunctions of literals in the combined theories it is not obvious how to compute interpolants in this setting. Similar to our algorithm, they do not restrict or interact with the SMT solver but take the proof as produced by the solver. In contrast to our approach, they manipulate the proof in a way that is worst-case exponential and rely on an interpolant generator for the conjunctive fragment of the combined theories.

McMillan [22] presents a technique to compute interpolants from Z3 proofs. Whenever a sub-proof contains mixed literals, he extracts lemmas from the proof tree and delegates them to a second (possibly slower) interpolating solver.

For the theory of linear integer arithmetic $\mathcal{LA}(\mathbb{Z})$ a lot of different techniques were proposed. Lynch et al. [19] present a method that produces interpolants as long as no mixed cuts were introduced. In the presence of such cuts, their interpolants might contain symbols that violate the symbol condition of Craig interpolants.

For linear Diophantine equations and linear modular equations, Jain et al. [17] present a method to compute linear modular equations as interpolants. Their method however is limited to equations and, thus, not suitable for the whole theory $\mathcal{LA}(\mathbb{Z})$.

Griggio [14] shows how to compute interpolants for $\mathcal{LA}(\mathbb{Z})$ based on the $\mathcal{LA}(\mathbb{Z})$-solver from MathSAT [13]. This solver uses branch-and-bound and the cuts from proofs [8] technique. Similar to the technique presented by Kroening et al. [18] the algorithm prevents generating mixed cuts and, hence, restricts the inferences done by the solver.

2 Preliminaries

In this section, we give an overview of what is needed to understand the procedure we will propose in the later sections. We will briefly introduce the logic and the theories used in this paper. Furthermore, we define key terms like Craig interpolants and symbol sets.

Logic, Theories, and SMT. We assume standard first-order logic. We operate within the quantifier-free fragments of the theory of equality with uninterpreted functions $\mathcal{EUF}$ and the theories of linear arithmetic over rationals $\mathcal{LA}(\mathbb{Q})$ and integers $\mathcal{LA}(\mathbb{Z})$. The quantifier-free fragment of $\mathcal{LA}(\mathbb{Z})$ is not closed under interpolation. Therefore, we augment the signature with division by constant functions $\left\lfloor \frac{\cdot}{k} \right\rfloor$ for all integers $k \geq 1$.

We use the standard notations $\models_T, \perp, \top$ to denote entailment in the theory T, contradiction, and tautology. In the following, we drop the subscript T as it always corresponds to the combined theory of $\mathcal{EUF}$, $\mathcal{LA}(\mathbb{Q})$, and $\mathcal{LA}(\mathbb{Z})$.

The literals in $\mathcal{LA}(\mathbb{Z})$ are of the form $s \leq c$, where c is an integer constant and s a linear combination of variables. For $\mathcal{LA}(\mathbb{Q})$ we use constants $c \in \mathbb{Q}_\varepsilon$, $\mathbb{Q}_\varepsilon := \mathbb{Q} \cup \{q - \varepsilon | q \in \mathbb{Q}\}$ where the meaning of $s \leq q - \varepsilon$ is $s < q$. For better readability we use, e. g., $x \leq y$ resp. $x > y$ to denote $x - y \leq 0$ resp. $y - x \leq -\varepsilon$. In the integer case we use $x > y$ to denote $y - x \leq -1$.

Our algorithm operates on a proof of unsatisfiability generated by an SMT solver based on DPLL(T) [24]. Such a proof is a resolution tree with the $\perp$-clause at its root. The leaves of the tree are either clauses from the input formulae[2] or theory lemmas that are produced by one of the theory solvers. The negation of a theory lemma is called a *conflict*.

The theory solvers for $\mathcal{EUF}$, $\mathcal{LA}(\mathbb{Q})$, and $\mathcal{LA}(\mathbb{Z})$ are working independently and exchange (dis-)equality literals through the DPLL engine in a Nelson-Oppen style [23]. Internally, the solver for linear arithmetic uses only inequalities in theory conflicts. In the proof tree, the (dis-)equalities are related to inequalities by the (valid) clauses $x = y \vee x < y \vee x > y$, and $x \neq y \vee x \leq y$. We call these leaves of the proof tree *theory combination clauses*.

Interpolants and Symbol Sets. For a formula F, we use $symb(F)$ to denote the set of non-theory symbols occurring in F. An interpolation problem is given by two formulae A and B such that $A \wedge B \models \perp$. An interpolant of A and B is a formula I such that (i) $A \models I$, (ii) $B \wedge I \models \perp$, and (iii) $symb(I) \subseteq symb(A) \cap symb(B)$.

[2] W. l. o. g. we assume input formulae are in conjunctive normal form.

We call a symbol $s \in \mathit{symb}(A) \cup \mathit{symb}(B)$ *shared* if $s \in \mathit{symb}(A) \cap \mathit{symb}(B)$, *A-local* if $s \in \mathit{symb}(A) \setminus \mathit{symb}(B)$, and *B-local* if $s \in \mathit{symb}(B) \setminus \mathit{symb}(A)$. Similarly, we call a term *A-local* (*B-local*) if it contains at least one A-local (B-local) and no B-local (A-local) symbols. We call a term *(AB-)shared* if it contains only shared symbols and *(AB-)mixed* if it contains A-local as well as B-local symbols. The same terminology applies to formulae.

Substitution in Formulae and Monotonicity. By $F[G_1] \dots [G_n]$ we denote a formula in negation normal form with sub-formulae $G_1, \dots, G_n$ that occur positively in the formula. Substituting these sub-formulae by formula $G_1', \dots, G_n'$ is denoted by $F[G_1'] \dots [G_n']$. By $F(t)$ we denote a formula with a sub-term t that can appear anywhere in F. The substitution of t with a term t' is denoted by $F(t')$.

The following lemma is important for the correctness proofs in the remainder of this technical report. It also represents a concept that is important for the understanding of the proposed procedure.

Lemma 1 (Monotonicity). *Given a formula $F[G_1] \dots [G_n]$ in negation normal form with sub-formulae $G_1, \dots, G_n$ occurring only positively in the formula and formulae $G_1', \dots, G_n'$, it holds that*

$$\left(\bigwedge_{i \in \{1,\dots,n\}} (G_i \rightarrow G_i') \right) \rightarrow (F[G_1] \dots [G_n] \rightarrow F[G_1'] \dots [G_n'])$$

Proof. We prove the claim by induction over the number of $\wedge$ and $\vee$ connectives in $F[\cdot] \dots [\cdot]$. If $F[G_1] \dots [G_n]$ is a literal different from $G_1, \dots, G_n$ the implication holds trivially. Also for the other base case $F[G_1] \dots [G_n] \equiv G_i$ for some $i \in \{1, \dots, n\}$ the property holds. For the induction step observe that if $F_1[G_1] \dots [G_n] \rightarrow F_1[G_1'] \dots [G_n']$ and $F_2[G_1] \dots [G_n] \rightarrow F_2[G_1'] \dots [G_n']$, then

$$F_1[G_1] \dots [G_n] \wedge F_2[G_1] \dots [G_n] \rightarrow F_1[G_1'] \dots [G_n'] \wedge F_2[G_1'] \dots [G_n'] \text{ and}$$
$$F_1[G_1] \dots [G_n] \vee F_2[G_1] \dots [G_n] \rightarrow F_1[G_1'] \dots [G_n'] \vee F_2[G_1'] \dots [G_n']. \qquad \square$$

3 Proof Tree-Based Interpolation

Interpolants can be computed from proofs of unsatisfiability as Pudlák and McMillan have already shown. In this section we will introduce their algorithms. Then, we will discuss the changes necessary to handle mixed literals introduced, e. g., by theory combination.

3.1 Pudlák's and McMillan's Interpolation Algorithms

Pudlák's and McMillan's algorithms assume that the pivot literals are not mixed. We will remove this restriction later. We define a common framework that is more general and can be instantiated to obtain Pudlák's or McMillan's algorithm to compute interpolants. For this, we use two projection functions on literals $\cdot \downharpoonleft A$ and $\cdot \downharpoonleft B$ as defined below. They have the properties (i) $\mathit{symb}(\ell \downharpoonleft A) \subseteq \mathit{symb}(A)$, (ii) $\mathit{symb}(\ell \downharpoonleft B) \subseteq \mathit{symb}(B)$, and (iii) $\ell \iff (\ell \downharpoonleft A \wedge \ell \downharpoonleft B)$. Other projection functions are possible and this allows for varying the strength of the resulting interpolant as shown in [9]. We extend the projection function to conjunctions of literals component-wise.

	Pudlák		McMillan	
	$\ell \downharpoonleft A$	$\ell \downharpoonleft B$	$\ell \downharpoonleft A$	$\ell \downharpoonleft B$
ℓ is A-local	ℓ	$\top$	ℓ	$\top$
ℓ is B-local	$\top$	ℓ	$\top$	ℓ
ℓ is shared	ℓ	ℓ	$\top$	ℓ

Given an interpolation problem A and B, a *partial interpolant* of a clause C is an interpolant of the formulae $A \wedge (\neg C \downharpoonleft A)$ and $B \wedge (\neg C \downharpoonleft B)$[3]. Partial interpolants can be computed inductively over the structure of the proof tree. A partial interpolant of a theory lemma C can be computed by a theory-specific interpolation routine as an interpolant of $\neg C \downharpoonleft A$ and $\neg C \downharpoonleft B$. Note that the conjunction is equivalent to $\neg C$ and therefore unsatisfiable. For an input clause C from the formula A (resp. B), a partial interpolant is $\neg(\neg C \setminus A)$ (resp. $\neg C \setminus B$) where $\neg C \setminus A$ is the conjunction of all literals of $\neg C$ that are not in $\neg C \downharpoonleft A$ and analogously for $\neg C \setminus B$. For a resolution step, a partial interpolant can be computed using (rule-res), which is given below. For this rule, it is easy to show that I_3 is a partial interpolant of $C_1 \vee C_2$ given that I_1 and I_2 are partial interpolants of $C_1 \vee \ell$ and $C_2 \vee \neg\ell$, respectively. Note that the "otherwise" case never triggers in McMillan's algorithm.

$$\frac{C_1 \vee \ell : I_1 \quad C_2 \vee \neg\ell : I_2}{C_1 \vee C_2 : I_3} \quad \text{where } I_3 = \begin{cases} I_1 \vee I_2 & \text{if } \ell \downharpoonleft B = \top \\ I_1 \wedge I_2 & \text{if } \ell \downharpoonleft A = \top \\ (I_1 \vee \ell) \wedge (I_2 \vee \neg\ell) & \text{otherwise} \end{cases} \qquad \text{(rule-res)}$$

As the partial interpolant of the root of the proof tree (which is labelled with the clause $\perp$) is an interpolant of the input formulae A and B, this algorithm can be used to compute interpolants.

Theorem 1. *The above-given partial interpolants are correct, i.e., if I_1 is a partial interpolant of $C_1 \vee \ell$ and I_2 is a partial interpolant of $C_2 \vee \neg\ell$ then I_3 is a partial interpolant of the clause $C_1 \vee C_2$.*

Proof. The third property, i.e., $\mathit{symb}(I_3) \subseteq \mathit{symb}(A) \cap \mathit{symb}(B)$, clearly holds if we assume it holds for I_1 and I_2. Note that in the "otherwise" case, ℓ is shared. We prove the other two partial interpolant properties separately.

[3] Note that $\neg C$ is a conjunction of literals. Thus, $\neg C \downharpoonleft A$ is well defined.

Inductivity. We have to show

$$A \wedge \neg C_1 \downharpoonleft A \wedge \neg C_2 \downharpoonleft A \models I_3.$$

For this we use the inductivity of I_1 and I_2:

$$A \wedge \neg C_1 \downharpoonleft A \wedge \neg \ell \downharpoonleft A \models I_1 \qquad \text{(ind1)}$$
$$A \wedge \neg C_2 \downharpoonleft A \wedge \ell \downharpoonleft A \models I_2 \qquad \text{(ind2)}$$

Assume A, $\neg C_1 \downharpoonleft A$, and $\neg C_2 \downharpoonleft A$. Then, (ind1) simplifies to $\neg \ell \downharpoonleft A \rightarrow I_1$ and (ind2) simplifies to $\ell \downharpoonleft A \rightarrow I_2$. We show that I_3 holds under these assumptions.

Case $\ell \downharpoonleft B = \top$. Then by the definition of the projection function, $\ell \downharpoonleft A = \ell$ and $\neg \ell \downharpoonleft A = \neg \ell$ hold. If ℓ holds, (ind2) gives us I_2, otherwise (ind1) gives us I_1, thus $I_3 = I_1 \vee I_2$ holds in both cases.

Case $\ell \downharpoonleft A = \top$. Then (ind1) gives us I_1 because $\neg \ell \downharpoonleft A = \top$ (the negation of ℓ is still not in A), and (ind2) gives us I_2. So $I_3 = I_1 \wedge I_2$ holds.

Case "otherwise". By the definition of the projection function $\ell \downharpoonleft A = \ell \downharpoonleft B = \ell$ and $\neg \ell \downharpoonleft A = \neg \ell \downharpoonleft B = \neg \ell$. If ℓ holds, the left conjunct $(I_1 \vee \ell)$ of I_3 holds and the right conjunct $(I_2 \vee \neg \ell)$ of I_3 is fulfilled because (ind2) gives us I_2. If $\neg \ell$ holds, (ind1) gives us I_1 and both conjuncts of I_3 hold.

Contradiction. We have to show:

$$B \wedge \neg C_1 \downharpoonleft B \wedge \neg C_2 \downharpoonleft B \wedge I_3 \models \bot$$

We use the contradiction properties of I_1 and I_2:

$$B \wedge \neg C_1 \downharpoonleft B \wedge \neg \ell \downharpoonleft B \wedge I_1 \models \bot \qquad \text{(cont1)}$$
$$B \wedge \neg C_2 \downharpoonleft B \wedge \ell \downharpoonleft B \wedge I_2 \models \bot \qquad \text{(cont2)}$$

If we assume B, $\neg C_1 \downharpoonleft B$, and $\neg C_2 \downharpoonleft B$, (cont1) simplifies to $\neg \ell \downharpoonleft B \wedge I_1 \rightarrow \bot$ and (cont2) simplifies to $\ell \downharpoonleft B \wedge I_2 \rightarrow \bot$. We show $I_3 \rightarrow \bot$.

Case $\ell \downharpoonleft B = \top$. Then (cont1) and $\neg \ell \downharpoonleft B = \top$ give us $I_1 \rightarrow \bot$, and (cont2) and $\ell \downharpoonleft B = \top$ give us $I_2 \rightarrow \bot$. Thus $I_3 \equiv I_1 \vee I_2$ is contradictory.

Case $\ell \downharpoonleft A = \top$. Then $\ell \downharpoonleft B = \ell$ and $\neg \ell \downharpoonleft B = \neg \ell$. Then, if ℓ holds, (cont2) gives us $I_2 \rightarrow \bot$. If $\neg \ell$ holds, (cont1) gives us $I_1 \rightarrow \bot$ analogously. In both cases, $I_3 \equiv I_1 \wedge I_2$ is contradictory.

Case "otherwise". By the definition of the projection function $\ell \downharpoonleft A = \ell \downharpoonleft B = \ell$ and $\neg \ell \downharpoonleft A = \neg \ell \downharpoonleft B = \neg \ell$ hold. Assuming $I_3 \equiv (I_1 \vee \ell) \wedge (I_2 \vee \neg \ell)$ holds, we prove a contradiction. If ℓ holds, the second conjunct of I_3 implies I_2. Then, (cont2) gives us a contradiction. If $\neg \ell$ holds, the first conjunct of I_3 implies I_1 and (cont1) gives us a contradiction. $\square$

3.2 Purification of Mixed Literals

The proofs generated by state-of-the-art SMT solvers may contain mixed literals. We tackle them by extending the projection functions to these literals. The problem here is that there is no projection function that satisfies the conditions stated in the previous section. Therefore, we relax the conditions by allowing fresh auxiliary variables to occur in the projections.

We consider two different kinds of mixed literals: First, (dis-)equalities of the form $a = b$ or $a \neq b$ for an A-local variable a and a B-local variable b are introduced, e. g., by theory combination or Ackermannization. Second, inequalities of the form $a + b \leq c$ are introduced, e. g., by extended branches [8] or bound propagation. Here, a is a linear combination of A-local variables, b is a linear combination of B-local and shared variables, and c is a constant. Adding the shared variable to the B-part is an arbitrary choice. One gets interpolants of different strengths by assigning some shared variables to the A-part. It is only important to keep the projection of each literal consistent throughout the proof.

We split mixed literals using auxiliary variables, which we denote by x or p_x in the following. The variable p_x has the type Boolean, while x has the same type as the variables in the literal. One or two fresh variables are introduced for each mixed literal. We count these variables as shared between A and B. The purpose of the auxiliary variable x is to capture the shared value that needs to be propagated between A and B. When splitting a literal ℓ into A- and B-part, we require that $\ell \Leftrightarrow \exists x, p_x.(\ell \downharpoonleft A) \wedge (\ell \downharpoonleft B)$. We need the additional Boolean variable p_x to split the literal $a \neq b$ into two (nearly) symmetric parts. This is achieved by the definitions below.

$$
\begin{aligned}
(a = b) \downharpoonleft A &:= (a = x) & (a = b) \downharpoonleft B &:= (x = b)\\
(a \neq b) \downharpoonleft A &:= (p_x \text{ xor } a = x) & (a \neq b) \downharpoonleft B &:= (\neg p_x \text{ xor } x = b)\\
(a + b \leq c) \downharpoonleft A &:= (a + x \leq 0) & (a + b \leq c) \downharpoonleft B &:= (-x + b \leq c)
\end{aligned}
$$

Since the mixed variables are considered to be shared, we allow them to occur in the partial interpolant of a clause C. However, a variable may only occur if C contains the corresponding literal. This is achieved by a special interpolation rule for resolution steps where the pivot literal is mixed. The rules for the different mixed literals are the core of our proposed algorithm and will be introduced in the following sections.

Lemma 2 (Partial Interpolation). *Given a mixed literal ℓ with auxiliary variable(s) $\boldsymbol{x}$ and clauses $C_1 \vee \ell$ and $C_2 \vee \neg\ell$ with corresponding partial interpolants I_1 and I_2. Let $C_3 = C_1 \vee C_2$ be the result of a resolution step on $C_1 \vee \ell$ and $C_2 \vee \neg\ell$ with pivot ℓ. If a partial interpolant I_3 satisfies the symbol condition, and*

$$(\forall \boldsymbol{x}.\, (\neg\ell \downharpoonleft A \rightarrow I_1) \wedge (\ell \downharpoonleft A \rightarrow I_2)) \rightarrow I_3 \tag{ind}$$

$$I_3 \rightarrow (\exists \boldsymbol{x}.\, (\neg\ell \downharpoonleft B \wedge I_1) \vee (\ell \downharpoonleft B \wedge I_2)) \tag{cont}$$

then I_3 is a partial interpolant of C_3.

Proof. We need to show inductivity and contradiction for the partial interpolants.

Inductivity. For this we use inductivity of I_1 and I_2:

$$A \wedge \neg C_1 \downharpoonleft A \wedge \neg \ell \downharpoonleft A \models I_1$$
$$A \wedge \neg C_2 \downharpoonleft A \wedge \ell \downharpoonleft A \models I_2$$

Since $\boldsymbol{x}$ does not appear in $C_1 \downharpoonleft A$, $C_2 \downharpoonleft A$ nor A, we can conclude

$$A \wedge \neg C_1 \downharpoonleft A \models \forall \boldsymbol{x}.\, \neg \ell \downharpoonleft A \rightarrow I_1$$
$$A \wedge \neg C_2 \downharpoonleft A \models \forall \boldsymbol{x}.\, \ell \downharpoonleft A \rightarrow I_2$$

Combining these and pulling the quantifier over the conjunction gives

$$A \wedge \neg C_1 \downharpoonleft A \wedge \neg C_2 \downharpoonleft A \models \forall \boldsymbol{x}.\, (\neg \ell \downharpoonleft A \rightarrow I_1) \wedge (\ell \downharpoonleft A \rightarrow I_2)$$

Using (ind), this shows that inductivity for I_3 holds:

$$A \wedge \neg C_1 \downharpoonleft A \wedge \neg C_2 \downharpoonleft A \models I_3.$$

Contradiction. First, we show the contradiction property for I_3:

$$B \wedge \neg C_1 \downharpoonleft B \wedge \neg C_2 \downharpoonleft B \wedge I_3 \models \bot.$$

Assume the formulae on the left-hand side hold. From (cond) we can conclude that there is some $\boldsymbol{x}$ such that

$$(\neg \ell \downharpoonleft B \wedge I_1) \vee (\ell \downharpoonleft B \wedge I_2)$$

If the first disjunct is true we can derive the contradiction using the contradiction property of I_1:

$$B \wedge \neg C_1 \downharpoonleft B \wedge \neg \ell \downharpoonleft B \wedge I_1 \models \bot$$

Otherwise, the second disjunct holds and we can use the contradiction property of I_2

$$B \wedge \neg C_2 \downharpoonleft B \wedge \ell \downharpoonleft B \wedge I_2 \models \bot$$

This shows the contradiction property for I_3. □

It is important to state here that the given purification of a literal into two new literals is not a modification of the proof tree or any of its nodes. The proof tree would no longer be well-formed if we replaced a mixed literal by the disjunction or conjunction of the purified parts. The purification is only used to define partial interpolants of clauses. In fact, it is only used in the correctness proof of our method and is not even done explicitly in the implementation.

3.3 Lemma Used in the Correctness Proof

The following lemma will help us prove the correctness of our proposed new interpolation rules.

Lemma 3 (Deep Substitution). *Let $F_1[G_{11}]\dots[G_{1n}]$ and $F_2[G_{21}]\dots[G_{2m}]$ be two formulae with sub-formulae G_{1i} for $1 \le i \le n$ and G_{2j} for $1 \le j \le m$ occurring positively in F_1 and F_2.*

If $\bigwedge_{i\in\{1,\dots,n\}} \bigwedge_{j\in\{1,\dots,m\}} G_{1i} \wedge G_{2j} \rightarrow G_{3ij}$ holds, then

$$F_1[G_{11}]\dots[G_{1n}] \wedge F_2[G_{21}]\dots[G_{2m}] \rightarrow$$
$$F_1[F_2[G_{311}]\dots[G_{31m}]]\dots[F_2[G_{3n1}]\dots[G_{3nm}]].$$

Proof.

$$\begin{aligned}
& \bigwedge_{i\in\{1,\dots,n\}} \bigwedge_{j\in\{1,\dots,m\}} ((G_{1i} \wedge G_{2j}) \rightarrow G_{3ij}) \\
\Leftrightarrow\; & \bigwedge_{i\in\{1,\dots,n\}} \bigwedge_{j\in\{1,\dots,m\}} (G_{1i} \rightarrow (G_{2j} \rightarrow G_{3ij})) \\
\Leftrightarrow\; & \bigwedge_{i\in\{1,\dots,n\}} (G_{1i} \rightarrow \bigwedge_{j\in\{1,\dots,m\}} (G_{2j} \rightarrow G_{3ij})) \\
\{\text{monotonicity}\} \Rightarrow\; & \bigwedge_{i\in\{1,\dots,n\}} (G_{1i} \rightarrow (F_2[G_{21}]\dots[G_{2m}] \rightarrow F_2[G_{3i1}]\dots[G_{3im}])) \\
\Leftrightarrow\; & \bigwedge_{i\in\{1,\dots,n\}} (F_2[G_{21}]\dots[G_{2m}] \rightarrow (G_{1i} \rightarrow F_2[G_{3i1}]\dots[G_{3im}])) \\
\Leftrightarrow\; & (F_2[G_{21}]\dots[G_{2m}] \rightarrow \bigwedge_{i\in\{1,\dots,n\}} (G_{1i} \rightarrow F_2[G_{3i1}]\dots[G_{3im}])) \\
\{\text{monotonicity}\} \Rightarrow\; & (F_2[G_{21}]\dots[G_{2m}] \rightarrow (F_1[G_{11}]\dots[G_{1n}] \rightarrow \\
& \quad F_1[F_2[G_{311}]\dots[G_{31m}]]\dots[F_2[G_{3n1}]\dots[G_{3nm}]])) \\
\Leftrightarrow\; & (F_1[G_{11}]\dots[G_{1n}] \wedge F_2[G_{21}]\dots[G_{2m}]) \rightarrow \\
& \quad F_1[F_2[G_{311}]\dots[G_{31m}]]\dots[F_2[G_{3n1}]\dots[G_{3nm}]]))
\end{aligned}$$

□

4 Uninterpreted Functions

In this section we will present the part of our algorithm that is specific to the theory $\mathcal{EUF}$. The only mixed atom that is considered by this theory is $a = b$ where a is A-local and b is B-local.

4.1 Leaf Interpolation

The $\mathcal{EUF}$ solver is based on the congruence closure algorithm [7]. The theory lemmas are generated from conflicts involving a single disequality that is in contradiction to a path of equalities. Thus, the clause generated from such a conflict consists of a single equality literal and several disequality literals.

When computing the partial interpolants of the theory lemmas, we internally split the mixed literals according to Section 3.2. Then we use an algorithm similar to [11] to compute an interpolant. This algorithm basically summarises the A-equalities that are adjacent on the path of equalities.

If the theory lemma contains a mixed equality $a = b$ (without negation), it corresponds to the single disequality in the conflict. This disequality is split into p_x xor $a = x$ and $\neg p_x$ xor $x = b$ and the resulting interpolant depends on the value of p_x. If $p_x = \bot$, the disequality is part of the B-part and x is the end of an equality path summing up the equalities from A. Thus, the computed interpolant contains a literal of the form $x = s$. If $p_x = \top$, then the A-part of the literal is $a \neq x$, and the resulting interpolant contains the literal $x \neq s$ instead. Thus, the resulting interpolant can be put into the form $I[p_x \text{ xor } x = s]$. Note that the formula p_x xor $x = s$ occurs positively in the interpolant and is the only part of the interpolant containing x and p_x. We define

$$EQ(x, s) := (p_x \text{ xor } x = s)$$

and require that the partial interpolant of a clause containing the literal $a = b$ always has the form $I[EQ(x, s)]$ where x and p_x do not occur anywhere else.

For theory lemmas containing the literal $a \neq b$, the corresponding auxiliary variable x may appear anywhere in the partial interpolant, even under a function symbol. A simple example is the theory conflict $s \neq f(a) \wedge a = (x =)b \wedge f(b) = s$, which has the partial interpolant $s \neq f(x)$. In general the partial interpolant of such a clause has the form $I(x)$.

When two partial interpolants for clauses containing $a = b$ are combined using (rule-res), i. e., the pivot literal is a non-mixed literal but the mixed literal $a = b$ occurs in C_1 and C_2, the resulting partial interpolant may contain $EQ(x, s_1)$ and $EQ(x, s_2)$ for different shared terms s_1, s_2. In general, we allow the partial interpolants to have the form $I[EQ(x, s_1)] \dots [EQ(x, s_n)]$.

4.2 Pivoting of Mixed Equalities

We require that every clause C containing $a = b$ with auxiliary variables x, p_x is always labelled with a formula of the form $I[EQ(x, s_1)] \dots [EQ(x, s_n)]$. As discussed above, the partial interpolants computed for conflicts in the congruence closure algorithm are of the form $I[EQ(x, s_1)] \dots [EQ(x, s_n)]$. This property is also preserved by (rule-res), and by Theorem 1 this rule also preserves the property of being a partial interpolant. On the other hand, a clause containing the literal $a \neq b$ is labelled with a formula of the form $I(x)$, i. e., the auxiliary variable x can occur at arbitrary positions. Again, the form $I(x)$ and the property of being a partial interpolant is also preserved by (rule-res).

We use the following rule to interpolate the resolution step on the mixed literal $a = b$.

$$\frac{C_1 \vee a = b : I_1[EQ(x, s_1)] \dots [EQ(x, s_n)] \qquad C_2 \vee a \neq b : I_2(x)}{C_1 \vee C_2 : I_1[I_2(s_1)] \dots [I_2(s_n)]} \quad \text{(rule-eq)}$$

The rule replaces every literal $EQ(x, s_i)$ in I_1 with the formula $I_2(s_i)$, in which every x is substituted by s_i. Therefore, the auxiliary variable introduced for the mixed literal $a = b$ is removed.

Theorem 2 (Soundness of (rule-eq)). *Let $a = b$ be a mixed literal with auxiliary variable x. If $I_1[EQ(x, s_1)] \ldots [EQ(x, s_n)]$ is a partial interpolant of $C_1 \vee a = b$ and $I_2(x)$ a partial interpolant of $C_2 \vee a \neq b$ then $I_1[I_2(s_1)] \ldots [I_2(s_n)]$ is a partial interpolant of the clause $C_1 \vee C_2$.*

Proof. The symbol condition for $I_1[I_2(s_1)] \ldots [I_2(s_n)]$ clearly holds if we assume that it holds for $I_1[EQ(x, s_1)] \ldots [EQ(x, s_n)]$ and $I_2(x)$. Hence, after we show (ind) and (cont), we can apply Lemma 2.

Inductivity. We assume

$$\begin{aligned} \forall x, p_x.\ &((p_x \text{ xor } a = x) \rightarrow I_1[p_x \text{ xor } x = s_1] \ldots [p_x \text{ xor } x = s_n]) \\ &\wedge (a = x \rightarrow I_2(x)) \end{aligned}$$

and show $I_1[I_2(s_1)] \ldots [I_2(s_n)]$. Instantiating $x := s_i$ for all $i \in \{1, \ldots, n\}$ and taking the second conjunct gives $\bigwedge_{i \in \{1,\ldots,n\}} (a = s_i \rightarrow I_2(s_i))$. Instantiating $p_x := \perp$ and $x := a$ and taking the first conjunct gives $I_1[a = s_1] \ldots [a = s_n]$. With monotonicity we get $I_1[I_2(s_1)] \ldots [I_2(s_n)]$ as desired.

Contradiction. We have to show

$$\begin{aligned} &I_1[I_2(s_1)] \ldots [I_2(s_n)] \rightarrow \\ &\qquad \exists x, p_x.\ (((\neg p_x \text{ xor } x = b) \wedge I_1[p_x \text{ xor } x = s_1] \ldots [p_x \text{ xor } x = s_n]) \\ &\qquad\qquad \vee (x = b \wedge I_2(x))) \end{aligned}$$

We show the implication for $p_x := \top$ and $x := b$. It simplifies to

$$I_1[I_2(s_1)] \ldots [I_2(s_n)] \rightarrow I_1[b \neq s_1] \ldots [b \neq s_n] \vee I_2(b)$$

If $I_2(b)$ holds the implication is true. If $I_2(b)$ does not hold, we have

$$\bigwedge_{i \in \{1,\ldots,n\}} (I_2(s_i) \rightarrow b \neq s_i)$$

With monotonicity we get $I_1[I_2(s_1)] \ldots [I_2(s_n)] \rightarrow I_1[b \neq s_1] \ldots [b \neq s_n]$. □

4.3 Example

We demonstrate our algorithm on the following example:

$$\begin{aligned} A \equiv &(\neg q \vee a = s_1) \wedge (q \vee a = s_2) \wedge f(a) = t \\ B \equiv &(\neg q \vee b = s_1) \wedge (q \vee b = s_2) \wedge f(b) \neq t \end{aligned}$$

The conjunction $A \wedge B$ is unsatisfiable. In this example, a is A-local, b is B-local and the remaining symbols are shared.

Assume the theory solver for $\mathcal{EUF}$ introduces the mixed literal $a = b$ and provides the lemmas (i) $f(a) \neq t \vee a \neq b \vee f(b) = t$, (ii) $a \neq s_1 \vee b \neq s_1 \vee a = b$, and (iii) $a \neq s_2 \vee b \neq s_2 \vee a = b$. Let the variable x be associated with the equality $a = b$. Then, we label the lemmas with (i) $f(x) = t$, (ii) $EQ(x, s_1)$, and (iii) $EQ(x, s_2)$.

We compute an interpolant for A and B using Pudlák's algorithm. Since the input is already in conjunctive normal form, we can directly apply resolution. Note that for Pudlák's algorithm every input clause has the partial interpolant $\bot$ ($\top$) if it is part of A (B). In the following derivation trees we apply the following simplifications without explicitely stating them:

$$F \wedge \top \equiv F$$
$$F \vee \bot \equiv F$$

From lemma (ii) and the input clauses $\neg q \vee a = s_1$ and $\neg q \vee b = s_1$ we can derive the clause $\neg q \vee a = b$. The partial interpolant of the derived clause is still $EQ(x, s_1)$.

$$\dfrac{\dfrac{\neg q \vee a = s_1 : \bot \qquad a \neq s_1 \vee b \neq s_1 \vee a = b : EQ(x, s_1)}{b \neq s_1 \vee \neg q \vee a = b : EQ(x, s_1)} \qquad b = s_1 \vee \neg q : \top}{\neg q \vee a = b : EQ(x, s_1)}$$

Similarly, from lemma (iii) and the input clauses $q \vee a = s_2$ and $q \vee b = s_2$ we can derive the clause $q \vee a = b$ with partial interpolant $EQ(x, s_2)$.

$$\dfrac{\dfrac{q \vee a = s_2 : \bot \qquad a \neq s_2 \vee b \neq s_2 \vee a = b : EQ(x, s_2)}{b \neq s_2 \vee q \vee a = b : EQ(x, s_2)} \qquad b = s_2 \vee q : \top}{q \vee a = b : EQ(x, s_2)}$$

A resolution step on these two clauses with q as pivot yields the clause $a = b$. Since q is a shared literal, Pudlák's algorithm introduces the case distinction. Hence, we get the partial interpolant $(EQ(x, s_2) \vee q) \wedge (EQ(x, s_1) \vee \neg q)$. Note that this interpolant has the form $I_1[EQ(x, s_1)][EQ(x, s_2)]$ and, therefore, satisfies the syntactical restrictions.

$$\dfrac{q \vee a = b : EQ(x, s_2) \qquad \neg q \vee a = b : EQ(x, s_1)}{a = b : (EQ(x, s_2) \vee q) \wedge (EQ(x, s_1) \vee \neg q)}$$

From the $\mathcal{EUF}$-lemma (i) and the input clauses $f(a) = t$ and $f(b) \neq t$, we can derive the clause $a \neq b$ with partial interpolant $f(x) = t$. Note that this interpolant has the form $I_2(x)$ which also corresponds to the syntactical restrictions needed for our method.

$$\dfrac{\dfrac{f(a) = t : \bot \qquad f(a) \neq t \vee a \neq b \vee f(b) = t : f(x) = t}{f(b) = t \vee a \neq b : f(x) = t} \qquad f(b) \neq t : \top}{a \neq b : f(x) = t}$$

If we apply the final resolution step on the mixed literal $a = b$ using (rule-eq), we get the interpolant $I_1[I_2(s_1)][I_2(s_2)]$ which corresponds to the interpolant $(f(s_2) = t \lor q) \land (f(s_1) = t \lor \neg q)$.

$$\frac{a = b : (EQ(x, s_2) \lor q) \land (EQ(x, s_1) \lor \neg q) \qquad a \neq b : f(x) = t}{\bot : (f(s_2) = t \lor q) \land (f(s_1) = t \lor \neg q)}$$

When resolving on q in the derivations above, the mixed literal $a = b$ occurs in both antecedents. This leads to the form $I[EQ(x, s_1)][EQ(x, s_2)]$. We can prevent this by resolving in a different order. We could first resolve the clause $q \lor a = b$ with the clause $a \neq b$ and obtain the partial interpolant $f(s_2) = t$ using (rule-eq).

$$\frac{a = b \lor q : EQ(x, s_2) \qquad a \neq b : f(x) = t}{q : f(s_2) = t}$$

Then we could resolve the clause $\neg q \lor a = b$ with the clause $a \neq b$ and obtain the partial interpolant $f(s_1) = t$ again using (rule-eq).

$$\frac{a = b \lor \neg q : EQ(x, s_1) \qquad a \neq b : f(x) = t}{\neg q : f(s_1) = t}$$

The final resolution step on q will then introduce the case distinction according to Pudlák's algorithm. This results in the same interpolant.

$$\frac{q : f(s_2) = t \qquad \neg q : f(s_1) = t}{\bot : (f(s_2) = t \lor q) \land (f(s_1) = t \lor \neg q)}$$

5 Linear Real and Integer Arithmetic

Our solver for linear arithmetic is based on a variant of the Simplex approach [10]. A theory conflict is a conjunction of literals ℓ_j of the form $\sum_i a_{ij} x_i \leq b_j$. The proof of unsatisfiability is given by Farkas coefficients $k_j \geq 0$ for each inequality ℓ_j. These coefficients have the properties $\sum_j k_j a_{ij} = 0$ and $\sum_j k_j b_j < 0$. In the following we use the notation of adding inequalities (provided the coefficients are positive). Thus, we write $\sum_j k_j \ell_j$ for $\sum_i (\sum_j k_j a_{ij}) x_i \leq \sum_j k_j b_j$. With the property of the Farkas coefficients we get a contradiction ($0 < 0$) and this shows that the theory conflict is unsatisfiable.

A conjunction of literals may have rational but no integer solutions. In this case, there are no Farkas coefficients that can prove the unsatisfiability. So for the integer case, our solver may introduce extended branches [8], which are just branches of the DPLL engine on newly introduced literals. In the proof tree this results in resolution steps with these literals as pivots.

Example 1. The formula $t \leq 2a \leq r \leq 2b + 1 \leq t$ has no integer solution but a rational solution. Introducing the branch $a \leq b \vee b < a$ leads to the theory conflicts $t \leq 2a \leq 2b \leq t - 1$ and $r \leq 2b + 1 \leq 2a - 1 \leq r - 1$ (note that $b < a$ is equivalent to $b + 1 \leq a$). The corresponding proof tree is given below. The Farkas coefficients in the theory lemmas are given in parenthesis. Note that the proof tree shows the clauses, i. e., the negated conflicts. A node with more than two parents denotes that multiple applications of the resolution rule are taken one after another.

$$
\dfrac{
\dfrac{\begin{array}{r}\neg(r \leq 2b+1)\ (\cdot 1)\\ \neg(b+1 \leq a)\ (\cdot 2)\\ \neg(2a \leq r)\ (\cdot 1)\end{array} \quad r \leq 2b+1 \quad 2a \leq r}{a \leq b}
\qquad
\dfrac{\begin{array}{r}\neg(t \leq 2a)\ (\cdot 1)\\ \neg(a \leq b)\ (\cdot 2)\\ \neg(2b+1 \leq t)\ (\cdot 1)\end{array} \quad t \leq 2a \quad 2b+1 \leq t}{\neg(a \leq b)}
}{\bot}
$$

Now consider the problem of deriving an interpolant between $A \equiv t \leq 2a \leq r$ and $B \equiv r \leq 2b + 1 \leq t$. We can obtain an interpolant by annotating the above resolution tree with partial interpolants. To compute a partial interpolant for the theory lemma $\neg(r \leq 2b+1) \vee \neg(b+1 \leq a) \vee \neg(2a \leq r)$, we purify the *negated* clause according to the definition in Section 3.2, which gives

$$r \leq 2b + 1 \wedge x_1 \leq a \wedge -x_1 + b + 1 \leq 0 \wedge 2a \leq r.$$

Then, we sum up the A-part of the conflict (the second and fourth literal) multiplied by their corresponding Farkas coefficients. This yields the interpolant $2x_1 \leq r$. Similarly, the negation of the theory lemma $\neg(t \leq 2a) \vee \neg(a \leq b) \vee \neg(2b+1 \leq t)$ is purified to

$$t \leq 2a \wedge x_2 + a \leq 0 \wedge -x_2 \leq b \wedge 2b + 1 \leq t,$$

which yields the partial interpolant $2x_2 + t \leq 0$. Note, that we have to introduce different variables for each literal. Intuitively, the variable x_1 stands for a and x_2 for $-a$. Using Pudlák's algorithm we can derive the same interpolants for the clause $a \leq b$ resp. $\neg(a \leq b)$.

For the final resolution step, the two partial interpolants $2x_1 \leq r$ and $2x_2 + t \leq 0$ are combined into the final interpolant of the problem. Summing up these inequalities with $x_1 = -x_2$ we get $t \leq r$. While this follows from A, it is not inconsistent with B. We need an additional argument that, given $r = t$, r has to be an even integer. This also follows from the partial interpolants when setting $x_1 = -x_2$: $t \leq -2x_2 = 2x_1 \leq r$. The final interpolant computed by our algorithm is $t \leq 2 \left\lfloor \frac{r}{2} \right\rfloor$.

In general, we can derive additional constraints on the variables if the constraint resulting from summing up the two partial interpolants holds very tightly. We know implicitly that $x_1 = -x_2$ is an integer value between $t/2$ and $r/2$. If t equals r or almost equals r there are only a few possible values which we can explicitly express using the division function as in the example above. We assume that the (partial) interpolant F always has a certain property. There is some term s and some constant k, such that for $s > 0$ the interpolant is always false and for $s < -k$ the interpolant is always true (in our case $s = t - r$ and $k = 0$).

For a partial interpolant that still contains auxiliary variables $\boldsymbol{x}$, we additionally require that s contains them with a positive coefficient and that F is monotone on $\boldsymbol{x}$, i. e., $\boldsymbol{x} \geq \boldsymbol{x}'$ implies $F(\boldsymbol{x}) \rightarrow F(\boldsymbol{x}')$.

To mechanise the reasoning used in the example above, our resolution rule for mixed inequality literals requires that the interpolant patterns that label the clauses have a certain shape. An auxiliary variable of a mixed inequality literal may only occur in the interpolant pattern if the negated literal appears in the clause. Let $\boldsymbol{x}$ denote the set of auxiliary variables that occur in the pattern. We require that these variables only occur inside a special sub-formula of the form $LA(s(\boldsymbol{x}), k, F(\boldsymbol{x}))$. The first parameter s is a linear term over the variables in $\boldsymbol{x}$ and arbitrary other terms not involving $\boldsymbol{x}$. The coefficients of the variables $\boldsymbol{x}$ in s must all be positive. The second parameter $k \in \mathbb{Q}_\varepsilon$ is a constant value. In the real case we only allow the values 0 and $-\varepsilon$. In the integer case we allow $k \in \mathbb{Z}, k \geq -1$. To simplify the presentation, we sometimes write $-\varepsilon$ for -1 in the integer case. The third parameter $F(\boldsymbol{x})$ is a formula that contains the variables from $\boldsymbol{x}$ at arbitrary positions. We require that F is monotone, i. e., $\boldsymbol{x} \geq \boldsymbol{x}'$ implies $F(\boldsymbol{x}) \rightarrow F(\boldsymbol{x}')$. Moreover, $F(\boldsymbol{x}) = \bot$ for $s(\boldsymbol{x}) > 0$ and $F(\boldsymbol{x}) = \top$ for $s(\boldsymbol{x}) < -k$. The sub-formula $LA(s(\boldsymbol{x}), k, F(\boldsymbol{x}))$ stands for $F(\boldsymbol{x})$ and it is only used to remember what the values of s and k are.

The intuition behind the formula $LA(s(\boldsymbol{x}), k, F(\boldsymbol{x}))$ is that $s(\boldsymbol{x}) \leq 0$ summarises the inequality chain that follows from the A-part of the formula. On this chain there may be some constraints on intermediate values. In the example above the A-part contains the chain $t \leq 2a \leq r$, which is summarised to $s \leq 0$ (with $s = t - r$). Furthermore the A-part implies that there is an even integer value between t and r. If $s < -k$ (with $k = 0$ in this case), t and r are distinct, and there always is an even integer between them. However, if $-k \leq s \leq 0$, the truth value of the interpolant depends on whether t is even.

In the remainder of the section, we will give the interpolants for the leaves produced by the linear arithmetic solver and for the resolvent of the resolution step where the pivot is a mixed linear inequality.

5.1 Leaf Interpolation

As mentioned above, our solver produces for a clause $C \equiv \neg\ell_1 \vee \cdots \vee \neg\ell_m$ some Farkas coefficients $k_1, \ldots, k_m \geq 0$ such that $\sum_j k_j \ell_j$ yields a contradiction $0 < 0$. A partial interpolant for a theory lemma can be computed by summing up the A-part of the conflict: I is defined as $\sum_j k_j(\ell_j \downharpoonleft A)$ (if $\ell_j \downharpoonleft A = \top$ we regard it as $0 \leq 0$, i. e., it is not added to the sum). It is a valid interpolant as it clearly follows from $\neg C \downharpoonleft A \iff \ell_1 \downharpoonleft A \wedge \cdots \wedge \ell_m \downharpoonleft A$. Moreover, we have that $I + \sum_j k_j(\ell_j \downharpoonleft B)$ yields $0 < 0$, since for every literal, even for mixed literals, $\ell_j \downharpoonleft A + \ell_j \downharpoonleft B = \ell_j$ holds[4]. This shows that $I \wedge \neg C \downharpoonleft B$ is unsatisfiable.

The linear constraint $\sum_j k_j(\ell_j \downharpoonleft A)$ can be expressed as $s(\boldsymbol{x}) \leq 0$. Thus, we can equivalently write this interpolant in our pattern as $LA(s(\boldsymbol{x}), -\varepsilon, s(\boldsymbol{x}) \leq 0)$.

[4] Strictly speaking this does not hold for shared literals, where $\ell \downharpoonleft A = \ell \downharpoonleft B = \ell$. In that case use $k_j = 0$ in $I + \sum_j k_j(\ell_j \downharpoonleft B)$ to see that I is indeed a partial interpolant.

Clause C: $a \neq b \vee a \leq b$	Clause C: $a \neq b \vee a \geq b$
$\neg C \downharpoonleft A$: $a = x \wedge -a + x_1 \leq 0$	$\neg C \downharpoonleft A$: $a = x \wedge a + x_2 \leq 0$
$\neg C \downharpoonleft B$: $x = b \wedge -x_1 + b < 0$	$\neg C \downharpoonleft B$: $x = b \wedge -x_2 - b < 0$
Interpolant I: $LA(-x + x_1, -\varepsilon, x_1 \leq x)$	Interpolant I: $LA(x + x_2, -\varepsilon, x \leq -x_2)$

Clause C: $a = b \vee a < b \vee a > b$
$\neg C \downharpoonleft A$: $(p_x \text{ xor } a = x) \wedge -a + x_1 \leq 0 \wedge a + x_2 \leq 0$
$\neg C \downharpoonleft B$: $(\neg p_x \text{ xor } x = b) \wedge -x_1 + b \leq 0 \wedge -x_2 - b \leq 0$
Interpolant I: $LA(x_1 + x_2, 0, x_1 \leq -x_2 \wedge (x_1 \geq -x_2 \rightarrow EQ(x, x_1)))$

Table 1. Interpolation of mixed theory combination clauses. We assume a is A-local, b is B-local, $a - b \leq 0$ has the auxiliary variable x_1, $b - a \leq 0$ has the auxiliary variable x_2 and $a = b$ the auxiliary variables x and p_x.

Since the Farkas coefficients are all positive and the auxiliary variables introduced to define $\ell \downharpoonleft A$ for mixed literals contain x positively, the resulting term $s(\boldsymbol{x})$ will also always contain x with a positive coefficient.

Theory combination lemmas. As mentioned in the preliminaries, we use theory combination clauses to propagate equalities from and to the Simplex core of the linear arithmetic solver. These clauses must also be labelled with partial interpolants. In the following we give interpolants for those theory combination lemmas. We will start with the case where no mixed literals occur, and treat lemmas containing mixed literals afterwards.

Interpolation of Non-Mixed Theory Combination Lemmas. If a theory combination lemma $t = u \vee t < u \vee t > u$ or $t \neq u \vee t \leq u$ contains no mixed literal, we can compute partial interpolants as follows. If all literals in the clause are A-local, the formula $\perp$ is a partial interpolant. If all literals are B-local, the formula $\top$ is a partial interpolant. These are the same interpolants Pudlák's algorithm would give for input clauses from A resp. B.

Otherwise, one of the literals belongs to A and one to B. The symbols t and u have to be shared between A and B since they appear in all literals. We can derive a partial interpolant by conjoining the negated literals projected to the A partition.

$$I \equiv (t \neq u) \downharpoonleft A \wedge (t \geq u) \downharpoonleft A \wedge (t \leq u) \downharpoonleft A. \qquad \text{for } t = u \vee t < u \vee t > u$$
$$I \equiv (t = u) \downharpoonleft A \wedge (t > u) \downharpoonleft A \qquad \text{for } t \neq u \vee t \leq u$$

Since we defined I as $\neg C \downharpoonleft A$, the first property of the partial interpolant holds trivially. Also $I \wedge \neg C \downharpoonleft B$ is equivalent to $\neg C$ and therefore false. The symbol condition is satisfied as t and u are shared symbols.

Interpolation of AB-Mixed Theory Combination Lemmas. If we are in the mixed case, all three literals are mixed. One of the two terms must be A-local (in the following we denote this term by a) the other term B-local (which we denote by

b). To purify the literals, we introduce a fresh auxiliary variable for each literal. Table 1 depicts all possible mixed theory lemmas together with the projections $\neg C \downharpoonright A$ and $\neg C \downharpoonright B$ and a partial interpolant of the clause.

Lemma 4. *The interpolants shown in Table 1 are correct partial interpolants of their respective clauses.*

Proof. First, we convince ourselves that these interpolants are of the right form: The variables x_1 and x_2 appear in the first parameter of LA with positive coefficients. For the first two clauses that contain the literal $a \neq b$, the interpolant is allowed to contain x at arbitrary positions. Note that in the first interpolant $x_1 \leq x$ is false for $-x + x_1 > 0$ and true for $-x + x_1 < \varepsilon$, i. e., $-x + x_1 \leq 0$. Also, $x_1 \geq x_1'$ implies $x_1 \leq x \rightarrow x_1' \leq x$. Similarly, for the second interpolant.

In the third clause, $F(x_1, x_2) = x_1 \leq -x_2 \wedge (x_1 \geq -x_2 \rightarrow EQ(x, x_1))$ is false for $x_1 + x_2 > 0$ (because of the first conjunct) and true for $x_1 + x_2 < 0$ (because the implication holds vacuously). Also, $x_1 \geq x_1'$ and $x_2 \geq x_2'$ implies $F(x_1, x_2) \rightarrow F(x_1', x_2')$. To see this, note that $F(x_1, x_2)$ is false if $x_1' \geq -x_2'$ and $x_1' \neq x_1$. The variable x appears only in an EQ-term which occurs positively in the partial interpolant.

Next we show

$$\neg C \downharpoonright A \models I \qquad \text{(Inductivity)}$$

$$\neg C \downharpoonright B \wedge I \models \bot \qquad \text{(Contradiction)}$$

Inductivity. For the clause $a \neq b \vee a \leq b$, the interpolant follows from $\neg C \downharpoonright A$, as $a = x$ and $-a + x_1 \leq 0$ imply $x_1 \leq x$. Similarly for the clause $a \neq b \vee a \geq b$, $\neg C \downharpoonright A$ contains $a = x$ and $a + x_2 \leq 0$, which implies $x \leq -x_2$. Now consider the clause $a = b \vee a < b \vee b < a$. Here, $\neg C \downharpoonright A$ implies $x_1 \leq -x_2$ and that if $x_1 \geq -x_2$ holds, then $x_1 = a = -x_2$. Hence, $x_1 \leq -x_2 \wedge x_1 \geq -x_2 \rightarrow EQ(x, x_1)$ holds.

Contradiction. Again we only show the first and third case. For the clause $C \equiv a \neq b \vee a \leq b$, note that $\neg C \downharpoonright B$ and $LA(-x + x_1, -\varepsilon, x_1 \leq x)$ give the contradiction $x_1 > b = x > x_1$. For the clause $C \equiv a = b \vee a < b \vee b < a$, $\neg C \downharpoonright B$ implies $x_1 \geq b \geq -x_2$. With $x_1 \leq -x_2$ from the interpolant this gives $x_1 = b$. Also, $x_1 \geq -x_2 \rightarrow EQ(x, x_1)$ from the interpolant gives p_x xor $x = b$. This is in contradiction with $\neg p_x$ xor $x = b$ from $\neg C \downharpoonright B$.

□

5.2 Pivoting of Mixed Literals

In this section we give the resolution rule for a step involving a mixed inequality $a + b \leq c$ as pivot element. In the following we denote the auxiliary variable of the negated literal $\neg(a + b \leq c)$ with x_1 and the auxiliary variable of $a + b \leq c$ with x_2. The intuition here is that x_1 and $-x_2$ correspond to the same value between a and $c - b$. The resolution rule for pivot element $a + b \leq c$ is as follows where the values for s_3, k_3 and F_3 are given later.

$$\frac{\begin{array}{c} C_1 \vee a + b \leq c : I_1[LA(c_1 x_1 + s_1(\boldsymbol{x}), k_1, F_1(x_1, \boldsymbol{x}))] \\ C_2 \vee \neg(a + b \leq c) : I_2[LA(c_2 x_2 + s_2(\boldsymbol{x}), k_2, F_2(x_2, \boldsymbol{x}))] \end{array}}{C_1 \vee C_2 : I_1[I_2[LA(s_3(\boldsymbol{x}), k_3, F_3(\boldsymbol{x}))]]} \qquad \text{(rule-la)}$$

The basic idea is to find for $\exists x_1 . F_1(x_1, \boldsymbol{x}) \wedge F_2(-x_1, \boldsymbol{x})$ an equivalent quantifier free formula $F_3(\boldsymbol{x})$. To achieve this we note that we only have to look on the value of F_1 for $-k_1 \leq c_1 x_1 + s_1(\boldsymbol{x}) \leq 0$, since outside of this interval F_1 is guaranteed to be true resp. false. The formula F_3 must also be monotone and satisfy the range condition. We choose

$$s_3(\boldsymbol{x}) = c_2 s_1(\boldsymbol{x}) + c_1 s_2(\boldsymbol{x}),$$

and then F_3 will be false for $s_3(\boldsymbol{x}) > 0$, since either $F_1(x_1, \boldsymbol{x})$ or $F_2(-x_1, \boldsymbol{x})$ is false. The value of k_3 must be chosen such that $s_3(\boldsymbol{x}) < -k_3$ guarantees the existence of a value x_1 with $c_1 x_1 + s_1(\boldsymbol{x}) < -k_1$ and $-c_2 x_1 + s_2(\boldsymbol{x}) < -k_2$. Hence, in the integer case, the gap between $\frac{s_2(\boldsymbol{x}) + k_2}{c_2}$ and $\frac{-s_1(\boldsymbol{x}) - k_1}{c_1}$ should be bigger than one. Then, $c_1 c_2 < c_2(-s_1(\boldsymbol{x}) - k_1) - c_1(s_2(\boldsymbol{x}) + k_2)$. So if we define

$$k_3 = c_2 k_1 + c_1 k_2 + c_1 c_2,$$

then there is a suitable x_1 for $s_3(\boldsymbol{x}) < -k_3$. For F_3 we can then use a finite case distinction over all values where the truth value of F_1 is not determined. This suggests defining

$$F_3(\boldsymbol{x}) :\equiv \bigvee_{i=0}^{\lceil \frac{k_1+1}{c_1} \rceil} F_1\left(\left\lfloor \frac{-s_1(\boldsymbol{x})}{c_1} \right\rfloor - i, \boldsymbol{x}\right) \wedge F_2\left(i - \left\lfloor \frac{-s_1(\boldsymbol{x})}{c_1} \right\rfloor, \boldsymbol{x}\right) \qquad \text{(int case)}$$

In the real case, if $k_1 = -\varepsilon$, the best choice is $x_1 = \frac{-s_1(\boldsymbol{x})}{c_1}$, for which $F_1(x_1)$ is guaranteed to be true. If $k_1 = 0$, we need to consider two cases:

$$\begin{aligned} k_3 &:= \begin{cases} k_2 & \text{if } k_1 = -\varepsilon \\ 0 & \text{if } k_1 = 0 \end{cases} \\ F_3(\boldsymbol{x}) &:= \begin{cases} F_2\left(\frac{s_1(\boldsymbol{x})}{c_1}, \boldsymbol{x}\right) & \text{if } k_1 = -\varepsilon \\ s_3(\boldsymbol{x}) < 0 \vee \left(F_1\left(-\frac{s_1(\boldsymbol{x})}{c_1}, \boldsymbol{x}\right) \wedge F_2\left(\frac{s_1(\boldsymbol{x})}{c_1}, \boldsymbol{x}\right)\right) & \text{if } k_1 = 0 \end{cases} \end{aligned} \qquad \text{(real case)}$$

Note that the formula of the integer case is asymmetric. If $\left\lceil \frac{k_2+1}{c_2} \right\rceil < \left\lceil \frac{k_1+1}{c_1} \right\rceil$ we can replace $-s_1$ by s_2, k_1 by k_2, and c_1 by c_2. This leads to a fewer number of disjuncts in F_3. Also note that we can remove F_1 from the last disjunct of F_3, as it will always be true.

With these definitions we can state the following lemma.

Lemma 5. *Let for $i = 1, 2$, $s_i(\boldsymbol{x})$ be linear terms over $\boldsymbol{x}$, $c_i \geq 0$, $k_i \in \mathbb{Z}_{\geq -1}$ (integer case) or $k_i \in \{0, -\varepsilon\}$ (real case), $F_i(x_i, \boldsymbol{x})$ monotone formulas with $F_i(x_i, \boldsymbol{x}) = \bot$ for $c_i x_i + s_i(\boldsymbol{x}) > 0$ and $F_i(x_i, \boldsymbol{x}) = \top$ for $c_i x_i + s_i(\boldsymbol{x}) < -k_i$. Let s_3, k_3, F_3 be as defined above. Then F_3 is monotone, $F_3(\boldsymbol{x}) = \bot$ for $s_3(\boldsymbol{x}) > 0$ and $F_3(\boldsymbol{x}) = \top$ for $s_3(\boldsymbol{x}) < -k_i$.*

Proof. Since F_1 and F_2 are monotone and they occur only positively in F_3, F_3 must also be monotone. If $s_3(\boldsymbol{x}) > 0$, then $\frac{-s1(\boldsymbol{x})}{c_1} < \frac{s_2}{c_2}$. Hence, for every $x \leq \frac{-s(\boldsymbol{x})}{c_1}$, $F_2(-x, \boldsymbol{x})$ is false since $-c_2 x + s_2(\boldsymbol{x}) > 0$. By definition, every disjunct of F_3 (except $s_3(\boldsymbol{x}) < 0$) contains $F_2(-x, \boldsymbol{x})$ for such an x, so $F_3(\boldsymbol{x})$ is false.

Now assume $s_3(\boldsymbol{x}) < -k_3$. For $k_1 = -\varepsilon$ in the real case, $F_3(\boldsymbol{x}) = F_2(-\frac{s_1(\boldsymbol{x})}{c_1})$ is true since $s_1(\boldsymbol{x}) + s_2(\boldsymbol{x}) < -k_2$. For $k_1 = 0$, F_3 is true by definition. In the integer case define $y := \left\lfloor \frac{-s_1(\boldsymbol{x})}{c_1} \right\rfloor - \left\lceil \frac{k_1+1}{c_1} \right\rceil$. This implies $c_1 y \leq -s_1(\boldsymbol{x}) - k_1 - 1$, hence $F_1(y, \boldsymbol{x})$ holds. Also $c_1 y \geq -s_1(\boldsymbol{x}) - k_1 - c_1$, hence

$$c_1 c_2 y + c_1 s_2(\boldsymbol{x}) \geq -s_3(\boldsymbol{x}) - c_2 k_1 - c_1 c_2 > k_3 - c_2 k_1 - c_1 c_2 = c_1 k_2.$$

Therefore, $F_2(-y, \boldsymbol{x})$ holds. Since y is included in the big disjunction of F_3, $F_3(\boldsymbol{x})$ is true.

Lemma 6. *Let for $i = 1, 2$, $s_i(\boldsymbol{x})$ be linear terms over $\boldsymbol{x}$, $c_i \geq 0$, $k_i \in \mathbb{Z}_{\geq -1}$ (integer case) or $k_i \in \{0, -\varepsilon\}$ (real case), $F_i(x_i, \boldsymbol{x})$ monotone formulas with $F_i(x_i, \boldsymbol{x}) = \bot$ for $c_i x_i + s_i(\boldsymbol{x}) > 0$ and $F_i(x_i, \boldsymbol{x}) = \top$ for $c_i x_i + s_i(\boldsymbol{x}) < -k_i$. Let F_3 be as defined above. Then*

$$F_3(\boldsymbol{x}) \leftrightarrow (\exists x_1 . F_1(x_1, \boldsymbol{x}) \wedge F_2(-x_1, \boldsymbol{x}))$$

Proof (for $\mathcal{LA}(\mathbb{Z})$). Since F_3 is a disjunction of $F_1(x, \boldsymbol{x}) \wedge F_2(-x, \boldsymbol{x})$ for different values of x, the implication from left to right is obvious. We only need to show the other direction. For this, choose x_1 such that $F_1(x_1, \boldsymbol{x}) \wedge F_2(-x_1, \boldsymbol{x})$ holds. We show $F_3(\boldsymbol{x})$. We define $y := \left\lfloor \frac{-s_1(\boldsymbol{x})}{c_1} \right\rfloor - \left\lceil \frac{k_1+1}{c_1} \right\rceil$. This implies $y \leq \frac{-s_1(\boldsymbol{x}) - k_1 - 1}{c_1}$. We show F_3 by a case split on $x_1 < y$.

Case $x_1 < y$. Since F_2 is monotone and $-x_1 > -y$, we have $F_2(-y, \boldsymbol{x})$. Also $F_1(y, \boldsymbol{x})$ holds since $c_1 y + s_1(\boldsymbol{x}) < -k_1$. This implies $F_3(\boldsymbol{x})$, since $F_1(y, \boldsymbol{x}) \wedge F_2(-y, \boldsymbol{x})$ is a disjunct of F_3.

Case $y \leq x_1$. Since $F_1(x_1, \boldsymbol{x})$ holds, $c_1 x_1 + s_1(\boldsymbol{x}) \leq 0$, hence $x_1 \leq \left\lfloor \frac{-s_1(\boldsymbol{x})}{c_1} \right\rfloor$. Thus, x_1 is one of the values $\left\lfloor \frac{-s_1(\boldsymbol{x})}{c_1} \right\rfloor - i$ for $0 \leq i \leq \left\lceil \frac{k_1+1}{c_1} \right\rceil$. This means the disjunction $F_3(\boldsymbol{x})$ includes $F_1(x_1, \boldsymbol{x}) \wedge F_2(-x_1, \boldsymbol{x})$. □

Proof (for $\mathcal{LA}(\mathbb{Q})$). In the case $k_1 = -\varepsilon$, $F_1(\frac{-s_1}{c_1}, \boldsymbol{x})$ is true. From the definition of F_3, we get the implication $F_3(\boldsymbol{x}) \rightarrow \exists x_1 . F_1(x_1, \boldsymbol{x}) \wedge F_2(-x_1, \boldsymbol{x})$ for $x_1 = \frac{-s_1}{c_1}$.

If $k_1 = 0$ and $s_3(x) < 0$, then $\frac{s_2}{c_2} < \frac{-s_1}{c_1}$ and for any value x_1 in between, $F_1(x_1, \boldsymbol{x}) \wedge F_2(-x_1, \boldsymbol{x})$ are true.

For the other direction assume that $F_1(x_1, \boldsymbol{x}) \wedge F_2(-x_1, \boldsymbol{x})$ holds. Since F_1 is not false, $x_1 \leq \frac{-s_1}{c_1}$ holds. If $x_1 = \frac{-s_1}{c_1}$ then F_3 holds by definition. In the case $k_1 = 0$ where $x_1 < \frac{-s_1}{c_1}$, we have $s_3(\boldsymbol{x}) < 0$, since $F_2(-x_1, \boldsymbol{x})$ is not false. In the case $k_1 = -\varepsilon$, we need to show that $F_2(\frac{s_1}{c_1}, \boldsymbol{x})$ holds. This follows from $x_1 \leq \frac{-s_1}{c_1}$ and monotonicity of F_2. □

This lemma can be used to show that (rule-la) is correct.

Theorem 3 (Soundness of (rule-la)). *Let $a + b \leq c$ be a mixed literal with the auxiliary variable x_2, and x_1 be the auxiliary variable of the negated literal. If $I_1[LA(c_1x_1{+}s_1, k_1, F_1)]$ is a partial interpolant of $C_1 \vee a{+}b \leq c$ and $I_2[LA(c_2x_2{+}s_2, k_2, F_2)]$ is a partial interpolant of $C_2 \vee \neg(a + b \leq c)$ then $I_1[I_2[LA(s_3, k_3, F_3)]]$ is a partial interpolant of the clause $C_1 \vee C_2$.*

To ease the presentation, we gave the rule (rule-la) with only one LA term per partial interpolant. The generalised rule requires the partial interpolants of the premises to have the shapes $I_1[LA_{11}] \ldots [LA_{1n}]$ and $I_2[LA_{21}] \ldots [LA_{2m}]$. The resulting interpolant is

$$I_1[I_2[LA_{311}] \ldots [LA_{31m}]] \ldots [I_2[LA_{3n1}] \ldots [LA_{3nm}]]$$

where LA_{3ij} is computed from LA_{1i} and LA_{2j} as explained above.

Proof. The symbol condition holds for I_3 if it holds for I_1 and I_2, which can be seen as follows. The only symbol that is allowed to occur in I_1 resp. I_2 but not in I_3 is the auxiliary variable introduced by the literal, i.e., x_1 resp. x_2. This variable may only occur inside the LA_1 resp. LA_2 terms as indicated and, by construction, x_1 and x_2 do not occur in LA_3. Furthermore, the remaining variables from $\boldsymbol{x}$ occur in $s_3(\boldsymbol{x})$ with a positive coefficient as required by our pattern and occur only inside the LA pattern in s_3 and F_3. Thus I_3 has the required form. We will use Lemma 2 to show that I_3 is a partial interpolant. For this we need to show inductivity (ind) and contradiction (cont).

In this proof we will use $I_1[LA_{1i}(x_1)]$ to denote the first interpolant

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

and similarly $I_2[LA_{2j}(x_2)]$ and $I_1[I_2[LA_{3ij}]]$, the latter standing for

$$I_1[I_2[LA_{311}] \ldots [LA_{31m}]] \ldots [I_2[LA_{3n1}] \ldots [LA_{3nm}]]$$

where

$$LA_{3ij} = LA(c_{2j}s_{1i} + c_{1i}s_{2j}, k_{3ij}, F_{3ij}).$$

Inductivity. We apply Lemma 6 on $x_1 = a$, which gives us

$$\bigwedge_{ij} LA_{1i}(a) \land LA_{2j}(-a) \to LA_{3ij}$$

Using the deep substitution lemma, we obtain

$$I_1\left[LA_{1i}(a)\right] \land I_2\left[LA_{2j}(-a)\right] \to I_1\left[I_2\left[LA_{3ij}\right]\right]. \qquad (*)$$

Now assume the left-hand-side of (ind), which in this case is

$$\forall x_1, x_2.\, (-a + x_1 \leq 0 \to I_1[LA_{1i}(x_1)]) \land (a + x_2 \leq 0 \to I_2[LA_{2j}(x_2)]).$$

Instantiating x_1 with a and x_2 with $-a$ gives us $I_1[LA_{1i}(a)]$ and $I_2[LA_{2j}(-a)]$. Thus by $(*)$, $I_3 \equiv I_1[I_2[LA_{3ij}]]$ holds as desired.

Contradiction. We assume $I_1[I_2[LA_{3ij}]]$ and show

$$\exists x_1, x_2.\, (-x_1 - b < -c \land I_1[LA_{1i}(x_1)]) \lor (-x_2 + b \leq c \land I_2[LA_{2j}(x_2)]) \qquad (*)$$

We do a case distinction on

$$\bigwedge_{i} (I_2[LA_{3ij}] \to \exists x_1.\, x_1 > c - b \land LA_{1i}(x_1))$$

If it holds, then we may get a different value for x_1 for every i. However, if $LA_{1i}(x_1)$ holds for some value, it also holds for any smaller value of x_1. Take x as the minimum of these values (or $x = c - b + 1$ if the implication holds vacuously for every i). Then, $-x - b < -c$ and $\bigwedge_i (I_2[LA_{3ij}] \to LA_{1i}(x))$. With monotonicity we get from $I_1[I_2[LA_{3ij}]]$ that $I_1[LA_{1i}(x)]$ holds. Hence, the left disjunct of formula $(*)$ holds.

In the other case there is some i with

$$I_2[LA_{3ij}] \land (\forall x_1.\, x_1 > c - b \to \neg LA_{1i}(x_1)). \qquad (**)$$

The second part of Lemma 6 gives us

$$\bigwedge_{j} (LA_{3ij} \to \exists x_1.\, LA_{1i}(x_1) \land LA_{2j}(-x_1))$$

Then, $x_1 \leq c - b$ by $(**)$. But if $LA_{2j}(-x_1)$ holds, then LA_{2j} also holds for the smaller value $b - c$. This gives us

$$\bigwedge_{j} (LA_{3ij} \to LA_{2j}(b - c))$$

We obtain $I_2[LA_{2j}(b - c)]$ by applying monotonicity on the left conjunct of formula $(**)$. Thus the right disjunct of formula $(*)$ holds for $x_2 = b - c$. $\square$

6 An Example for the Combined Theory

The previous examples showed how to use our technique to compute an interpolant in the theory of uninterpreted functions, or the theory of linear arithmetic. We will now present an example in the combination of these theories by applying our scheme to a proof of unsatisfiability of the interpolation problem

$$A \equiv t \leq 2a \wedge 2a \leq s \wedge f(a) = q$$
$$B \equiv s \leq 2b \wedge 2b \leq t + 1 \wedge \neg(f(b) = q)$$

where a, b, s, and t are integer constants, q is a constant of the uninterpreted sort U, and f is a function from integer to U.

We derive the interpolant using Pudlák's algorithm and the rules shown in this paper. Note that the formula is already in conjunctive normal form. Since we use Pudlák's algorithm, every input clause is labelled with $\bot$ if it is an input clause from A, and $\top$ if it is an input clause from B. We will simplify the interpolants by removing neutral elements of Boolean connectives.

Since the variables a and b are shared between the theory of uninterpreted functions and the theory of linear arithmetic, we get some theory combination clauses for a and b. The only theory combination clause needed to prove unsatisfiability of $A \wedge B$ is $a = b \vee \neg(b \leq a) \vee \neg(a \leq b)$ which has the partial interpolant $LA(x_1 + x_2, 0, F[EQ(x, x_1)])$ where $F[G] \equiv x_1 \leq -x_2 \wedge (x_1 \geq -x_2 \rightarrow G)$. Here, x_1 is used to purify[5] $b \leq a$ and x_2 is used to purify $a \leq b$.

We get two lemmas from $\mathcal{LA}(\mathbb{Z})$: The first one, $\neg(2a \leq s) \vee \neg(s \leq 2b) \vee a \leq b$, states that we can derive $a \leq b$ from $2a \leq s$ and $s \leq 2b$. Let x_3 be the variable used to purify $\neg(a \leq b)$. Note that we purify the literals in the conflict, i. e., the negation of the lemma. Then, this lemma can be annotated with the partial interpolant $LA(2x_3 - s, -1, 2x_3 \leq s)$. We can resolve this lemma with the unit clauses from the input to get $a \leq b$.

$$\dfrac{\dfrac{\neg(2a \leq s) \vee \neg(s \leq 2b) \vee a \leq b : LA(2x_3 - s, -1, 2x_3 \leq s) \qquad 2a \leq s : \bot}{\neg(s \leq 2b) \vee a \leq b : LA(2x_3 - s, -1, 2x_3 \leq s) \qquad s \leq 2b : \top}}{a \leq b : LA(2x_3 - s, -1, 2x_3 \leq s)}$$

The second $\mathcal{LA}(\mathbb{Z})$-lemma, $\neg(t \leq 2a) \vee \neg(2b \leq t + 1) \vee b \leq a$, states that we can derive $b \leq a$ from $t \leq 2a$ and $2b \leq t + 1$. Let x_4 be the variable used to purify $\neg(b \leq a)$. Then, we can annotate the lemma with the partial interpolant $LA(2x_4 + t, -1, 2x_4 + t \leq 0)$ and propagate this partial interpolant to the unit clause $b \leq a$ by resolution with input clauses.

$$\dfrac{\dfrac{\neg(t \leq 2a) \vee \neg(2b \leq t + 1) \vee b \leq a : LA(2x_4 + t, -1, 2x_4 + t \leq 0) \quad t \leq 2a : \bot}{\neg(2b \leq t + 1) \vee b \leq a : LA(2x_4 + t, -1, 2x_4 + t \leq 0) \; 2b \leq t + 1 : \top}}{b \leq a : LA(2x_4 + t, -1, 2x_4 + t \leq 0)}$$

[5] Note that we purify the conflict, i. e., the negated clause

Additionally, we get one lemma from $\mathcal{EUF}$, $f(b) = q \vee \neg(f(a) = q) \vee \neg(a = b)$, that states that, given $f(a) = q$ and $a = b$, by congruence, $f(b) = q$ has to hold. Let x be the variable used to purify $a = b$. Then, we can label this lemma with the partial interpolant $f(x) = q$. Note that this interpolant has the form $I(x)$ as required by our interpolation scheme. We propagate this partial interpolant to the unit clause $\neg(a = b)$ by resolving the lemma with the input clauses.

$$\cfrac{\cfrac{f(b) = q \vee \neg(f(a) = q) \vee \neg(a = b) : f(x) = q \qquad f(b) = q : \top}{\neg(f(a) = q) \vee \neg(a = b) : f(x) = q} \qquad f(a) = q : \bot}{\neg(a = b) : f(x) = q}$$

From the theory combination clause $a = b \vee \neg(b \leq a) \vee \neg(a \leq b)$ and the three unit clauses derived above, we show a contradiction. We start by resolving with the unit clause $a = b$ using (rule-eq) and produce the partial interpolant $LA(x_1 + x_2, 0, f(x_1) = q)$.

$$\frac{\begin{array}{c} a = b \vee \neg(b \leq a) \vee \neg(a \leq b) : LA(x_1 + x_2, 0, F[EQ(x, x_1)]) \\ \neg(a = b) : f(x) = q \end{array}}{\neg(b \leq a) \vee \neg(a \leq b) : LA(x_1 + x_2, 0, F[f(x_1) = q])}$$

The next step resolves on $b \leq a$ using (rule-la). Note that we used x_1 to purify $b \leq a$ and x_4 to purify $\neg(b \leq a)$. Hence, these variables will be removed from the resulting partial interpolant. From the partial interpolants of the antecedents, $LA(2x_4 + t, -1, 2x_4 + t \leq 0)$ and $LA(x_1 + x_2, 0, F[f(x_1) = q])$, we get the following components:

$$\begin{array}{llll} c_1 = 2 & s_1 = t & k_1 = -1 & F_1(x_4) \equiv 2x_4 + t \leq 0 \\ c_2 = 1 & s_2 = x_2 & k_2 = 0 & F_2(x_1) \equiv F[f(x_1) = q] \end{array}$$

These components yield $k_3 = 1 \cdot (-1) + 2 \cdot 0 + 2 \cdot 1 = 1$. Furthermore, $\left\lceil \frac{k_1 + 1}{c_1} \right\rceil = 0$ leads to one disjunct in F_3. The corresponding values are $\left\lfloor \frac{-t}{2} \right\rfloor$, resp. $-\left\lfloor \frac{-t}{2} \right\rfloor$. $F_1(\left\lfloor \frac{-t}{2} \right\rfloor)$ is always true and can be omitted. The resulting formula $G(x_2) := F_3(\boldsymbol{x})$ is

$$G(x_2) \equiv -\left\lfloor \frac{-t}{2} \right\rfloor \leq -x_2 \wedge \left(\left\lfloor \frac{-t}{2} \right\rfloor \geq -x_2 \rightarrow f\left(-\left\lfloor \frac{-t}{2} \right\rfloor \right) = q \right).$$

The partial interpolant for the clause $\neg(a \leq b)$ is $LA(t + 2x_2, 1, G(x_2))$.

$$\frac{\begin{array}{c} b \leq a : LA(2x_4 + t, -1, 2x_4 + t \leq 0) \\ \neg(b \leq a) \vee \neg(a \leq b) : LA(x_1 + x_2, 0, f(x_1) = q) \end{array}}{\neg(a \leq b) : LA(t + 2x_2, 1, G(x_2))}$$

In the final resolution step, we resolve $a \leq b$ labelled with partial interpolant $LA(2x_3 - s, -1, 2x_3 \leq s)$ against $\neg(a \leq b)$ labelled with $LA(t + 2x_2, 1, G(x_2))$.

Note that the literals have been purified with x_3 and x_2, respectively. We get the components

$$\begin{array}{llll} c_1 = 2 & s_1 = -s & k_1 = -1 & F_1(x_3) \equiv 2x_3 \leq s \\ c_2 = 2 & s_2 = t & k_2 = 1 & F_2(x_2) \equiv G(x_2). \end{array}$$

We get $k_3 = 2 \cdot (-1) + 2 \cdot 1 + 2 \cdot 2 = 4$. Again, $\left\lceil \frac{k_1+1}{c_1} \right\rceil = 0$ yields one disjunct in F_3 with the values $\left\lfloor \frac{s}{2} \right\rfloor$, and $-\left\lfloor \frac{s}{2} \right\rfloor$, respectively. Again, $F_1(\left\lfloor \frac{s}{2} \right\rfloor)$ is always true and can be omitted. The resulting formula is

$$\begin{aligned} H &\equiv G\left(-\left\lfloor \frac{s}{2} \right\rfloor\right) \\ &\equiv -\left\lfloor \frac{-t}{2} \right\rfloor \leq \left\lfloor \frac{s}{2} \right\rfloor \wedge \left(\left\lfloor \frac{-t}{2} \right\rfloor \geq \left\lfloor \frac{s}{2} \right\rfloor \rightarrow f\left(-\left\lfloor \frac{-t}{2} \right\rfloor\right) = q\right). \end{aligned}$$

The final resolution step yields an interpolant for this problem.

$$\frac{a \leq b : LA(2x_3 - s, -1, \bot) \qquad \neg(a \leq b) : LA(t + 2x_2, 1, G(x_2))}{\bot : LA(-2s + 2t, 4, H)}$$

Thus H is the final interpolant. Now we argue validity of this interpolant.

Interpolant follows from the A-part. The A-part contains $2a \leq s$, which implies $a \leq \left\lfloor \frac{s}{2} \right\rfloor$. From $t \leq 2a$ we get $-\left\lfloor \frac{-t}{2} \right\rfloor \leq a$. Hence, $-\left\lfloor \frac{-t}{2} \right\rfloor \leq \left\lfloor \frac{s}{2} \right\rfloor$. Moreover, $-\left\lfloor \frac{-t}{2} \right\rfloor \geq \left\lfloor \frac{s}{2} \right\rfloor$ implies $-\left\lfloor \frac{-t}{2} \right\rfloor = a$. So with the A-part we get $f(-\left\lfloor \frac{-t}{2} \right\rfloor) = q$.

Interpolant is inconsistent with the B-part. The B-part implies $s \leq 2b \leq t+1$. Hence, we have $\left\lfloor \frac{s}{2} \right\rfloor \leq b \leq \left\lfloor \frac{t+1}{2} \right\rfloor$. A case distinction on whether t is even or odd yields $\left\lfloor \frac{t+1}{2} \right\rfloor = -\left\lfloor \frac{-t}{2} \right\rfloor$. Therefore, $\left\lfloor \frac{s}{2} \right\rfloor \leq b \leq -\left\lfloor \frac{-t}{2} \right\rfloor$ holds. Hence, the interpolant guarantees $f(-\left\lfloor \frac{-t}{2} \right\rfloor) = q$ and $-\left\lfloor \frac{-t}{2} \right\rfloor \leq \left\lfloor \frac{s}{2} \right\rfloor$. Hence, $b = -\left\lfloor \frac{-t}{2} \right\rfloor$ and with $f(b) \neq q$ from the B-part we get a contradiction.

Symbol condition is satisfied. The symbol condition is trivially satisfied since $symb(A) = \{a, t, s, f, q\}$ and $symb(B) = \{b, t, s, f, q\}$. The shared symbols are t, s, f, and q which are exactly the symbols occurring in the interpolant.

7 Conclusion and Future Work

We presented a novel interpolation scheme to extract Craig interpolants from resolution proofs produced by SMT solvers without restricting the solver or reordering the proofs. The key ingredients of our method are virtual purifications of troublesome mixed literals, syntactical restrictions of partial interpolants, and specialised interpolation rules for pivoting steps on mixed literals.

In contrast to previous work, our interpolation scheme does not need specialised rules to deal with extended branches as commonly used in state-of-the-art SMT solvers to solve $\mathcal{LA}(\mathbb{Z})$-formulae. Furthermore, our scheme can deal with resolution steps where a mixed literal occurs in both antecedents, which are forbidden by other schemes [5,12].

Our scheme works for resolution based proofs in the DPLL(T) context provided there is a procedure that generates partial interpolants with our syntactic restrictions for the theory lemmas. We sketched these procedures for the theory lemmas generated by either congruence closure or linear arithmetic solvers producing Farkas proofs. In this paper, we limited the presentation to the combination of the theory of uninterpreted functions, and the theory of linear arithmetic over the integers or the reals. Nevertheless, the scheme could be extended to support other theories. This requires defining the projection functions for mixed literals in the theory, defining a pattern for partial interpolants, and proving a corresponding resolution rule.

We plan to produce interpolants of different strengths using the technique from D'Silva et al. [9]. This is orthogonal to our interpolation scheme (particularly to the partial interpolants used for mixed literals). Furthermore, we want to extend the correctness proof to show that our scheme works with inductive sequences of interpolants [21] and tree interpolants [15]. We also plan to extend this scheme to other theories including arrays and quantifiers.

References

1. Dirk Beyer, Damien Zufferey, and Rupak Majumdar. CSIsat: Interpolation for LA+EUF. In *CAV*, pages 304–308. Springer, 2008.
2. Angelo Brillout, Daniel Kroening, Philipp Rümmer, and Thomas Wahl. Beyond quantifier-free interpolation in extensions of Presburger arithmetic. In *VMCAI*, pages 88–102. Springer, 2011.
3. Roberto Bruttomesso, Simone Rollini, Natasha Sharygina, and Aliaksei Tsitovich. Flexible interpolation with local proof transformations. In *ICCAD*, pages 770–777. IEEE, 2010.
4. Jürgen Christ, Jochen Hoenicke, and Alexander Nutz. SMTInterpol: An interpolating SMT solver. In *SPIN*, pages 248–254. Springer, 2012.
5. Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani. Efficient interpolant generation in satisfiability modulo theories. In *TACAS*, pages 397–412. Springer, 2008.
6. William Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *J. Symb. Log.*, 22(3):269–285, 1957.
7. David Detlefs, Greg Nelson, and James B. Saxe. Simplify: A theorem prover for program checking. *J. ACM*, 52(3):365–473, 2005.
8. Isil Dillig, Thomas Dillig, and Alex Aiken. Cuts from proofs: A complete and practical technique for solving linear inequalities over integers. In *CAV*, pages 233–247. Springer, 2009.
9. Vijay D'Silva, Daniel Kroening, Mitra Purandare, and Georg Weissenbacher. Interpolant strength. In *VMCAI*, pages 129–145. Springer, 2010.
10. Bruno Dutertre and Leonardo de Moura. A fast linear-arithmetic solver for DPLL(T). In *CAV*, pages 81–94. Springer, 2006.
11. Alexander Fuchs, Amit Goel, Jim Grundy, Sava Krstic, and Cesare Tinelli. Ground interpolation for the theory of equality. In *TACAS*, pages 413–427. Springer, 2009.

12. Amit Goel, Sava Krstic, and Cesare Tinelli. Ground interpolation for combined theories. In *CADE*, pages 183–198. Springer, 2009.
13. Alberto Griggio. A practical approach to satisability modulo linear integer arithmetic. *JSAT*, 8(1/2):1–27, 2012.
14. Alberto Griggio, Thi Thieu Hoa Le, and Roberto Sebastiani. Efficient interpolant generation in satisfiability modulo linear integer arithmetic. In *TACAS*, pages 143–157. Springer, 2011.
15. Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. Nested interpolants. In *POPL*, pages 471–482. ACM, 2010.
16. Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth L. McMillan. Abstractions from proofs. In *POPL*, pages 232–244. ACM, 2004.
17. Himanshu Jain, Edmund M. Clarke, and Orna Grumberg. Efficient craig interpolation for linear diophantine (dis)equations and linear modular equations. *Formal Methods in System Design*, 35(1):6–39, 2009.
18. Daniel Kroening, Jérôme Leroux, and Philipp Rümmer. Interpolating quantifier-free Presburger arithmetic. In *LPAR*, pages 489–503. Springer, 2010.
19. Christopher Lynch and Yuefeng Tang. Interpolants for linear arithmetic in SMT. In *ATVA*, pages 156–170. Springer, 2008.
20. Kenneth L. McMillan. An interpolating theorem prover. In *TACAS*, pages 16–30. Springer, 2004.
21. Kenneth L. McMillan. Lazy abstraction with interpolants. In *CAV*, pages 123–136. Springer, 2006.
22. Kenneth L. McMillan. Interpolants from Z3 proofs. In *FMCAD*, pages 19–27. FMCAD Inc., 2011.
23. Greg Nelson and Derek C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. Program. Lang. Syst.*, 1(2):245–257, 1979.
24. Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Abstract DPLL and abstract DPLL modulo theories. In *LPAR*, pages 36–50. Springer, 2004.
25. Pavel Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *J. Symb. Log.*, 62(3):981–998, 1997.
26. Greta Yorsh and Madanlal Musuvathi. A combination method for generating interpolants. In *CADE*, pages 353–368. Springer, 2005.