Farkas-Based Tree Interpolation*

Sepideh Asadi[1], Martin Blicha[1,2], Antti Hyvärinen[1],
Grigory Fedyukovich[3], and Natasha Sharygina[1]

[1] Università della Svizzera Italiana (USI), Lugano, Switzerland
{firstname.lastname}@usi.ch
[2] Charles University, Faculty of Mathematics and Physics, Czech Republic
[3] Florida State University, Tallahassee, USA
grigory@cs.fsu.edu

Abstract. Linear arithmetic over reals (LRA) underlies a wide range of SMT-based modeling approaches, and, strengthened with Craig interpolation using Farkas' lemma, is a central tool for efficient over-approximation. Recent advances in LRA interpolation have resulted in a range of promising interpolation algorithms with so far poorly understood properties. In this work we study the Farkas-based algorithms with respect to tree interpolation, a practically important approach where a set of interpolants is constructed following a given tree structure. We classify the algorithms based on whether they guarantee the tree interpolation property, and present how to lift a recently introduced approach producing conjunctive LRA interpolants to tree interpolation in the quantifier-free LRA fragment of first-order logic. Our experiments show that the standard interpolation and the approach using conjunctive interpolants are complementary in tree interpolation, and suggest that their combination would be very powerful in practice.

Keywords: Craig interpolation · Tree interpolation property · LRA interpolation systems · SMT Solving · Symbolic model checking.

1 Introduction

Given an unsatisfiable first-order formula ϕ partitioned into two sets A and B, a (binary) *Craig interpolant* [9] is a formula I that is implied by A, unsatisfiable with B, and defined on the shared symbols of A and B. For certain applications it is useful to consider sets of related interpolants obtained by partitioning ϕ in different ways into A, B. Often these applications require further properties to hold for the computed interpolants. For example, consider the following scenario from *upgrade checking* of software [15]: a program with function calls is modeled together with safety properties as an unsatisfiable formula. Once a programmer introduces changes to the functions, it is often important to know whether the same properties are satisfied by the new program. The key insight in the use of

* This work was supported by Swiss National Science Foundation grant 200021_185031 and by Czech Science Foundation grant 20-07487S.

Table 1: Validity of the tree interpolation property (TI) for the studied interpolation algorithms. The signs ✓ and × indicate, respectively, that the property holds and fails.

Interpolation algorithm	**TI**	**Theorem**
Farkas Itp^F	✓	1, 2
Dual Farkas $\overline{Itp^F}$	×	7
Decomposing Farkas Itp^D	✓	3, 4, 5
Dual Decomposing Farkas $\overline{Itp^D}$	×	Corollary 2
Flexible Farkas $Itp^{(\alpha)}$	×	6

interpolants in this scenario is as follows. If each function is over-approximated by an interpolant, it is sufficient, under certain conditions, to check whether each new function is contained in the corresponding old function's interpolant in the logical sense. Such a check might be significantly lighter than re-verification of the whole program. However, if several functions were changed simultaneously, the resulting interpolants need to guarantee that no matter how the functions were changed, as long as the changes stay within the over-approximations, the program is correct. This requirement places restrictions on interpolants and ultimately on the interpolation algorithms.

It turns out that these conditions are guaranteed to hold if the interpolation algorithm satisfies the *tree interpolation property (TI)* (see Sec. 2). The property is useful not only in the above scenario, but also in many other applications, including solving constrained Horn clauses [16, 27], and synthesis [13].

Many modeling approaches used in verification rely heavily on the use of linear algebra either by directly encoding arithmetic operations, or as part of an algorithm for more general arithmetic. As a result, over-approximation in linear real arithmetic (LRA) through interpolation is an active research area. *Farkas interpolation* [23] is a central class of algorithms for LRA interpolation and is widely used in verification tools. The idea underlying these algorithms is to first show a linear system unsatisfiable with a decision version of the Simplex algorithm. The *Farkas coefficients* computed as a side product can then be restricted to the linear system belonging to the A-part to obtain the interpolant.

Having different algorithms for interpolation is of great practical interest since the choice of a good interpolation algorithm may well determine whether an application terminates quickly or diverges. Up to now there have been few interpolation algorithms that guarantee TI in LRA, and we believe that this has severely limited their use in practical applications. In this work, we identify five variants of the Farkas interpolation algorithms and show that only two of them can be used as a basis for tree interpolation algorithms. Our results are summarized in Table 1.[4]

The algorithm Itp^F, introduced in [23], produces a single inequality, whereas Itp^D [6] is a more recent algorithm based on *decomposing* the Farkas interpolants

[4] The Farkas interpolation algorithm Itp^F guarantees TI (see, e.g., [7]). We show this for a stronger notion of tree interpolants.

into conjunctions. Applying Itp^F in tree interpolation is relatively straightforward. However, for Itp^D, it is important to take the tree structure of the interpolation problem into consideration when constructing the decomposition. We show how the global tree structure can be brought to the binary interpolation problems through the use of *decomposition strategies*. A carefully designed strategy guarantees TI while still providing a rich variety of LRA inequalities in the resulting interpolants.

This work opens the possibility of using efficient proof-based interpolation portfolios in applications requiring TI. While a thorough study on the use of an interpolation portfolio in such applications is out of the scope for this theoretical paper, we verify experimentally that the resulting interpolants can differ in ways that have practical implications in their use in such applications. We show this by using two measures that we believe to be practical indicators of semantic difference on a set of quantifier-free first-order formulas obtained from software model checking.

Related work. The binary Craig interpolation has been extensively studied (see, e.g., [11,23,25,28,30]), and its practical success has motivated a line of research on tree interpolation [5,18,24,27] which we extend here. In particular, we identify five binary Farkas-based LRA interpolation systems from [1,6,23], classify them based on whether they can guarantee TI, and describe in detail a novel, non-trivial approach of adjusting the binary approach from [6] for tree interpolation. We believe that our approach can be applied in the analysis of other conjunctive binary interpolation approaches for LRA, such as the one in [8]. Similarly to us, [7] studies tree interpolation in LRA. Compared to [7], we extend the study by considering four other algorithms and strengthening the existing result. We base the propositional part of our results on the studies in [17,26,29], where tree interpolation is discussed in a purely propositional setting.

We mention here some applications that we believe to be relevant to our current work. In [13] the authors synthesize winning strategies by exploiting tree interpolants computed by the Z3 SMT solver [24]. The state-of-the-art Horn solver Eldarica [19] uses tree interpolation to refine abstraction: it maps each (spurious) counterexample DAG to a tree interpolation problem. Finally, tree interpolants are also used in regression verification of evolving software. See, for example, [2,29], that use tree interpolants to over-approximate functions for incremental verification of program versions.

The paper is organized as follows. We first introduce in Sec. 2 the necessary background on tree interpolation and then, in Sec. 3, provide our main result that the decomposed interpolants guarantee TI. We prove the three smaller, negative results in Sec. 4. In Sec. 5 we show experimentally that the approach produces a range of interpolants not available through existing means, and finally offer conclusions in Sec. 6.

2 Background

Our context is that of SMT (Satisfiability Modulo Theories [3,10]) on quantifier-free formulas in the theory of linear arithmetic over the reals, LRA. A *term* in LRA is either a constant, a variable, or the application of a function symbol in LRA. An LRA *atom* is of the form $t < c$ or $t \leq c$, where t is a term and c is a constant. Given an LRA atom At, we denote by $symb(At)$ the set of variables in At. A *literal* is either an atom At or its negation $\overline{At}$. A *clause* cl is a finite disjunction of literals, and a *formula* in *conjunctive normal form* (CNF) is a conjunction of clauses. We interchangeably interpret a clause as a set of literals and a CNF formula as a set of clauses. We denote by Fla the set of all formulas in CNF. For a formula ϕ, we denote by $\neg\phi$ its negation. We extend the notation $symb$ to CNF formulas, writing $symb(\phi)$ for the set of atoms in ϕ.

A CNF formula ϕ is *satisfiable* if there exists an assignment to its variables so that each clause in ϕ contains a *true* literal. A *resolution refutation* (or *refutation*) of a CNF formula ϕ is a tree labeled with clauses. The root of the tree has the empty clause $\bot$, and the leaves have either *source clauses* appearing directly in ϕ, or *theory clauses* that are tautologies in LRA learned through an unsatisfiable conjunctive query to the LRA solver. The inner nodes are clauses derived by the *resolution rule*

$$\frac{C_1 \vee p \qquad C_2 \vee \overline{p}}{C_1 \vee C_2}$$

where $C_1 \vee p$ and $C_2 \vee \overline{p}$ are the *antecedents*, $C_1 \vee C_2$ the *resolvent*, p is the *pivot* of the resolution step.

The notion of interpolant goes back to Craig's interpolation theorem for first-order logic [9]. In this work we consider approaches where interpolants are constructed from proofs of unsatisfiability.

Definition 1 (binary interpolation). *Given an unsatisfiable CNF formula ϕ partitioned into two disjoint formulas A and B, we denote a* binary interpolation instance *by $(A \,|\, B)$. An* interpolation algorithm *Itp is a procedure that maps an interpolation instance to a formula $I = Itp(A \,|\, B)$ such that (i) $A \implies I$, (ii) $I \implies \neg B$, and (iii) $symb(I) \subseteq symb(A) \cap symb(B)$.*

If I is an interpolant for $(A \,|\, B)$, then $\neg I$ is an interpolant for $(B \,|\, A)$. This interpolant is called *dual interpolant* of $(B \,|\, A)$.

Part of our discussion combines Craig interpolation in propositional logic and LRA. For the propositional part, we use the Pudlák's interpolation algorithm [25], which we treat as an instance of D'Silva et al.'s labeling interpolation system [11]. The approach first constructs the refutation using standard SMT methods. The interpolation works then by labeling each clause with an interpolant starting from the leaf clauses towards the empty clause. The leaf theory clauses are labeled using an LRA interpolation after which the propositional labeling can be applied in a standard way. For lack of space, we refer the reader to Appendix A for details.

We in particular concentrate on *tree interpolants*, generalizations of binary interpolants, obtained from a single refutation.

Definition 2 (weak tree-interpolation property[5]). *Let* $X_1 \wedge \ldots \wedge X_n \wedge Y \wedge Z \implies \bot$. *Let* $I_{X_1}, \ldots, I_{X_n}$ *and* $I_{X_1 \ldots X_n Y}$ *be interpolants for interpolation instances* $(X_1 \mid X_2 \wedge \ldots \wedge X_n \wedge Y \wedge Z)$, …, $(X_n \mid X_1 \wedge \ldots \wedge X_{n-1} \wedge Y \wedge Z)$, *and* $(X_1 \wedge \ldots \wedge X_n \wedge Y \mid Z)$, *respectively. The* $n+2$*-tuple* $(I_{X_1}, \ldots, I_{X_n}, Y, I_{X_1 \ldots X_n Y})$ *has the* weak tree-interpolation property *iff* $I_{X_1} \wedge \ldots \wedge I_{X_n} \wedge Y \implies I_{X_1 \ldots X_n Y}$.

We are now ready to define an instance of the tree interpolation problem.

Definition 3 (tree interpolation instance). *Let* ϕ *be an unsatisfiable SMT formula in CNF,* (V, E) *a directed tree with vertices* V *containing a unique root* $v_r \in V$, *and directed edges* $E \subseteq V \times V$. *Let furthermore* F *be a labeling function* $F : V \longrightarrow Fla$ *that maps vertices* V *to sets of clauses of* ϕ *such that* $\bigwedge_{v \in V} F(v) = \phi$ *and* $F(v) \cap F(w) = \emptyset$ *whenever* $v \neq w$. *We call* $\langle (V, E), F \rangle$ *a* tree interpolation instance.

Let E^* denote the reflexive transitive closure of E. We denote the nodes in the subtree rooted at a node v by $subtree(v) = \{w \mid (w, v) \in E^*\}$ and the complement of the subtree as $\overline{subtree(v)} = V \setminus subtree(v)$. We also extend the function F to sets of nodes as $F(U) = \bigwedge_{v \in U} F(v)$. With this notation we can define the *tree interpolant* for a problem $\langle (V, E), F \rangle$ as follows.

Definition 4 (tree interpolant). *A* tree interpolant *for a tree interpolation instance* $\langle (V, E), F \rangle$ *is a labeling function* $\tau\iota : V \longrightarrow Fla$ *that assigns a formula to every vertex in* V *satisfying the following conditions:*

1. $\tau\iota(v_r) = \bot$,
2. *for all* $v \in V$ *with children* $c_1, \ldots, c_n$, *the* $(n+2)$*-tuple* $(\tau\iota(c_1), \ldots, \tau\iota(c_n), F(v), \tau\iota(v))$ *has the weak tree-interpolation property, i.e.,* $\bigwedge_{i=1}^{n} \tau\iota(c_i) \wedge F(v) \implies \tau\iota(v)$,
3. $\tau\iota(v)$ *uses only the common language of* $subtree(v)$ *and* $\overline{subtree(v)}$, *i.e.,* $symb(\tau\iota(v)) \subseteq symb(F(subtree(v))) \cap symb(F(\overline{subtree(v)}))$.

A tree interpolation algorithm *TItp is a procedure that maps any tree interpolation instance to a tree interpolant* $\tau\iota = TItp(\langle (V, E), F \rangle)$.

We make the following observation that will be central in our discussion in Sec. 3:

Remark 1. Given a binary interpolation algorithm *Itp*, we can construct an algorithm $TItp_{Itp}$ that computes the labels of nodes v by iteratively applying *Itp* on a single resolution refutation for different binary partitionings as

$$\tau\iota(v) = Itp(F(subtree(v)) \mid F(\overline{subtree(v)}))$$

If *Itp* guarantees that for each node v and its children $c_1, \ldots, c_n$ the tuple $(\tau\iota(c_1), \ldots, \tau\iota(c_n), F(v), \tau\iota(v))$ satisfies the weak tree-interpolation property, then the algorithm $TItp_{Itp}$ is guaranteed to produce a tree interpolant, that is, $TItp_{Itp}$ is a tree interpolation algorithm. As a subtle, important consequence,

[5] For example in [7] this is called the *tree interpolation property*.

a certain interpolation algorithm class, called *decomposing Farkas interpolation algorithms* and discussed in Sec. 3.3, needs to be instantiated into actual algorithms using *decomposition strategies* that make the algorithms aware of the tree structure before they can be used as a component of a tree interpolation algorithm.

2.1 Linear Systems

The problem domain in this work is $\mathbb{R}$, the set of real numbers. The (column) vector of n elements is denoted by $\mathbf{v} = (v_1, \ldots, v_n)^\intercal$. The vector of all zeroes is denoted by $\mathbf{0}$.

A linear system S is a conjunction of m inequalities which we treat as a set $S = \{l_i \mid i = 1, \ldots, m\}$ involving the set of n variables $X = \{x_1, \ldots, x_n\}$ such that each l_i is of the form $\sum_j c_{ij} x_j \bowtie b_i$, where $\bowtie \in \{\leq, <\}$, $c_{11}, c_{12}, \ldots, c_{mn}$ are the coefficients of the system, and $b_1, \ldots, b_m$ are constants. We often fix an order for the system and denote it with the matrix notation $C\mathbf{x} \bowtie \mathbf{b}$, where C is the $m \times n$ matrix of coefficients c_{ij}, $\mathbf{x} = (x_1, \ldots, x_n)^\intercal$, and $\mathbf{b} = (b_1, \ldots, b_m)^\intercal$.

For the rest of the paper we use just $\leq$ instead of $\bowtie$. This does not affect the correctness of the proofs presented in this paper but greatly simplifies the presentation. Throughout the paper by system S we refer to a finite set of linear inequalities in the form of

$$\begin{aligned} l_1 \equiv \quad & c_{11}x_1 + c_{12}x_2 + \cdots + c_{1n}x_n \leq b_1 \\ & \vdots \\ l_m \equiv \quad & c_{m1}x_1 + c_{m2}x_2 + \cdots + c_{mn}x_n \leq b_m \end{aligned}$$

Finally for the matrix C and constants $\mathbf{b}$ of system S, and a sub-system $S' \subseteq S$ we use the notations $C_{S'}$ and $\mathbf{b}_{S'}$ to denote the matrix and constants of the sub-system S'. Intuitively $C_{S'}$ and $\mathbf{b}_{S'}$ denote the restrictions of C and $\mathbf{b}$ where only the coordinates corresponding to the subsystem S' are kept. More formally, let $S := C\mathbf{x} \leq \mathbf{b}$ be a system of m linear inequalities, and $S' := ((C_{i_1}), \ldots, (C_{i_k}))^\intercal \mathbf{x} \leq (b_{i_1}, \ldots, b_{i_k})^\intercal$ be a subsystem of S with $k \leq m$ linear inequalities, where C_{i_j} is the ${i_j}^{\text{th}}$ row of C, b_{i_j} the ${i_j}^{\text{th}}$ element of $\mathbf{b}$, and $i_j < i_{j+1}$ for all $1 \leq j \leq k-1$. We denote by $C_{S'}$ the matrix $((C_{i_1}), \ldots, (C_{i_k}))^\intercal$ and by $\mathbf{b}_{S'}$ the vector $(b_{i_1}, \ldots, b_{i_k})^\intercal$.

3 Tree Interpolation for Linear Real Arithmetic

In this section we show our main result, that the decomposing Farkas interpolation algorithm Itp^D guarantees the tree interpolation property. We first introduce a stronger version of tree interpolation property than Def. 2 that will be useful in the proofs and discussion.

Definition 5 (strong tree-interpolation property[6]). *Let* $X_1 \wedge \ldots \wedge X_n \wedge Z \implies \bot$. *Let* $I_{X_1}, \ldots, I_{X_n}$ *and* $I_{X_1 \ldots X_n}$ *be interpolants for interpolation instances* $(X_1 \mid X_2 \wedge \ldots \wedge X_n \wedge Z)$, …, $(X_n \mid X_1 \wedge \ldots \wedge X_{n-1} \wedge Z)$, *and* $(X_1 \wedge \ldots \wedge X_n \mid Z)$, *respectively. The* $n+1$*-tuple* $(I_{X_1}, \ldots, I_{X_n}, I_{X_1 \ldots X_n})$ *has the strong tree-interpolation property iff* $(I_{X_1} \wedge \ldots \wedge I_{X_n}) \implies I_{X_1 \ldots X_n}$.

It is easy to show that if a binary interpolation algorithm guarantees the strong tree-interpolation property, it also guarantees the weak tree-interpolation property because $Y \implies I_Y$. We use the term *tree interpolation property* without qualifiers when we refer to both its weak and strong versions.

Algorithms for solving linear systems in SMT solvers make use of the Simplex algorithm [12] and are based on the Farkas' lemma.

Lemma 1 (Farkas' lemma). *Let* $C \in \mathbb{R}^{m \times n}$. $C\mathbf{x} \leq \mathbf{b}$ *is unsatisfiable if and only if there exists a vector* $\mathbf{f} \geq \mathbf{0}$ *such that* $\mathbf{f}^\intercal C = \mathbf{0}$ *and* $\mathbf{f}^\intercal \mathbf{b} < 0$.

We refer to the vector $\mathbf{f}$ as the vector of Farkas coefficients. Given this vector it is possible to immediately compute two interpolants:

Definition 6 (Farkas and dual Farkas interpolants in LRA [23]). *Given an interpolation instance* $(A \mid B)$ *over a linear system* $S = C\mathbf{x} \leq \mathbf{b}$ *and its Farkas coefficients* $\mathbf{f}$, *the* Farkas interpolant *for* $(A \mid B)$ *is the inequality*

$$I^F := \mathbf{f}_A^\intercal (C_A \mathbf{x} - \mathbf{b}_A) \leq 0,$$

and the dual Farkas interpolant *for* $(A \mid B)$ *is a negation of the Farkas interpolant for* $(B \mid A)$*:*

$$\overline{I^F} := \neg\, (\mathbf{f}_B^\intercal (C_B \mathbf{x} - \mathbf{b}_B) \leq 0),$$

where $\mathbf{f}_A$ *and* $\mathbf{f}_B$ *are the restrictions of* $\mathbf{f}$ *to the subsystems* A *and* B, *respectively.*

Recently [6] introduced an algorithm to gain more control over the strength of LRA interpolants. The underlying idea is to not directly sum the inequalities in A-part, but instead split the sum into sub-sums. This yields an interpolant that is a conjunction (decomposition) of possibly more than one component of the Farkas interpolant. In the following, we formally define what type of decomposition is suitable for interpolation instances.

Definition 7 (proper decomposition for interpolation [6]). *Let* $S = C\mathbf{x} \leq \mathbf{b}$ *be a system of linear inequalities over a set of variables* $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ *and let* $L \subseteq X$. *Let* $\mathbf{w} \geq \mathbf{0}$ *be a vector such that all variables from* L *are eliminated in* $\mathbf{w}^\intercal C\mathbf{x}$. *We say that a set of vectors* $\mathrm{Dec}(\mathbf{w}^\intercal C\mathbf{x}, L)$ *is a* proper decomposition for interpolation *if it forms a decomposition of* $\mathbf{w}$, *i.e.,*

$$\mathbf{w} = \sum_{\mathbf{v} \in \mathrm{Dec}(\mathbf{w}^\intercal C\mathbf{x}, L)} \mathbf{v};$$

and for all $\mathbf{v} \in \mathrm{Dec}(\mathbf{w}^\intercal C\mathbf{x}, L)$, *(i)* $\mathbf{v} \geq \mathbf{0}$ *and (ii) all variables from* L *are eliminated in* $\mathbf{v}^\intercal C\mathbf{x}$.

[6] This property appears in the literature under names *generalized simultaneous abstraction* [17] and *symmetric interpolation* [21].

Definition 8 (decomposed interpolants and their duals [6]). *Let* $(A \mid B)$ *be an interpolation instance in* LRA*,* L_A *the local variables of* A *(those not appearing in* B*), and* $\mathbf{f}$ *the vector of Farkas coefficients of the system. Let* $\{\mathbf{f}_1, \ldots, \mathbf{f}_k\} = \mathrm{Dec}(\mathbf{f_A}^\intercal C_A \mathbf{x}, L_A)$ *be a proper decomposition. Then*

$$I^D = \bigwedge_{i=1}^{k} \mathbf{f}_i{}^\intercal (C_A \mathbf{x} - \mathbf{b}_A) \leq 0$$

is a decomposed interpolant *of size* k *for* $(A \mid B)$*.*

Similarly, for subsystem B *with its local variables* L_B*, if* $\{\mathbf{f}'_1, \ldots, \mathbf{f}'_m\} = \mathrm{Dec}(\mathbf{f_B}^\intercal C_B \mathbf{x}, L_B)$ *then the* dual decomposed interpolant *for* $(A \mid B)$ *is the negation of the decomposed interpolant for* $(B \mid A)$*:*

$$\overline{I^D} \equiv \neg \left(\bigwedge_{j=1}^{m} \mathbf{f}'_j{}^\intercal (C_B \mathbf{x} - \mathbf{b}_B) \leq 0 \right)$$

When the set of local variables is clear from the context we omit the second argument of Dec. Note that *trivial* proper decomposition of size 1 always exists: it is the Farkas interpolant.

Next, we illustrate the key difference between Farkas (I^F) and decomposed Farkas (I^D) interpolants by an example that will serve as our running example.

Example 1. Consider S as the unsatisfiable conjunction of linear inequalities:

$$\left.\begin{array}{r} x_1 + x_2 \leq 0 \\ -x_1 + x_3 \leq 0 \end{array}\right\} X_1$$
$$\left.\begin{array}{r} x_1 + x_4 \leq 0 \\ -x_1 + x_5 \leq 0 \end{array}\right\} X_2$$
$$\left.\begin{array}{r} -x_2 - x_5 + x_6 \leq 0 \end{array}\right\} Y$$
$$\left.\begin{array}{r} -x_3 - x_4 - x_6 \leq -1 \end{array}\right\} Z,$$

where we will denote the inequalities by $l_1, \ldots, l_6$, respectively. Consider the following disjoint sets $X_1 = \{l_1, l_2\}$, $X_2 = \{l_3, l_4\}$, $Y = \{l_5\}$, and $Z = \{l_6\}$ as shown above. The unsatisfiability of S is witnessed by the Farkas coefficients $\mathbf{f}^\intercal = (1, 1, 1, 1, 1, 1)$. Let Itp^F be the Farkas interpolation algorithm. Consider interpolation instance $(X_1 \wedge X_2 \wedge Y \mid Z)$, with $\mathbf{f}^\intercal_{XY} = (1, 1, 1, 1, 1)$ that eliminate x_1, x_2, x_5, the local variables of $X_1 \wedge X_2 \wedge Y$ with respect to the rest of S. The interpolant $Itp^F(X_1 \wedge X_2 \wedge Y \mid Z)$ is $I^F_{XY} = x_3 + x_4 + x_6 \leq 0$. Let Itp^D be the decomposing Farkas interpolation algorithm. The interpolation instance $(X_1 \wedge X_2 \wedge Y \mid Z)$ admits a decomposition of $\mathbf{f}^\intercal_{XY} C_{XY} \mathbf{x}$ as $D = \{(1, 0, 0, 1, 1)^\intercal, (0, 1, 1, 0, 0)^\intercal\}$ that eliminates the local variables x_1, x_2, x_5. The interpolant $Itp^D(X_1 \wedge X_2 \wedge Y \mid Z) = I^D_{XY} = x_6 \leq 0 \wedge x_3 + x_4 \leq 0$ computed with respect to this decomposition contains two conjuncts and differs from the Farkas interpolant.

3.1 Proper Labeling

We rely on resolution refutation that incorporates theory lemmas that are created by the theory solver and get propagated to the SAT solver. As theory solver provides a separate proof for each theory clause, we can compute an LRA interpolant for the negation of each theory clause. Once these are obtained, the final interpolant is computed using Pudlák's propositional interpolation algorithm [25].

The binary interpolation algorithms for linear real arithmetic discussed above require that the theory atoms be placed into exactly one partition. The origin of the partition is, however, the CNF partition, where it is common that atoms (as opposed to clauses) belong to several partitions. This can lead to subtle problems in the definitions and eventually implementations. We first illustrate the problem with an example before defining *proper labeling* in Def. 9 that we will use for resolving the problem.

Example 2. Consider the sets of clauses $X = \{a \leq b, (\overline{a \leq c}) \vee x\}$, $Y = \{b \leq c, (\overline{a \leq c}) \vee y\}$, and $Z = \{\overline{x} \vee \overline{y}\}$, and the theory clause $cl := (\overline{a \leq b}) \vee (\overline{b \leq c}) \vee a \leq c$ required for the refutation of $X \wedge Y \wedge Z$. The atom $a \leq c$ can be considered a part of both partitions X and Y when computing an interpolant for some binary partitioning of the theory clause cl. However, the strong TI is not guaranteed if we change the partition of $a \leq c$ between different binary interpolation instances. For example, placing $a \leq c$ in Y while interpolating $(X \mid Y \wedge Z)$ for cl yields the Farkas theory interpolant $I^F_X = a \leq b$. Placing $a \leq c$ in X while interpolating $(Y \mid X \wedge Z)$ and $(X \wedge Y \mid Z)$ for cl gives $I^F_Y = b \leq c$, and $I^F_{XY} = \bot$. Clearly using these interpolants violates the strong TI since $a \leq b \wedge b \leq c \implies \bot$.

We define a general version of this *proper labeling* of the theory clauses and argue that a fixed proper labeling must be used for a sequence of binary interpolation problems if tree interpolation property is to be guaranteed.

Definition 9 (proper labeling). *Let $X_1, \ldots, X_n$ be sets of clauses. We say that $X_1 \wedge \ldots \wedge X_n$ is a partitioned CNF formula F and we say that a function from atoms of F to partitions $PL : Atoms(F) \to \{1, \ldots, n\}$ is a* proper labeling *if for each atom At it holds that $PL(At) = i$ implies that there is a clause in X_i containing At (or its negation).*

Proper labeling is used in a resolution refutation of a partitioned CNF formula to determine the partitioning of theory clauses and consequently the input for theory interpolation algorithms. In the rest of the text we are going to assume that a refutation of a partitioned CNF formula always comes with some fixed proper labeling and we say that the refutation is *properly labeled*.

3.2 Tree Interpolation Property in Farkas Interpolation Algorithm

For the Farkas interpolation algorithm we first state and prove a simplified version of Def. 5 limited to three partitions and then generalize the result for an arbitrary number of partitions by an iterative application of Theorem 1.

Theorem 1 (strong TI in Farkas interpolation). *Let $X \wedge Y \wedge Z$ be an unsatisfiable partitioned CNF formula in* LRA *and let $\mathbb{P}$ be its properly labeled resolution refutation. Let Itp^{P+F} denote the interpolation algorithm that uses Pudlák's algorithm for the propositional part and Farkas algorithm for the theory clauses. Let I_X, I_Y, and I_{XY} be the binary interpolants $Itp^{P+F}(X \,|\, Y \wedge Z)$, $Itp^{P+F}(Y \,|\, X \wedge Z)$, and $Itp^{P+F}(X \wedge Y \,|\, Z)$, respectively. Then $(I_X \wedge I_Y) \implies I_{XY}$.*

Proof. We show using structural induction that for any clause cl in the refutation $\mathbb{P}$ and for all possible partial interpolants, property $(I_X \wedge I_Y) \implies I_{XY}$ holds.

Since a leaf in $\mathbb{P}$ could either be a source clause or a theory clause, we study each case separately. The proof has two steps: base case, and inductive step. The base case itself consists of two steps, depending on the nature of leaf clauses. Both the inductive step and the base case for source clauses are shown in [29] and are therefore given here as a sketch. Their full proofs are in Appendix A.

Base case (source clause, *sketch*) We can show one-by-one for the three cases $cl \in X, cl \in Y$, and $cl \in Z$ that the strong tree-interpolation property holds. See proof of Lemma 2 in Appendix A.

Base case (theory clause). Let $cl \equiv \bar{\ell_1} \vee \cdots \vee \bar{\ell_n}$ from $\mathbb{P}$ be a theory clause in LRA, S the system of linear inequalities corresponding to $\neg cl$, and $\mathbf{f}$ the vector of Farkas coefficients witnessing the unsatisfiability of S.

As refutation $\mathbb{P}$ is properly labeled (Def. 9), each literal ℓ_i is uniquely assigned to either X, Y or Z. The LRA interpolants for the theory clause computed for the binary interpolation instances $(X \,|\, Y \wedge Z)$, $(Y \,|\, X \wedge Z)$, and $(X \wedge Y \,|\, Z)$ are thus $I_X^F = \mathbf{f}_X^\intercal(C_X\mathbf{x} - \mathbf{b}_X) \leq 0$, $I_Y^F = \mathbf{f}_Y^\intercal(C_Y\mathbf{x} - \mathbf{b}_Y) \leq 0$, and $I_{XY}^F = \mathbf{f}_{XY}^\intercal(C_{XY}\mathbf{x} - \mathbf{b}_{XY}) \leq 0$, respectively, where we denote by XY the subsystem corresponding to $X \wedge Y$. By construction, we have $\mathbf{f}_{XY}^\intercal(C_{XY}\mathbf{x} - \mathbf{b}_{XY}) = \mathbf{f}_X^\intercal(C_X\mathbf{x} - \mathbf{b}_X) + \mathbf{f}_Y^\intercal(C_Y\mathbf{x} - \mathbf{b}_Y)$. It follows that if $\mathbf{f}_X^\intercal(C_X\mathbf{x} - \mathbf{b}_X) \leq 0$ and $\mathbf{f}_Y^\intercal(C_Y\mathbf{x} - \mathbf{b}_Y) \leq 0$, then also $\mathbf{f}_{XY}^\intercal(C_{XY}\mathbf{x} - \mathbf{b}_{XY}) \leq 0$, which is exactly the desired result that $I_X^F \wedge I_Y^F \implies I_{XY}^F$.

Inductive step (inner node, *sketch*). By case analysis on the different binary interpolations in the nodes of the refutation $\mathbb{P}$ it is possible to show that each *partial interpolant* associated with the resolvent has TI (see proof of Lemma 3 in Appendix A). □

The result of Theorem 1 can be generalized to prove that Itp^{P+F} guarantees strong (and consequently weak) TI.

Theorem 2 (generalizing strong TI in Farkas interpolation). *Let $X_1 \wedge \ldots \wedge X_n \wedge Z$, $n \geq 2$, be an unsatisfiable partitioned CNF formula in* LRA *and let $\mathbb{P}$ be its properly labeled resolution refutation. Let Itp^{P+F} denote the interpolation algorithm that uses Pudlák's algorithm for the propositional part and Farkas algorithm for the theory clauses. Let I_{X_1}, …, I_{X_n}, and $I_{X_1 \ldots X_n}$ be the binary interpolants $Itp^{P+F}(X_1 \,|\, X_2 \wedge \ldots \wedge X_n \wedge Z)$, …, $Itp^{P+F}(X_n \,|\, X_1 \wedge \ldots \wedge X_{n-1} \wedge Z)$ and $Itp^{P+F}(X_1 \wedge \ldots \wedge X_n \,|\, Z)$, respectively. Then $(I_{X_1} \wedge \ldots \wedge I_{X_n}) \implies I_{X_1 \ldots X_n}$, i.e., the tuple $(I_{X_1}, \ldots, I_{X_n}, I_{X_1 \ldots X_n})$ has the strong tree-interpolation property.*

Proof. We prove the theorem by induction. For the base case with $n = 2$ we may apply Theorem 1. Assume now that $n \geq 3$, and the theorem holds for $n-1$. Using the induction hypothesis for the tuple $(X_1, \ldots, X_{n-1}, X_n \wedge Z)$ we get that

$$I_{X_1} \wedge \ldots \wedge I_{X_{n-1}} \implies I_{X_1 \ldots X_{n-1}}.$$

By applying Theorem 1 again for $(X_1 \wedge \ldots \wedge X_{n-1}, X_n, Z)$ we obtain

$$I_{X_1 \ldots X_{n-1}} \wedge I_{X_n} \implies I_{X_1 \ldots X_n}.$$

Combining these two implications yields the desired result $I_{X_1} \wedge \ldots \wedge I_{X_n} \implies I_{X_1 \ldots X_n}$. □

By Theorem 2, the interpolation algorithm Itp^{P+F} guarantees the strong TI. We can use the technique described in Remark 1 to obtain a tree interpolation algorithm $TItp_{Itp^{P+F}}$ for computation of tree interpolants.

Corollary 1. *$TItp_{Itp^{P+F}}$ is a tree interpolation algorithm, that is, it computes tree interpolants.*

Note that while the result that $TItp_{Itp^{P+F}}$ is a tree interpolation algorithm is known from [7], our result that Itp^{P+F} guarantees the strong TI is new to the best of our knowledge.

3.3 A Tree interpolation Algorithm based on Decomposing Farkas Interpolation

In this section we consider the decomposing Farkas interpolation algorithm of [6], and show that if the decompositions satisfy a certain property, then the algorithm guarantees the tree interpolation property. We also show that if the condition is not satisfied, the tree interpolation property is not guaranteed.

A central difference between decomposing Farkas interpolation algorithm and Farkas interpolation algorithm is that the former is a template rather than a concrete algorithm. In practice this means that the algorithm is parameterized by the decomposition of the (restricted) vector of Farkas coefficients and can yield different interpolants for different decompositions. For tree interpolation one wants to relate interpolants computed by multiple binary interpolation instances over the same proof. Therefore also the decompositions need to respect this relation for the binary interpolation instances. We first show that tree interpolation property is not guaranteed in general for the decomposed interpolants, and then define a constraint on the decompositions that guarantees the tree interpolation property.

Example 3. Consider our running example of Ex. 1 and let $X := X_1 \wedge X_2$. Using the decomposing Farkas interpolation algorithm Itp^D, the interpolation instance $(X \,|\, Y \wedge Z)$ admits different non-trivial decompositions of the restricted vector of Farkas coefficients $\mathbf{f}_X^\intercal = (1, 1, 1, 1)$, for example $D_1 = \{(1, 0, 0, 1)^\intercal, (0, 1, 1, 0)^\intercal\}$

and $D_2 = \{(1,1,0,0)^\intercal, (0,0,1,1)^\intercal\}$. Both D_1 and D_2 successfully eliminate the single X-local variable x_1, as required. The outcome of $Itp^D(X \mid Y \wedge Z)$ using D_1 is $I_X^1 = x_2 + x_5 \leq 0 \wedge x_3 + x_4 \leq 0$. When using D_2, the resulting interpolant is $I_X^2 = x_2 + x_3 \leq 0 \wedge x_4 + x_5 \leq 0$. Consider $(X \wedge Y \mid Z)$ that admits only a single non-trivial decomposition of $\mathbf{f}_{XY}^\intercal = (1,1,1,1,1)$ that eliminates XY-local variables x_1, x_2, x_5: $D_3 = \{(1,0,0,1,1)^\intercal, (0,1,1,0,0)^\intercal\}$. The interpolant computed with respect to this decomposition is $Itp^D(X \wedge Y \mid Z) = I_{XY} = x_6 \leq 0 \wedge x_3 + x_4 \leq 0$. Consider $(Y \mid X \wedge Z)$ where there is no opportunity for decomposition since its first part consists of the single inequality l_5. The computed interpolant is $Itp^D(Y \mid X \wedge Z) = I_Y = -x_2 - x_5 + x_6 \leq 0$. We can easily see that the strong tree-interpolation property (Def. 5) is satisfied for I_X^1: $I_X^1 \wedge I_Y \implies I_{XY}$. However, the property is not satisfied for I_X^2, since $I_X^2 \wedge I_Y \not\implies I_{XY}$. Since in this example $I_Y = Y$, the same is true for the weak tree-interpolation property (Def. 2).

Intuitively, the tree interpolation property might not hold when the subsystem's decomposition (D_2) does not agree with its supersystem's decomposition (D_3) (when restricted to the subsystem). More generally, the inequalities resulting from (the restriction of) supersystem's decomposition must be logically covered by the inequalities of subsystem's decomposition. The following *monotonicity* condition captures this formally.

Definition 10 (monotonic decompositions). *Let $S = C\mathbf{x} \leq \mathbf{b}$ be an unsatisfiable system of linear inequalities and let $(A_1 \mid B_1), \ldots, (A_n \mid B_n)$ be a set of binary interpolation problems over S. Let Itp^D denote the decomposing Farkas interpolation algorithm (Def. 8). We say that Itp^D uses* monotonic decompositions *if whenever $A_i \subseteq A_j$, then for all vectors $\mathbf{w} \in \mathrm{Dec}(\mathbf{f}_{A_j}^\intercal C_{A_j}\mathbf{x}, L_{A_j})$ there exists $U \subseteq \mathrm{Dec}(\mathbf{f}_{A_i}^\intercal C_{A_i}\mathbf{x}, L_{A_i})$ such that $\sum_{\mathbf{u} \in U} \mathbf{u}^\intercal (C_{A_i}\mathbf{x} - \mathbf{b}_{A_i}) \leq 0 \implies \mathbf{w}_{A_i}^\intercal (C_{A_i}\mathbf{x} - \mathbf{b}_{A_i}) \leq 0$, where $\mathbf{w}_{A_i}$ is the restriction of $\mathbf{w}$ to the subsystem A_i.*

Now we can proceed to prove that Itp^D can guarantee the tree interpolation property.

Theorem 3 (strong TI in decomposing Farkas interpolation). *Let $X \wedge Y \wedge Z$ be an unsatisfiable partitioned CNF formula in* LRA *and let $\mathbb{P}$ be its properly labeled resolution refutation. Let Itp^{P+D} denote the interpolation algorithm that uses Pudlák's algorithm for the propositional part and decomposing Farkas algorithm for the theory clauses. Let I_X, I_Y, and I_{XY} be the binary interpolants $Itp^{P+D}(X \mid Y \wedge Z)$, $Itp^{P+D}(Y \mid X \wedge Z)$, and $Itp^{P+D}(X \wedge Y \mid Z)$, respectively. If Itp^D uses monotonic decompositions for every theory clause in $\mathbb{P}$ then $(I_X \wedge I_Y) \implies I_{XY}$.*

Proof (by structural induction). We only show the proof of the implication for a leaf with a theory clause. In the remaining cases the proof is the same as that of Theorem 1.

Let cl be a theory clause in LRA and let $S = C\mathbf{x} \leq \mathbf{b}$ be the system of linear inequalities corresponding to $\neg cl$. Let $\mathbf{f}$ denote the Farkas coefficients

witnessing the unsatisfiability of S. Note that the proper labeling of $\mathbb{P}$ (recall Def. 9) partitions cl into three disjoint sets of literals X, Y, Z. Let Dec_X, Dec_Y and Dec_{XY} denote the decompositions $\mathrm{Dec}(\mathbf{f}_X^\intercal C_X\mathbf{x}, L_X)$, $\mathrm{Dec}(\mathbf{f}_Y^\intercal C_Y\mathbf{x}, L_Y)$, and $\mathrm{Dec}(\mathbf{f}_{XY}^\intercal C_{XY}\mathbf{x}, L_{XY})$, where XY denotes the subsystem $X \wedge Y$. We need to prove that

$$\left(\bigwedge_{\mathbf{p}\in Dec_X} \mathbf{p}^\intercal(C_X\mathbf{x}-\mathbf{b}_X)\le 0\right)\wedge\left(\bigwedge_{\mathbf{q}\in Dec_Y} \mathbf{q}^\intercal(C_Y\mathbf{x}-\mathbf{b}_Y)\le 0\right)\implies \left(\bigwedge_{\mathbf{r}\in Dec_{XY}} \mathbf{r}^\intercal(C_{XY}\mathbf{x}-\mathbf{b}_{XY})\le 0\right)$$

It is enough to fix $\mathbf{r}\in Dec_{XY}$ and prove that $\mathbf{r}^\intercal(C_{XY}\mathbf{x}-\mathbf{b}_{XY})\le 0$ is implied by the antecedent. As the subsystem XY consists of two disjoint subsystems X and Y, it holds that $\mathbf{r}^\intercal = (\mathbf{r}_X^\intercal\ \mathbf{r}_Y^\intercal)$, $\mathbf{b}_{XY}^\intercal = (\mathbf{b}_X^\intercal\ \mathbf{b}_Y^\intercal)$, $C_{XY}^\intercal = (C_X^\intercal\ C_Y^\intercal)$ and $\mathbf{r}_X^\intercal(C_X\mathbf{x}-\mathbf{b}_X)+\mathbf{r}_Y^\intercal(C_Y\mathbf{x}-\mathbf{b}_Y) = \mathbf{r}^\intercal(C_{XY}\mathbf{x}-\mathbf{b}_{XY})$. Consequently, it is enough to prove that

$$\bigwedge_{\mathbf{p}\in Dec_X} \mathbf{p}^\intercal(C_X\mathbf{x}-\mathbf{b}_X)\le 0 \implies \mathbf{r}_X^\intercal(C_X\mathbf{x}-\mathbf{b}_X)\le 0$$

and

$$\bigwedge_{\mathbf{q}\in Dec_Y} \mathbf{q}^\intercal(C_Y\mathbf{x}-\mathbf{b}_Y)\le 0 \implies \mathbf{r}_Y^\intercal(C_Y\mathbf{x}-\mathbf{b}_Y)\le 0$$

We only show how to prove the first implication, the second one is analogous. According to our assumption, Itp^D uses monotonic decompositions. Hence for $\mathbf{r}\in Dec_{XY}$ there exists $U\subseteq Dec_X$ such that $\sum_{\mathbf{u}\in U}\mathbf{u}^\intercal(C_X\mathbf{x}-\mathbf{b}_X)\le 0 \implies \mathbf{r}_X^\intercal(C_X\mathbf{x}-\mathbf{b}_X)\le 0$. This is exactly what we need to finish the proof since

$$\bigwedge_{\mathbf{p}\in Dec_X} \mathbf{p}^\intercal(C_X\mathbf{x}-\mathbf{b}_X)\le 0 \implies \bigwedge_{\mathbf{u}\in U} \mathbf{u}^\intercal(C_X\mathbf{x}-\mathbf{b}_X)\le 0 \implies \sum_{\mathbf{u}\in U} \mathbf{u}^\intercal(C_X\mathbf{x}-\mathbf{b}_X)\le 0 \implies \mathbf{r}_X^\intercal(C_X\mathbf{x}-\mathbf{b}_X)\le 0.$$

□

We first generalize the result of Theorem 3 to an arbitrary number of partitions, then discuss how monotonic decompositions can be achieved, and finally show that tree interpolants can be computed using the decomposing Farkas interpolation algorithm.

Theorem 4 (generalizing strong TI in decomposing Farkas interpolation). *Let $X_1 \wedge \ldots \wedge X_n \wedge Z$, $n \ge 2$, be an unsatisfiable partitioned CNF formula in* LRA *and let $\mathbb{P}$ be its properly labeled refutation. Let Itp^{P+D} denote the interpolation algorithm that uses Pudlák's algorithm for the propositional part and decomposing Farkas algorithm for the theory clauses. Let I_{X_1}, ..., I_{X_n}, and $I_{X_1\ldots X_n}$ be the binary interpolants $Itp^{P+D}(X_1 \,|\, X_2 \wedge \ldots \wedge X_n \wedge Z)$,*

..., $Itp^{P+D}(X_n \mid X_1 \wedge \ldots \wedge X_{n-1} \wedge Z)$ and $Itp^{P+D}(X_1 \wedge \ldots \wedge X_n \mid Z)$, respectively. If Itp^D uses monotonic decompositions for every theory clause in $\mathbb{P}$, then $(I_{X_1} \wedge \ldots \wedge I_{X_n}) \implies I_{X_1 \ldots X_n}$, i.e., the tuple $(I_{X_1}, \ldots, I_{X_n}, I_{X_1 \ldots X_n})$ has the strong tree-interpolation property.

Proof. The proof is done by induction, the same way as the proof of Theorem 2, here relying on Theorem 3. □

To show how monotonic decompositions can be achieved, we first introduce a notion of *decomposition strategy* that determines the decompositions used for the related interpolation instances.

Definition 11 (decomposition strategy). *Let S be an unsatisfiable set of inequalities, $\mathbf{f}$ its witnessing vector of Farkas coefficients, and $\langle (V, E), F \rangle$ the related tree interpolation instance. The tree interpolation instance defines a set of binary interpolation instances $(F(subtree(v)) \mid F(\overline{subtree(v)}))$ for each $v \in V$. A* decomposition strategy *σ assigns to each vertex $v \in V$ some decomposition $\mathrm{Dec}(\mathbf{f}_{S_v}^\intercal C_{S_v} \mathbf{x}, L_{S_v})$, where $S_v = F(subtree(v))$. We denote the decomposing Farkas interpolation algorithm using strategy σ as $Itp^D(\sigma)$.*

An example of a decomposition strategy that guarantees monotonic decompositions is a *gradual* decomposition. The idea is to first decompose the larger subsystem and then, instead of computing independent decompositions for its subsystems, to decompose only elements of the decomposition of the larger system.

Definition 12 (gradual decomposition). *Given an unsatisfiable set of inequalities S, its witnessing vector of Farkas coefficients $\mathbf{f}$, and a tree interpolation instance $\langle (V, E), F \rangle$, a* gradual decomposition GDec *is a decomposition strategy defined inductively on $\langle (V, E), F \rangle$ from root to leaves as*

1. $\mathrm{GDec}(v_r) = \{\mathbf{f}\}$ *for root v_r,*
2. *otherwise*

$$\mathrm{GDec}(v) = \bigcup_{\mathbf{w} \in \mathrm{GDec}(par(v))} \mathrm{Dec}(\mathbf{w}_{S_v}^\intercal C_{S_v} \mathbf{x}, L(S_v)),$$

where $par(v)$ is the (unique) parent of v, and $S_v = F(subtree(v))$.

Intuitively, the gradual decomposition in a given node v decomposes each element $\mathbf{w}$ of the v's parent's decomposition separately, instead of independently decomposing $\mathbf{f}_{S_v}$, the restriction of the Farkas coefficients to the subsystem of v's subtree. Figure 1 compares the gradual decomposition and independent decomposition on the system from Ex. 3. It shows the tree structure of the three partitions X, Y, Z, and possible decompositions of the vector of Farkas coefficients for the corresponding binary interpolation problems $(X \mid Y \wedge Z)$, $(X \wedge Y \mid Z)$ and $(X \wedge Y \wedge Z \mid \top)$. The solid gray arrows labeled with a subsystem represent the restriction to that subsystem and the dashed arrows represent decomposition.

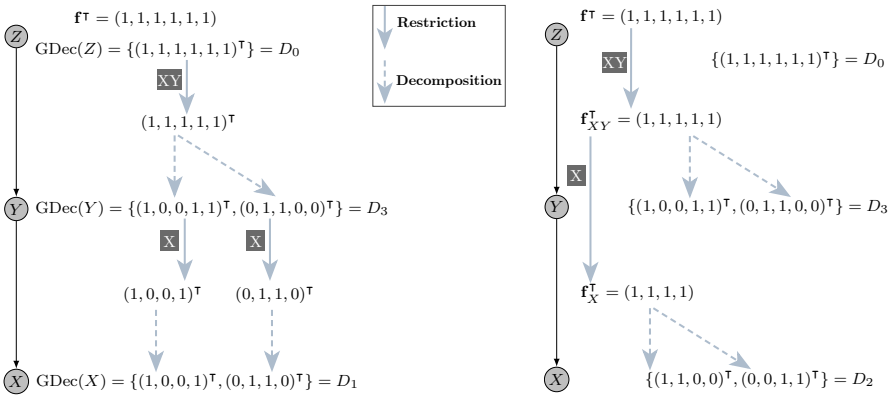

Fig. 1: A gradual decomposition (*left*) versus an independent decomposition (*right*).

The decompositions corresponding to vertices Z and Y (D_0 and D_3) are the same in both cases. The difference manifests when computing the decomposition corresponding to vertex X. On the *left*, gradual decomposition ensures that the decomposition D_1 agrees with D_3 by trying to decompose each (restricted) element of D_3 separately. In this case no further decomposition is possible, thus D_1 is equal to D_3 restricted to the subsystem X. On the *right*, if gradual decomposition is not used, then the decomposition corresponding to vertex X does not take into account what happens at X's parent Y and independently decomposes the restricted vector of Farkas coefficients $\mathbf{f}_X$. This can result in a different decomposition D_2 which, however, violates the *monotonicity* condition from Def. 10 with respect to the decomposition D_3.

Theorem 5. *The algorithm $TItp_{Itp^{P+D}(\mathrm{GDec})}$, where decomposing Farkas interpolation algorithm uses gradual decomposition, is a tree interpolation algorithm, that is, it computes tree interpolants.*

Proof. Using the idea described in Remark 1, given a tree interpolation instance from Def. 3 and a properly labeled refutation $\mathbb{P}$ of $\bigwedge_{v\in V} F(V)$, we define

$$\tau\iota(v) := Itp^{P+D}(\mathrm{GDec})(F(subtree(v)) \,|\, F(\overline{subtree(v)})).$$

Note that the proper labeling of $\mathbb{P}$ recreates the tree-structured partitioning of every theory clause in $\mathbb{P}$, which is required by the definition of the gradual decomposition.

The first and third conditions of tree interpolant automatically follow from our definition of $\tau\iota$. The second condition follows from Theorem 4 since gradual decomposition GDec ensures that the decompositions are monotonic. To see this, recall from the definition of monotonic decomposition (Def. 10) that for each $v \in V$, its parent p, and for each $\mathbf{w} \in \mathrm{GDec}(p)$, there must exist $U \subseteq \mathrm{GDec}(v)$ such that $\sum_{\mathbf{u}\in U} \mathbf{u}^\intercal(C_{S_v}\mathbf{x} - \mathbf{b}_{S_v}) \leq 0 \implies \mathbf{w}_{S_v}^\intercal(C_{S_v}\mathbf{x} - \mathbf{b}_{S_v}) \leq 0$. From the definition of GDec it follows that $U = \mathrm{Dec}(\mathbf{w}_{S_v}^\intercal C_{S_v}\mathbf{x}, L(S_v))$ is the witnessing subset of $\mathrm{GDec}(v)$. □

Note that gradual decomposition is not a concrete strategy, but rather a strategy scheme. It leaves freedom for choosing a decomposition in a particular vertex as long as the decomposition respects the parent's decomposition. One particular instance of a gradual decomposition is *trivial* gradual decomposition *Triv* that always uses the trivial decomposition of size 1. Since trivial decompositions result in Farkas interpolants, it follows that $TItp_{Itp^{P+D}(Triv)} \equiv TItp_{Itp^{P+F}}$.

4 Negative Results for the Algorithms for Flexible, Dual Farkas and Dual Decomposed Interpolation

We prove three smaller, negative results on binary interpolation algorithms that cannot be used as a basis of a tree interpolation algorithm. In particular, we show that the binary interpolation algorithms $\overline{Itp^F}$ and $\overline{Itp^D}$ discussed in Sec. 3, and an algorithm from [1] that we denote by $Itp^{(\alpha)}$, do not guarantee TI.

We first formally define the *flexible* interpolants from [1] in our notation.

Definition 13 (flexible Farkas interpolant [1]). *Let $(A \mid B)$ be an interpolation instance from a system $C\mathbf{x} \leq \mathbf{b}$. Then the interpolants I^F and $\overline{I^F}$ are, respectively, $\mathbf{f}_A^\intercal(C_A\mathbf{x} - \mathbf{b}_A) \leq 0$ and $\mathbf{f}_B^\intercal(C_B\mathbf{x} - \mathbf{b}_B) > 0$. The* flexible Farkas interpolant $I^{(\alpha)}$ *is defined as* $\mathbf{f}_A^\intercal C_A\mathbf{x} + \mathbf{f}_B^\intercal\mathbf{b}_B - \alpha\mathbf{f}_{AB}^\intercal\mathbf{b}_{AB} \leq 0$ *where* $0 < \alpha \leq 1$.

Flexible interpolants are useful in practice as they provide a more fine-grained approach than the Farkas and dual Farkas algorithms.[7] However, they cannot be used in general as a basis for a tree interpolation algorithm:

Theorem 6. *The flexible Farkas interpolation algorithm $Itp^{(\alpha)}$ for $0 < \alpha < 1$ does not guarantee the strong nor the weak tree-interpolation property.*

Proof. Consider our running example from Ex. 1 and four binary interpolation instances $(X_1 \mid X_2 \wedge Y \wedge Z)$, $(X_2 \mid X_1 \wedge Y \wedge Z)$, $(Y \mid X_1 \wedge X_2 \wedge Z)$, and $(X_1 \wedge X_2 \wedge Y \mid Z)$. Let I_{X_1}, I_{X_2}, I_Y, and I_{XY} denote the interpolants from these interpolation instances. The strong tree-interpolation property is formulated as

$$I_{X_1} \wedge I_{X_2} \wedge I_Y \implies I_{XY} \tag{1}$$

and the weak tree-interpolation property is formulated as

$$I_{X_1} \wedge I_{X_2} \wedge Y \implies I_{XY} \tag{2}$$

.

The flexible Farkas interpolants for the interpolation instance are

$$I_{X_1}^{(\alpha)} = (x_2 + x_3 \leq 1 - \alpha) \qquad I_{X_2}^{(\alpha)} = (x_4 + x_5 \leq 1 - \alpha)$$
$$I_Y^{(\alpha)} = (-x_2 - x_5 + x_6 \leq 1 - \alpha) \qquad I_{XY}^{(\alpha)} = (x_3 + x_4 + x_6 \leq 1 - \alpha)$$

The implications of Eq. (1) and Eq. (2) are both falsified with assignment $x_2 \mapsto 0, x_3 \mapsto 1 - \alpha, x_4 \mapsto 1 - \alpha, x_5 \mapsto 0, x_6 \mapsto 0$ for any $0 < \alpha < 1$. □

[7] Farkas interpolation algorithm can be seen as the special case $Itp^{(1)}$, but dual Farkas interpolation algorithm is not a special case of the flexible interpolation algorithm.

We next show that also the dual Farkas interpolation algorithm cannot be used as a basis for a tree interpolation algorithm.

Theorem 7. *The dual Farkas interpolation algorithm does not guarantee the strong nor the weak tree-interpolation property.*

Proof. Again in Ex. 1, the dual Farkas interpolants are computed as

$$\overline{I^F}_{X_1} = (x_2 + x_3 < 1) \qquad \overline{I^F}_{X_2} = (x_4 + x_5 < 1)$$
$$\overline{I^F}_{Y} = (-x_2 - x_5 + x_6 < 1) \qquad \overline{I^F}_{XY} = (x_3 + x_4 + x_6 < 1)$$

The implications of Eq. (1) and Eq. (2) are both falsified with assignment $x_2 \mapsto 0, x_3 \mapsto 0.5, x_4 \mapsto 0.5, x_5 \mapsto 0, x_6 \mapsto 0$. □

From Theorem 7 we immediately get the following result.

Corollary 2. *The dual decomposing Farkas interpolation algorithm $\overline{Itp^D}$ does not have the strong nor the weak tree-interpolation property.*

Proof. The interpolants computed by dual Farkas interpolation algorithm $\overline{Itp^F}$ are special cases of dual decomposed interpolants using trivial decompositions. Since Eq. (1) and Eq. (2) are not valid for $\overline{Itp^F}$, they are also invalid for $\overline{Itp^D}$. □

5 Experimental Evaluation

This section provides experimental evidence on the usefulness of the decomposed Farkas tree interpolants obtained using the gradual decomposition algorithm from Sec. 3.3. In the experiments we use the SMT solver OpenSMT [20] for solving and interpolation. The solver implements a wide range of interpolation algorithms, including in particular both Farkas and decomposing Farkas algorithms [6]. These implementations allowed us to manually perform the required experiments also for gradual decomposition. In the following, for convenience, we use I^D and I^F for the tree interpolant resulting from the algorithm $TItp_{Itp^{P+D}(GDec)}$ and $TItp_{Itp^{P+F}}$, respectively.

To obtain benchmarks we used the tool FreqHorn [14] to create bounded model checking (BMC) [4] formulas from the Horn clauses available at https://github.com/chc-comp/chc-comp19-benchmarks/tree/master/lra-ts. We then applied both $TItp_{Itp^{P+D}(GDec)}$ and $TItp_{Itp^{P+F}}$ to the entire BMC formulas. In total the benchmarks consist of 514 LRA UNSAT formulas. We chose these benchmarks since they provide a natural tree structure that can be used by the gradual decomposition, the interpolants often have many non-trivial decompositions (up to 965), and are relatively big (up to 12k LoC).

Our goal in the experiments is to study whether the I^D are genuinely different from the I^F. We chose two example measures for difference of interpolants: (i) the number of top-level conjuncts, and (ii) the number of distinct LRA atoms. In (i), the number of top-level conjuncts is a measure of generalizability of the

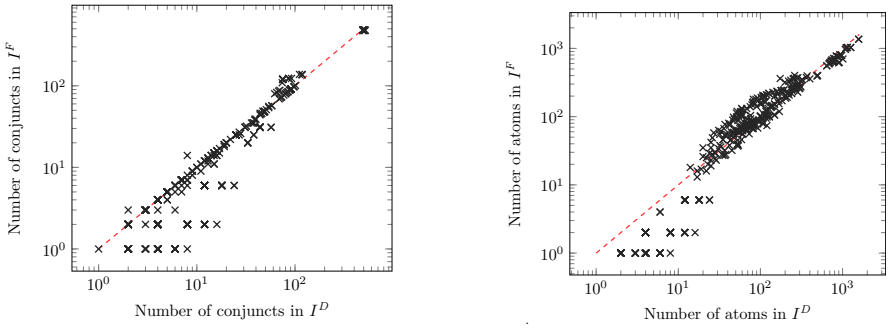

Fig. 2: Comparing decomposed Farkas and Farkas tree interpolants w.r.t the number of conjuncts (*left*) and theory atoms (*right*) in the interpolants.

interpolant. In some applications (see, for example, [14, 22]) it is useful to further abstract an over-approximation, and in case the interpolant has several conjuncts, an easy way to achieve this is by dropping some of them. More technically, for formula f, we define this measure as $N_\wedge(f)$, where $N_\wedge(f) := 1$ if f is not a conjunction at the top-level, and $N_\wedge(f) := n$ if f is of the form $A_1 \wedge \ldots \wedge A_n$. In (ii), the number of distinct theory atoms indicates the complexity of the interpolant in the sense that an instance with more atoms represents a larger Boolean search space. Technically, we define this as the size of the largest subset of $Atoms(f)$ containing only theory atoms.

Fig. 2 (*left*) shows the number of top-level conjuncts for I^F and I^D as a scatter plot. In 53% of the benchmarks I^D have strictly more conjuncts in comparison to I^F. Excluding the cases where the number of top-level conjuncts is the same in I^F and I^D, 91% of benchmarks (270 vs. 25) have strictly more conjuncts in I^D. While a non-negligible number of instances (219) have the same number of conjuncts in I^F and I^D, the majority of the instances are different.

Fig. 2 (*right*) compares the number of unique LRA atoms in I^F and I^D. In almost one-third of the cases I^D contains fewer atoms, suggesting that the decompositions identify semantic structure that is shared between the LRA interpolation queries. We want to emphasize that this is, to the best or our knowledge, a new result that we expect to have practical impact. Based on the numerical results, and contrary to what the figure suggests, there are typically more atoms in I^D. Concretely, out of 514 benchmarks, 63% have fewer theory atoms in I^F. This is in particular because in 57 benchmarks there are two atoms in I^D and one atom in I^F, all represented by the single point (2,1) in the plot. This is somewhat expected, since on a single LRA interpolation query Itp^D is guaranteed to give at least as many atoms as Itp^F. In addition, in no case the number of theory atoms is the same in I^F and I^D, giving a strong indication that the interpolation algorithms differ in practice.

In conclusion, the results imply that a portfolio of the interpolation algorithms $TItp_{Itp^{P+D}(GDec)}$ and $TItp_{Itp^{P+F}}$ provides a range of interpolants that are substantially different from those available using only $TItp_{Itp^{P+F}}$.

6 Conclusion

We identified five classes of interpolation systems for LRA based on Farkas' lemma, and investigated whether and under what conditions they can be used for tree interpolation. In addition to strengthening a known positive result for the Farkas algorithm, we showed that also the binary decomposing Farkas interpolation algorithm can be used as a basis for tree interpolation by using a novel method called *gradual decomposition*. We also showed that TI is not guaranteed by the dual Farkas, the dual decomposing Farkas, and a flexible variant of the Farkas interpolation algorithms.

We showed experimentally, based on two different measures, that Farkas and decomposed Farkas interpolants are often different. In addition, interestingly, it is not uncommon that the decomposed interpolants have fewer theory atoms than the Farkas interpolants, and that it is more common that the decomposed interpolants have more conjuncts also at the top-level of the formulas compared to Farkas interpolants. The existence of the decomposing Farkas interpolation algorithm for tree interpolation enables a liberty in the interpolant choice previously unavailable in the field. We are hopeful that the decomposed interpolants will become a powerful component of interpolation portfolios resulting in more scalable and general solving. In a future work we plan to implement the gradual decomposition in a more automatic way and experiment with the implementation in a more applied setting.

References

1. Alt, L., Hyvärinen, A.E.J., Sharygina, N.: LRA interpolants from no man's land. In: Proc. HVC 2017. LNCS, vol. 10629, pp. 195–210. Springer (2017)
2. Asadi, S., Blicha, M., Hyvärinen, A., Fedyukovich, G., Sharygina, N.: Incremental verification by SMT-based summary repair. In: Proc. FMCAD 2020. IEEE digital library (2020)
3. Barrett, C., Sebastiani, R., Seshia, S., Tinelli, C.: Satisfiability modulo theories, Frontiers in Artificial Intelligence and Applications, vol. 185, pp. 825–885. IOS Press, 1 edn. (2009)
4. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without bdds. In: Proc. TACAS 2003. LNCS, vol. 1579, pp. 193–207. Springer (1999)
5. Blanc, R., Gupta, A., Kovács, L., Kragl, B.: Tree interpolation in Vampire. In: Proc. LPAR 2013. LNCS, vol. 8312, pp. 173–181 (2013)
6. Blicha, M., Hyvärinen, A.E.J., Kofron, J., Sharygina, N.: Decomposing Farkas interpolants. In: Proc. TACAS 2019. LNCS, vol. 11427, pp. 3–20. Springer (2019)
7. Christ, J., Hoenicke, J.: Proof tree preserving tree interpolation. J. Autom. Reasoning **57**(1), 67–95 (2016)
8. Cimatti, A., Griggio, A., Sebastiani, R.: Efficient generation of craig interpolants in satisfiability modulo theories. ACM Trans. Comput. Log. **12**(1), 7:1–7:54 (2010)
9. Craig, W.: Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. In: J. of Symbolic Logic. pp. 269–285 (1957)
10. Detlefs, D., Nelson, G., Saxe, J.B.: Simplify: A theorem prover for program checking. Journal of the ACM **52**(3), 365–473 (2005)

11. D'Silva, V., Kroening, D., Purandare, M., Weissenbacher, G.: Interpolant strength. In: Proc. VMCAI 2010. LNCS, vol. 5944, pp. 129–145. Springer (2010)
12. Dutertre, B., de Moura, L.M.: A fast linear-arithmetic solver for DPLL(T). In: Proc. CAV 2006. LNCS, vol. 4144, pp. 81–94. Springer (2006)
13. Farzan, A., Kincaid, Z.: Strategy synthesis for linear arithmetic games. PACMPL **2**(POPL), 61:1–61:30 (2018)
14. Fedyukovich, G., Bodík, R.: Accelerating Syntax-Guided Invariant Synthesis. In: TACAS, Part I. LNCS, vol. 10805, pp. 251–269. Springer (2018)
15. Fedyukovich, G., Sery, O., Sharygina, N.: eVolCheck: Incremental upgrade checker for C. In: Proc. TACAS 2013. LNCS, vol. 7795, pp. 292–307. Springer (2013)
16. Gupta, A., Popeea, C., Rybalchenko, A.: Solving recursion-free Horn clauses over LI+UIF. In: Proc. APLAS 2011. LNCS, vol. 7078, pp. 188–203. Springer (2011)
17. Gurfinkel, A., Rollini, S.F., Sharygina, N.: Interpolation properties and SAT-based model checking. In: Proc. ATVA 2013. pp. 255–271 (2013)
18. Heizmann, M., Hoenicke, J., Podelski, A.: Nested interpolants. In: Proc. POPL 2010. pp. 471–482. ACM (2010)
19. Hojjat, H., Rümmer, P.: The ELDARICA Horn Solver. In: FMCAD. pp. 158–164. IEEE (2018)
20. Hyvärinen, A.E.J., Marescotti, M., Alt, L., Sharygina, N.: OpenSMT2: An SMT solver for multi-core and cloud computing. In: Proceedings of Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference. LNCS, vol. 9710, pp. 547–553. Springer (2016)
21. Jhala, R., McMillan, K.L.: Interpolant-based transition relation approximation. In: Etessami, K., Rajamani, S.K. (eds.) Computer Aided Verification. pp. 39–51. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
22. Komuravelli, A., Gurfinkel, A., Chaki, S., Clarke, E.M.: Automatic Abstraction in SMT-Based Unbounded Software Model Checking. In: CAV. LNCS, vol. 8044, pp. 846–862. Springer (2013)
23. McMillan, K.L.: An interpolating theorem prover. In: Proc. TACAS 2004. LNCS, vol. 2988, pp. 16–30. Springer (2004)
24. McMillan, K.L., Rybalchenko, A.: Solving constrained Horn clauses using interpolation. Tech. rep., MSR-TR-2013-6 (2013)
25. Pudlák, P.: Lower bounds for resolution and cutting plane proofs and monotone computations. Journal of Symbolic Logic **62**(3), 981–998 (1997)
26. Rollini, S.F., Sery, O., Sharygina, N.: Leveraging interpolant strength in model checking. In: Proc. CAV 2012. LNCS, vol. 7358, pp. 193–209. Springer (2012)
27. Rümmer, P., Hojjat, H., Kuncak, V.: Disjunctive interpolants for Horn-clause verification. In: Proc. CAV 2013. LNCS, vol. 8044, pp. 347–363. Springer (2013)
28. Rybalchenko, A., Sofronie-Stokkermans, V.: Constraint solving for interpolation. In: Proc. VMCAI 2007. LNCS, vol. 4349, pp. 346–362. Springer (2007)
29. Sery, O., Fedyukovich, G., Sharygina, N.: Incremental upgrade checking by means of interpolation-based function summaries. In: Proc. FMCAD 2012. pp. 114 – 121. IEEE (2012)
30. Sharma, R., Nori, A.V., Aiken, A.: Interpolants as classifiers. In: Proc. CAV 2012. LNCS, vol. 7358, pp. 71–87. Springer (2012)

Appendix A

In this appendix we give auxiliary material for more formal treatment of the connection between propositional and theory interpolation. The propositional

resolution rule state that an assignment satisfying the clauses $cl^+ \vee p$ and $cl^- \vee \overline{p}$ also satisfies $cl^+ \vee cl^-$.

Our propositional interpolation works on a refutation of a formula $A \wedge B$. We denote atoms of A and B as $Atoms(A, B)$. Note that each $At \in Atoms(A, B)$ may appear only in A, only in B, or in both conjuncts; Similarly to the notation in [11], we assign a *color* among $\{a, b, ab\}$ independently to each At, depending on whether At occurs only in A, only in B, or in both, respectively.

Table 2 describes the *Pudlák* interpolation algorithm, where the notation $p{:}\varepsilon$ indicates that a literal p has color ε.

Table 2: Pudlák's interpolation algorithm

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

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

Lemma 2 (source clause, base case). *The strong tree-interpolation property holds for Pudlák's interpolation algorithm Itp^P for source clauses.*

Proof. Let cl be a source clause. There are three cases: $cl \in X$, $cl \in Y$, or $cl \in Z$. We consider the three interpolation instances $(X \,|\, Y \wedge Z)$, $(Y \,|\, X \wedge Z)$, and $(X \wedge Y \,|\, Z)$, and check whether TI holds, i.e., whether

$$Itp^P(X \,|\, Y \wedge Z) \wedge Itp^P(Y \,|\, X \wedge Z) \implies Itp^P(X \wedge Y \,|\, Z). \tag{3}$$

The relevant part in the algorithm is shown in Table 2 (*left*).

- $cl \in X$: When $cl \in X$, using Pudlák's interpolation algorithm and substituting the interpolants in Eq. (3), we have $(\bot \wedge \top) \implies \bot$, which is valid.
- $cl \in Y$: The case $cl \in Y$ is symmetric to the case when $cl \in X$, and thus valid.
- $cl \in Z$: When $cl \in Z$, we have again by substiting in Eq. (3) $(\top \wedge \top) \implies \top$, which is valid.

□

Lemma 3 (inner node). *Let p be a variable. In refutation $\mathbb{P}$, if partial interpolants for nodes $cl^+ \vee p$ and $cl^- \vee \hat{p}$ satisfy the strong tree-interpolation property, then the partial interpolant for $cl^+ \vee cl^-$ satisfy the strong tree-interpolation property.*

Proof. We show that for all resolvents in refutation $\mathbb{P}$, the implication $(I_X \wedge I_Y) \implies I_{XY}$ holds, where $I_X = (X \,|\, Y \wedge Z), I_Y = (Y \,|\, X \wedge Z)$, and $I_{XY} = (XY \,|\, Z)$.

we consider a node $cl^+ \vee cl^-$ representing resolution over a variable p with parent nodes $p \vee cl^+$ and $\bar{p} \vee cl^-$. From the inductive hypotheses, we have partial interpolants I_X^+, I_Y^+, and I_{XY}^+ for the node $p \vee cl^+$ so that $(I_X^+ \wedge I_Y^+) \implies I_{XY}^+$ and partial interpolants I_X^-, I_Y^-, and I_{XY}^- for the node $\bar{p} \vee cl^-$ so that $(I_X^- \wedge I_Y^-) \implies I_{XY}^-$.

We consider different cases of coloring of p. Depending on presence of p in the three partitions, i.e., X, Y, and Z, and also depending on interpolation instances $(X \mid Y \wedge Z)$, $(Y \mid X \wedge Z)$, and $(X \wedge Y \mid Z)$, p is colored a, b, or ab (Table 3).

Table 3: Coloring of variable p for each partial interpolant.

appearance of p	class of p for each partial interpolant		
	I_X	I_Y	I_{XY}
X	a	b	a
Y	b	a	a
Z	b	b	b
$X \cap Y$	ab	ab	a
$X \cap Z$	ab	b	ab
$Y \cap Z$	b	ab	ab
$X \cap Y \cap Z$	ab	ab	ab

In case of $p \in X$, based on Pudlák's algorithm 2, $I_X \equiv I_X^+ \vee I_X^-$, $I_Y \equiv I_Y^+ \wedge I_Y^-$, $I_{XY} \equiv I_{XY}^+ \vee I_{XY}^-$.

Using the inductive hypothesis, we have $((I_X^+ \vee I_X^-) \wedge I_Y^+ \wedge I_Y^-) \implies (I_{XY}^+ \vee I_{XY}^-)$, which is the required claim $(I_X \wedge I_Y) \implies I_{XY}$. The case $p \in Y$ is symmetric.

In case of $p \in Z$, we have $I_X \equiv I_X^+ \wedge I_X^-$, $I_Y \equiv I_Y^+ \wedge I_Y^-$, $I_{XY} \equiv I_{XY}^+ \wedge I_{XY}^-$. Using the inductive hypothesis, we have $(I_X^+ \wedge I_X^- \wedge I_Y^+ \wedge I_Y^-) \implies (I_{XY}^+ \wedge I_{XY}^-)$, which is the required claim $(I_X \wedge I_Y) \implies I_{XY}$.

In case of $p \in X \cap Y \cap Z$, using $sel(p, P, Q)$ as a shortcut for $(p \vee P) \wedge (\bar{p} \vee Q)$, we get: $I_X = sel(p, I_X^+, I_X^-)$, $I_Y = sel(p, I_Y^+, I_Y^-)$, $I_{XY} = sel(p, I_{XY}^+, I_{XY}^-)$. Using the inductive hypothesis and considering both possible values of p, we have $(sel(p, I_X^+, I_X^-) \wedge sel(p, I_Y^+, I_Y^-)) \implies sel(p, I_{XY}^+, I_{XY}^-)$, which is the desired claim $(I_X \wedge I_Y) \implies I_{XY}$. The other cases where $p \in X \cap Y$ or $p \in X \cap Z$ or $p \in Y \cap Z$ are subsumed by this case as $(P \wedge Q) \implies sel(p, P, Q) \implies (P \vee Q)$. □