International Journal on Software Tools for Technology Transfer
https://doi.org/10.1007/s10009-021-00641-z

GENERAL

Special Issue: TACAS 2019

Using linear algebra in decomposition of Farkas interpolants

Martin Blicha[1,2] · Antti E. J. Hyvärinen[1] · Jan Kofroň[2] · Natasha Sharygina[1]

Accepted: 20 July 2021

Abstract
The use of propositional logic and systems of linear inequalities over reals is a common means to model software for formal verification. Craig interpolants constitute a central building block in this setting for over-approximating reachable states, e.g. as candidates for inductive loop invariants. Interpolants for a linear system can be efficiently computed from a Simplex refutation by applying the Farkas' lemma. However, these interpolants do not always suit the verification task—in the worst case, they can even prevent the verification algorithm from converging. This work introduces the decomposed interpolants, a fundamental extension of the Farkas interpolants, obtained by identifying and separating independent components from the interpolant structure, using methods from linear algebra. We also present an efficient polynomial algorithm to compute decomposed interpolants and analyse its properties. We experimentally show that the use of decomposed interpolants in model checking results in immediate convergence on instances where state-of-the-art approaches diverge. Moreover, since being based on the efficient Simplex method, the approach is very competitive in general.

Keywords Model checking · Satisfiability modulo theory · Linear real arithmetic · Craig interpolation

1 Introduction

The goal of software verification is to prove specified system properties. To perform verification using automated tools, first, the system needs to be transformed into a representation more suitable for rigorous analysis than source or machine code. In this work, we study a representation in propositional logic together with a system of linear inequalities. It allows for employing techniques and tools from the area of logic, such as SAT and SMT solvers [7,15] and Craig interpolation [12].

In this paper, we focus on safety properties [29]. In particular, we aim at proving facts about parts of the programs and generalizing them. Such generalizations serve as a basis for inductive invariants—formulas representing loops in the program, which make the verification difficult—for guiding the search for a correctness proof in approaches such as IC3 [9] and k-induction [41], both known to scale to the verification of highly complex systems.

Finding good proofs and generalizing them is hard. A widely used approach, satisfiability modulo theories (SMT) [7,15], models a system with fragments of first-order logic. Solvers for SMT combine a resolution-based variant of the DPLL-algorithm [13,14,42] for propositional logic with decision procedures for first-order theories. The SMT-LIB initiative [6] offers currently 55 different first-order fractions called *SMT-LIB logics*. What is common to these logics is that their solving requires typically only a handful of algorithms. Arguably, the two most important algorithms are a congruence closure algorithm for deciding quantifier-free equality with uninterpreted functions [32], and a Simplex-

This work was partially supported by the Charles University institutional funding SVV 260588, by the Czech Science Foundation project 20-07487S, and by the Swiss National Science Foundation (SNSF) project 200021_185031.

✉ Martin Blicha
martin.blicha@usi.ch

Antti E. J. Hyvärinen
antti.hyvaerinen@usi.ch

Jan Kofroň
jan.kofron@d3s.mff.cuni.cz

Natasha Sharygina
natasha.sharygina@usi.ch

[1] Università della Svizzera italiana (USI), Lugano, Switzerland

[2] Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic

Published online: 05 August 2021

based procedure for linear arithmetic over real or rational numbers [18].

Generalizing proofs to inductive invariants is commonly done by Craig interpolation [12]. Here, the system of formulas is split into two parts, say, A and B, resulting in an *interpolation problem* (A, B). The proof of unsatisfiability for $A \wedge B$ is used to extract an *interpolant* I, a formula that is defined over the common symbols of A and B, is implied by A, and is unsatisfiable with B. We perceive the interpolant as a generalization of A with respect to B. Several interpolants can be computed for a given interpolation problem, and not all of them are useful for proving safety. This phenomenon gives rise to employing a *portfolio* [22] of interpolation algorithms that is then applied in the hopes of aiding to find the safety proof with the help of different interpolants.

The approaches to interpolation based on Farkas' lemma construct a linear-real-arithmetic (LRA) interpolant by summing all inequalities appearing in A into a single inequality. We call the resulting interpolant the *Farkas interpolant*. While a single inequality is desirable in some cases, it prevents IC3-style algorithms from converging in others [37]. We show how methods from linear algebra can be applied on a Farkas interpolant to obtain *decomposed interpolants* that do not consist of a single inequality and guarantee the convergence of the model-checking algorithm for some cases where Farkas interpolants fail. A major advantage of decomposed interpolants is that they can still be computed from Farkas coefficients produced by Simplex-based decision procedures, allowing us to re-use the highly tuned implementations present in many state-of-the-art SMT solvers.

Intuitively, while computing the decomposed interpolants, we do not directly sum the inequalities in A, but, instead, we split the sum into sub-sums. The result is an interpolant that is a conjunction of often more than one component of the Farkas interpolant. This allows us not only to solve the convergence problem observed in some model-checking examples but also to gain more control over the strength of LRA interpolants. In summary, the contributions of this paper are:

1. a new Farkas-lemma-based interpolation algorithm for LRA conflicts, which guarantees to decompose a Farkas interpolant to more than one inequality if such decomposition exists;
2. establishing properties regarding logical strength of interpolants produced by our algorithm with respect to the original Farkas interpolants,
3. implementation of our new interpolation algorithm in OpenSMT, our SMT solver, and integration of our approach with the model checker SALLY [26],
4. a set of extensive experiments on a large set of model-checking benchmarks where we evaluate (1) the effect of replacing a traditional interpolation engine with our interpolation algorithm, and (2) the performance of the portfolio of interpolation techniques available in OpenSMT and MathSAT and the contribution of decomposition techniques to the performance of each portfolio.

Fig. 1 Motivating example

```
x = 0;
y = 0;
while (*) {
  x = x + y;
  y = y + 1;
}
assert(x >= 0);
```

This article is an extended version of a conference publication that appeared in [8]. With respect to the aforementioned points, the contribution of this paper over [8] includes:

- Our previous algorithm, presented in [8], relied on a heuristic and did not provide a guarantee to discover a decomposition if one exists. The algorithm presented in this paper provides the guarantee.
- We present complexity analysis of the proposed algorithm.
- We present new results of the experiments reflecting the use of the new version of the algorithm and the progress of the SMT solvers used.
- We provide a detailed comparison with a related approach [11].

The structure of the paper is as follows. In Sec. 2, we motivate the investigation of decomposed interpolants on a concrete model-checking problem where our approach guarantees immediate convergence, but Farkas interpolation diverges. In Sec. 3, we review the related work. In Sec. 4, we define the notation used in the paper, and in Sec. 5 and 6, we detail our main theoretical contribution. We provide experimental results in Sec. 7 and finally conclude in Sec. 8.

2 Motivation

To motivate our work, consider the code in Fig. 1.[1]

In this code, the "$*$" character represents a nondeterministic choice (e.g. user input); thus, the body of the while loop can be executed any number of times. The assert statement captures the property of the program that variable "x" should always be nonnegative after exiting the while loop.

This code can be modelled as a transition system $S = (I, T, Err)$ given in Eq. (1); here, I and Err are predicates that capture the initial and error states, respectively, and T is

[1] This example was first brought to our attention by Prof. Arie Gurfinkel. A similar example appears in [10,37].

the transition relation. The symbols x, y are real variables, and x', y' are their next-state versions.

$$S = \begin{cases} I := (x = 0) \wedge (y = 0), \\ T := (x' = x + y) \wedge (y' = y + 1), \\ Err := (x < 0) \end{cases} \quad (1)$$

The aforementioned example is one variant from a family of similar transition systems that are known not to converge in straightforward implementations of IC3-based algorithms using LRA interpolation. To prove the safety of the transition system (I, T, Err), we search for a safe inductive invariant, i.e. a predicate R that satisfies (1) $I(X) \rightarrow R(X)$, (2) $R(X) \wedge T(X, X') \rightarrow R(X')$, and (3) $R(X) \wedge Err(X) \rightarrow \bot$.

We demonstrate the problem that occurs in model checking when using Farkas interpolants on a simplified run of a model checker for our example. After checking that the initial state satisfies the property $P := x \geq 0$ (the negation of Err), the inductiveness of P is checked. The inductive check is reduced to a satisfiability check of a formula representing the question whether it is possible to reach a $\neg P$-state (a state where $\neg P$ holds) by one step from any P-state:

$$x \geq 0 \wedge x' = x + y \wedge y' = y + 1 \wedge x' < 0.$$

This formula is satisfiable, and a generalized counterexample to induction (CTI) is extracted. In our case, the CTI is $x + y < 0$.[2] This means that if we make one step from a P-state that additionally satisfies $x + y < 0$, we end up in a $\neg P$-state. Therefore, we have to check whether this CTI is consistent with the initial states. This is again encoded as a satisfiability check of a formula

$$x = 0 \wedge y = 0 \wedge x + y < 0.$$

This formula is unsatisfiable, and we can extract an *interpolant* to obtain a generalized reason why this CTI is not consistent with the initial states (not reachable in 0 steps in our system). The interpolant is computed for the partitioning $(x = 0 \wedge y = 0, x + y < 0)$. The Farkas interpolant for this partitioning is $x + y \geq 0$, and we denote it as L_1. Interpolation properties guarantee that L_1 is valid in all initial states. Moreover, P is inductive *relative to* L_1, formally

$$x \geq 0 \wedge x + y \geq 0 \wedge x' = x + y \wedge y' = y + 1 \implies x' \geq 0.$$

This means that by making one step from a P-state that is also an L_1-state we always end up in a P-state again. However, now we need to show that L_1 holds in all reachable states. We check whether L_1 is inductive (even relative to P). Similarly as before, we encode this as a satisfiability check of a formula

$$x + y \geq 0 \wedge x \geq 0 \wedge x' = x + y \wedge y' = y + 1 \wedge x' + y' < 0.$$

Again, this formula is satisfiable, and a generalized CTI is $x + 2y < -1$. This CTI is refuted as inconsistent with the initial states similarly to the first one. The formula

$$x = 0 \wedge y = 0 \wedge x + 2y < -1$$

is unsatisfiable, and Farkas interpolant generalizing the refutation is $L_2 := x + 2y \geq 0$. Similarly as before, it can be easily checked that L_1 is inductive *relative* to L_2, but L_2 is not inductive (not even relative to P and L_1). The CTI is $x + 3y < -1$, and it is refuted by a Farkas interpolant $L_3 := x + 3y \geq 0$. L_2 is now inductive relative to L_3, but L_3 is not inductive, etc. The model checker diverges, since for L_n a CTI $x + ny < -1$ is discovered and a new obligation to show inductiveness of L_{n+1} is generated.

However, let us get back to the first interpolation query $(x = 0 \wedge y = 0, x + y < 0)$. Farkas interpolation, which always computes an interpolant in the form of a *single* inequality, is not the only option. It is possible to compute an interpolant that is a *conjunction* of inequalities. In our case, $L := x \geq 0 \wedge y \geq 0$ is also an interpolant. This interpolant L is stronger than the Farkas interpolant; the property P is inductive relative to L, and, most importantly, L is inductive:

$$(x \geq 0 \wedge y \geq 0) \wedge x' = x + y \wedge y' = y + 1 \implies (x' \geq 0 \wedge y' \geq 0)$$

is a valid formula. Actually, P follows from L, so L represents the inductive strengthening of P that witnesses the safety of our system.

In this work, we present an approach that allows the computation of Craig interpolants in LRA in this conjunctive form.

3 Related work

The possible weakness of Farkas interpolants for use in model checking was recognized in [37]. The authors demonstrate that Farkas interpolation does not satisfy the condition needed for proving convergence of a model-checking algorithm PD- KIND [26]. Indeed, the model checker SALLY [26], which implements PD-KIND, diverges on our example from Sec. 2 if Farkas interpolation is used in its underlying interpolation engine. To resolve this problem, [37] introduces a new interpolation procedure that guarantees the convergence of a special sequence of interpolation problems often occurring

[2] The exact procedure for obtaining the CTI is not important for the current discussion.

in model-checking problems. However, this interpolation algorithm is based on a decision procedure called conflict resolution [28], which is not as efficient as the Simplex-based decision procedure used by most state-of-the-art SMT solvers. In contrast, we show how the original Simplex-based decision procedure using Farkas coefficients can be modified to produce interpolants not restricted to the single-inequality form, while additionally obtaining strength guarantees with respect to the original Farkas interpolants.

The reasoning engine SPACER [27] is also known to be affected by this weakness of Farkas interpolants. The verification framework SEAHORN [20], which relies on SPACER, uses additional invariants obtained from abstract interpretation to avoid the divergence.

The interpolation in linear real arithmetic (LRA) itself has received a significant amount of attention recently. The work on LRA interpolation dates back to 1997 [33]. A compact set of rules for deriving LRA interpolants from the proof of unsatisfiability in an inference system was presented in [31]. The interpolants in these works were the Farkas interpolants. Current methods usually compute Farkas interpolants from explanations of unsatisfiability extracted directly from the Simplex-based decision procedure inside the SMT solver [18]. Recently in [4], we presented a way of computing an infinite family of interpolants between a primal and a dual interpolant with variable strength. However, those interpolants are still restricted to single inequalities.

The first discussion on how to obtain interpolants in the form of conjunction of inequalities from Farkas coefficients is present in [11]. However, their approach is based on a simple heuristic which does not discover the possibility for decompositions in some cases where our approach finds the decomposition easily. Moreover, their focus was on the interpolation techniques themselves, and they do not discuss the applications of decomposed interpolants. We provide a detailed comparison to our approach in Sec. 6.3.

Other works on LRA interpolants include, e.g. [1,36,38]. Both [1] and [38] focus on producing simple overall interpolants by attempting to re-use (partial) interpolants from pure LRA conflicts. Our focus is not on the overall interpolant, but on a single LRA conflict. However, in the context of interpolants from proofs produced by SMT solvers, our approach also has the potential for re-using components of interpolants for LRA conflicts across the whole proof. Beside interpolation algorithms for LRA conflicts, there exists a large body of work on propositional interpolation [2,16,21,25].

4 Preliminaries

We work in the domain of *Satisfiability Modulo Theories* (SMT) [7,15], where satisfiability of formulas is determined with respect to some background theory. In particular, we are concerned with the *lazy* approach to SMT that combines a SAT solver dealing with the propositional structure of a formula and a *theory* solver for checking consistency of a conjunction of theory literals. The proof of unsatisfiability in this approach is basically a propositional proof that incorporates *theory lemmas* learned by the theory solver and propagated to the SAT solver. The proof-based interpolation algorithm then combines any propositional-proof-based interpolation algorithm with *theory interpolator*. Theory interpolator provides an interpolant for each theory conflict—an unsatisfiable conjunction of theory literals.

Linear arithmetic and linear algebra. We use the letters x, y, z to denote variables and c, k to denote constants. Vector of n variables is denoted by $\mathbf{x} = (x_1, \ldots, x_n)^\intercal$ where n is usually known from context. $\mathbf{x}[i]$ denotes the element of $\mathbf{x}$ at position i, i.e. $\mathbf{x}[i] = x_i$. The vector of all zeroes is denoted as $\mathbf{0}$, and $\mathbf{e_i}$ denotes the unit vector with $\mathbf{e_i}[i] = 1$ and $\mathbf{e_i}[j] = 0$ for $j \neq i$. For two vectors $\mathbf{x} = (x_1, \ldots, x_n)^\intercal$ and $\mathbf{y} = (y_1, \ldots, y_n)^\intercal$, we say that $\mathbf{x} \leq \mathbf{y}$ iff $x_i \leq y_i$ for each $i \in \{1, \ldots, n\}$. $\mathbb{Q}$ denotes the set of rational numbers, $\mathbb{Q}^n$ the n-dimensional vector space of rational numbers and $\mathbb{Q}^{m \times n}$ the set of rational matrices with m rows and n columns. A transpose of matrix M is denoted as $M^\intercal$. A *kernel* (or *null space*) of a matrix M is the vector space $ker(M) = \{\mathbf{x} \mid M\mathbf{x} = \mathbf{0}\}$. A matrix is said to be in *row echelon form* (REF) if all nonzero rows are above all rows containing only zeros and the leading coefficient (first nonzero value) of each row is always strictly to the right of the leading coefficient of the row above. A matrix is said to be in *reduced row echelon form* (RREF) if it is in REF, the leading entry of each nonzero row is 1, and each column containing the leading entry of some row has zeros everywhere else. REF of a matrix can be obtained by Gaussian elimination, while RREF can be obtained by Gauss–Jordan elimination.

We adopt the notation of matrix product for linear arithmetic. For a linear term $l = c_1x_1 + \cdots + c_nx_n$, we write $\mathbf{c}^\intercal\mathbf{x}$ to denote l. Without loss of generality, we assume that all linear inequalities are of the form $\mathbf{c}^\intercal\mathbf{x} \bowtie c$ with $\bowtie \in \{\leq, <\}$. By linear system over variables $\mathbf{x}$, we mean a finite set of linear inequalities $S = \{C_i \mid 1 \leq i \leq m\}$, where each C_i is a linear inequality over $\mathbf{x}$. Note that from the logical perspective, each C_i is an atom in the language of the theory of linear arithmetic; thus, system S can be expressed as a formula $\bigwedge_{i=1}^{m} C_i$ and we use these representations interchangeably. A linear system is satisfiable if there exists an evaluation of variables that satisfies all inequalities; otherwise, it is unsatisfiable. This is the same as the (un)satisfiability of the formula representing the system.

We extend the matrix notation also to the whole linear system. For the sake of simplicity, we use $\leq$ instead of $\bowtie$, even if the system contains a mix of strict and non-strict inequalities.

The only important difference is that a (weighted) sum of a linear system (as defined below) results in a strict inequality, instead of a non-strict one, when at least one strict inequality is present in the sum with a nonzero coefficient. The theory, proofs, and algorithm remain valid also in the presence of strict inequalities. We write $C\mathbf{x} \leq \mathbf{c}$ to denote the linear system S where C denotes the matrix of all coefficients of the system, $\mathbf{x}$ are the variables, and $\mathbf{c}$ is the vector of the right sides of the inequalities. With the matrix notation, we can easily express the sum of (multiples) of inequalities. Given a system of inequalities $C\mathbf{x} \leq \mathbf{c}$ and a vector of "weights" (multiples) of the inequalities $\mathbf{k} \geq \mathbf{0}$, the inequality that is the (weighted) sum of the system can be expressed as $\mathbf{k}^\intercal C\mathbf{x} \leq \mathbf{k}^\intercal \mathbf{c}$.

Craig interpolation. Given two formulas $A(\mathbf{x}, \mathbf{y})$ and $B(\mathbf{y}, \mathbf{z})$ such that $A \wedge B$ is unsatisfiable, a *Craig interpolant* [12] is a formula $I(\mathbf{y})$ such that $A \implies I$ and $I \implies \neg B$.

The pair of formulas (A, B) is also referred to as an *interpolation problem*. In linear arithmetic, the interpolation problem is a linear system S partitioned into two parts: A and B.

One way to compute a solution to an interpolation problem in linear arithmetic, used in many modern SMT solvers, is based on Farkas' lemma [19,39]. Farkas' lemma states that for an unsatisfiable system of linear inequalities $S \equiv C\mathbf{x} \leq \mathbf{c}$, there exist *Farkas* coefficients $\mathbf{k} \geq \mathbf{0}$ such that $\mathbf{k}^\intercal C\mathbf{x} \leq \mathbf{k}^\intercal \mathbf{c} \equiv 0 \leq -1$. In other words, the weighted sum of the system given by the Farkas coefficients is a contradictory inequality. If a strict inequality is part of the sum, the result might also be $0 < 0$.

The idea behind the interpolation algorithm based on Farkas coefficients is simple. Intuitively, given the partitioning of the linear system into A and B, we compute only the weighted sum of A. It is not hard to see that this sum is an interpolant. It follows from A because a weighted sum of a linear system with nonnegative weights is always implied by the system. It is inconsistent with B because its sum with the weighted sum of B (using Farkas coefficients) is a contradictory inequality by Farkas' lemma. Finally, it cannot contain any A-local variables, as can be seen from the following reasoning: all variables are eliminated in the weighted sum of the whole system. Since A-local variables are by definition absent in B, they must be eliminated already in the weighted sum of A.

More formally, for an unsatisfiable linear system $S := C\mathbf{x} \leq \mathbf{c}$ over n variables, where $C \in \mathbb{Q}^{m \times n}$, $\mathbf{c} \in \mathbb{Q}^m$, and its partition to $A := C_A\mathbf{x} \leq \mathbf{c_A}$ and $B := C_B\mathbf{x} \leq \mathbf{c_B}$, where $C_A \in \mathbb{Q}^{k \times n}$, $C_B \in \mathbb{Q}^{l \times n}$, $\mathbf{c_A} \in \mathbb{Q}^k$, $\mathbf{c_B} \in \mathbb{Q}^l$ and $k + l = m$, there exist Farkas coefficients $\mathbf{k}^\intercal = (\mathbf{k_A}^\intercal\ \mathbf{k_B}^\intercal)$ such that

$$(\mathbf{k_A}^\intercal\ \mathbf{k_B}^\intercal)\begin{pmatrix} C_A \\ C_B \end{pmatrix} = 0,\ (\mathbf{k_A}^\intercal\ \mathbf{k_B}^\intercal)\begin{pmatrix} \mathbf{c_A} \\ \mathbf{c_B} \end{pmatrix} = -1,$$

and the *Farkas interpolant* for (A, B) is the inequality

$$I^F := \mathbf{k_A}^\intercal C_A\mathbf{x} \leq \mathbf{k_A}^\intercal \mathbf{c_A}. \tag{2}$$

5 Decomposed Interpolants

In this section, we present our new approach to computing interpolants in linear arithmetic based on Farkas coefficients. The definition of Farkas interpolant of Eq. (2) corresponds to the weighted sum of A-part of the unsatisfiable linear system. This sum can be decomposed into j sums by decomposing the vector $\mathbf{k_A}$ into j vectors

$$\mathbf{k_A} = \sum_{i=1}^{j} \mathbf{k_{A,i}},\ \text{with } \mathbf{0} \leq \mathbf{k_{A,i}} \leq \mathbf{k_A} \text{ for all } i, \tag{3}$$

thus obtaining j inequalities

$$I_i := \mathbf{k_{A,i}}^\intercal C_A\mathbf{x} \leq \mathbf{k_{A,i}}^\intercal \mathbf{c_A} \tag{4}$$

If $\mathbf{k_{A,i}}$ are such that the left-hand side of the inequalities I_i contains only shared variables, the decomposition has an interesting application in interpolation, as illustrated below.

Definition 1 (decomposed interpolants) Given an interpolation instance (A, B), if there exists a sum of the form Eq. (3) such that the left side of Eq. (4) contains only shared variables for all $1 \leq i \leq j$, then the set of inequalities $D = \{I_1, \ldots, I_j\}$ is a *decomposition*.

In that case the formula $\bigwedge_{i=1}^{j} I_i$ is a *decomposed interpolant* (DI) of size j for (A, B).

The decomposed interpolants are proper interpolants, as stated in the following theorem.

Theorem 1 *Let (A, B) be an interpolation problem in linear arithmetic. If $D = \{I_1, \ldots, I_k\}$ is a decomposition, then $I^D = I_1 \wedge \ldots \wedge I_k$ is an interpolant for (A, B).*

Proof Let $I^D = I_1 \wedge \ldots \wedge I_k$. First, $A \implies I^D$ since for all I_i, $A \implies I_i$. This is immediate from the fact that A is a system of linear inequalities $C_A\mathbf{x} \leq \mathbf{c_A}$, $I_i = (\mathbf{k_{A,i}}^\intercal C_A\mathbf{x} \leq \mathbf{k_{A,i}}^\intercal \mathbf{c_A})$ and $\mathbf{0} \leq \mathbf{k_{A,i}}$.

Second, $I^D \wedge B \implies \bot$ since I^D implies Farkas interpolant I^F. This holds because $\mathbf{k_A} = \sum_i \mathbf{k_{A,i}}$ and $\mathbf{0} \leq \mathbf{k_{A,i}}$.

Third, I^D contains only the shared variables by the definition of decomposition (Definition 1). Therefore, I^D is an interpolant. □

Each interpolation instance has a DI of size one, a *trivial* decomposition, corresponding to the Farkas interpolant of Eq. (2). However, interpolation problems, in general, can admit bigger decompositions. In the following, we give a

concrete example of an instance with decomposition of size two.

Example 1 Let (A, B) be an interpolation problem in linear arithmetic with $A = (x_1 + x_2 \leq 0) \wedge (x_1 + x_3 \leq 0) \wedge (-x_1 \leq 0)$ and $B = (-x_2 - x_3 \leq -1)$. The linear systems corresponding to A and B are

$$C_A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ -1 & 0 & 0 \end{pmatrix}, \qquad \mathbf{c_A} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$
$$C_B = \begin{pmatrix} 0 & -1 & -1 \end{pmatrix}, \qquad \mathbf{c_B} = \begin{pmatrix} -1 \end{pmatrix}.$$

Farkas coefficients are

$$\mathbf{k_A^\intercal} = \begin{pmatrix} 1 & 1 & 2 \end{pmatrix} \text{ and } \mathbf{k_B^\intercal} = \begin{pmatrix} 1 \end{pmatrix},$$

while Farkas interpolant for (A, B) is the inequality $I^F := x_2 + x_3 \leq 0$. However, if we decompose $\mathbf{k_A}$ into

$$\mathbf{k_{A,1}^\intercal} = \begin{pmatrix} 1 & 0 & 1 \end{pmatrix} \text{ and } \mathbf{k_{A,2}^\intercal} = \begin{pmatrix} 0 & 1 & 1 \end{pmatrix},$$

we obtain the decomposition $\{x_2 \leq 0, x_3 \leq 0\}$ producing the decomposed interpolant $I^{DI} := x_2 \leq 0 \wedge x_3 \leq 0$ of size two.

5.1 Strength-Based Ordering of Decompositions

Decomposition of Farkas coefficients for a single interpolation problem is in general not unique. However, we can provide some structure to the space of possible interpolants by ordering interpolants with respect to their logical strength. To achieve this, we define the *coarseness* of a decomposition based on its ability to partition the terms of the interpolant into finer sums and then prove that coarseness provides us with a way of measuring the interpolant strength.

Definition 2 Let D_1, D_2 denote two decompositions of the same interpolation problem of size m, n, respectively, where $n < m$. Let $(\mathbf{q_1}, \ldots, \mathbf{q_m})$ denote the decomposition of Farkas coefficients corresponding to D_1 and let $(\mathbf{r_1}, \ldots, \mathbf{r_n})$ denote the decomposition of Farkas coefficients corresponding to D_2. We say that decomposition D_1 is *finer* than D_2 (or equivalently D_2 is *coarser* than D_1) and denote this as $D_1 \prec D_2$ when there exists a partitioning $P = \{p_1, \ldots, p_n\}$ of the set $\{\mathbf{q_1}, \ldots, \mathbf{q_m}\}$ such that for each i with $1 \leq i \leq n$, $\mathbf{r_i} = \sum_{\mathbf{q} \in p_i} \mathbf{q}$.

Interpolants of decompositions ordered by their coarseness can be ordered by logical strength, as stated by the following lemma:

Lemma 1 *Assume D_1, D_2 are two decompositions of the same interpolation problem such that $D_1 \prec D_2$. Let I^{D_1}, I^{D_2} be the decomposed interpolants corresponding to D_1, D_2. Then, I^{D_1} implies I^{D_2}.*

Proof Informally, the implication follows from the fact that each linear inequality of I^{D_2} is a sum of some inequalities in I^{D_1}.

Formally, let I_i denote the i-th inequality in I^{D_2}. Then, $I_i = (\mathbf{r_i^\intercal} C_A \mathbf{x} \leq \mathbf{r_i^\intercal c_A})$. Since $D_1 \prec D_2$, there is a set $\{I_{i_1}, \ldots, I_{i_j}\} \subseteq D_1$ such that for each k with $1 \leq k \leq j$, $I_{i_k} = (\mathbf{q_{i_k}^\intercal} C_A \mathbf{x} \leq \mathbf{q_{i_k}^\intercal c_A})$ and $\mathbf{r_i} = \sum_{k=1}^{j} \mathbf{q_{i_k}}$.

Since $\mathbf{q_{i_k}} \geq \mathbf{0}$, it holds that $I_{i_1} \wedge \cdots \wedge I_{i_j} \implies I_i$. This means that I^{D_1} implies every conjunct of I^{D_2}. □

Note that the trivial, single-element decomposition corresponding to Farkas interpolant is the greatest element of this decomposition ordering. Also, for any decomposition of size more than one, replacing any number of elements by their sum yields a coarser decomposition.

Finally, we emphasize that it is difficult to argue about the suitability of a decomposition for a particular purpose based solely on strength. For example, a user may opt for a coarser decomposition because summing up just some elements of a decomposition may result in eliminating a shared variable.

5.2 Strength of the Dual Interpolants

Before we describe the details of the decomposing interpolation procedure, we extend the picture of interpolation strength related to the decomposed interpolants.

Some applications of interpolation can benefit from computing coarser over-approximation (i.e. weaker interpolants). For example, a weaker function summary can cover more changes in an upgrade checking scenario [40], and weaker over-approximations of reachability in a transition system can converge to fix-point faster [16]. Using the notion of *dual interpolation*, decompositions can also be used to compute interpolants *weaker* than Farkas interpolant (or even its dual).

Given an interpolation problem (A, B) and an interpolation procedure Itp, we denote the interpolant computed by Itp for (A, B) as $Itp(A, B)$. Then, Itp' denotes the *dual* interpolation procedure, which works as follows: $Itp'(A, B) = \neg Itp(B, A)$. The well-known duality theorem for interpolation states that Itp' is a correct interpolation procedure.

Let us denote the interpolation procedure based on Farkas' lemma as Itp_F and the decomposing interpolation procedure as Itp_{DI}. The relation between Itp_F and its dual Itp'_F has been established in [4], namely that $Itp_F(A, B) \implies Itp'_F(A, B)$. We have shown in Lemma 1 that a decomposed interpolant always implies Farkas interpolant computed from the same Farkas coefficients. Formally, $Itp_{DI}(A, B) \implies Itp_F(A, B)$. Similar result can be established for the dual interpolation procedures: as $Itp_{DI}(B, A) \implies Itp_F(B, A)$,

it follows that $\neg Itp_F(B, A) \implies \neg Itp_{DI}(B, A)$ and consequently $Itp'_F(A, B) \implies Itp'_{DI}(A, B)$.

Combining the results on logical strength together, we obtain a chain of implications:

$$Itp_{DI}(A, B) \implies Itp_F(A, B) \implies Itp'_F(A, B) \implies Itp'_{DI}(A, B).$$

Note that while both Itp_F and Itp'_F compute interpolants as a single inequality and interpolants produced by Itp_{DI} are *conjunctions* of inequalities, interpolants produced by Itp'_{DI} are *disjunctions* of inequalities.

In the following section, we describe the details of the Itp_{DI} interpolation procedure.

6 Finding Decompositions

In this section, we present our approach for finding decompositions for linear arithmetic interpolation problems given their Farkas coefficients.

We focus on the task of finding decomposition of $\mathbf{k_A}^\intercal C_A \mathbf{x}$. Recall that $C_A \in \mathbb{Q}^{l \times n}$ and $\mathbf{x}$ is a vector of variables of length n. Without loss of generality, assume that there are no B-local variables since columns of C_A corresponding to B-local variables would contain all zeroes by definition in any case.

Furthermore, without loss of generality, assume the variables in the inequalities of A are ordered such that all A-local variables are before the shared ones. Then, let us write

$$C_A = \begin{pmatrix} L & S \end{pmatrix}, \quad \mathbf{x}^\intercal = \begin{pmatrix} \mathbf{x}_L{}^\intercal & \mathbf{x}_S{}^\intercal \end{pmatrix} \tag{5}$$

where $\mathbf{x}_L$ is the vector of A-local variables of size p, $\mathbf{x}_S$ the vector of shared variables of size q, $n = p + q$, $L \in \mathbb{Q}^{l \times p}$, and $S \in \mathbb{Q}^{l \times q}$. We know that $\mathbf{k_A}^\intercal L = \mathbf{0}$ and the goal is to find $\mathbf{k_{A,i}}$ such that $\sum_i \mathbf{k_{A,i}} = \mathbf{k_A}$ and for each i $\mathbf{0} \leq \mathbf{k_{A,i}} \leq \mathbf{k_A}$ and $\mathbf{k_{A,i}}^\intercal L = \mathbf{0}$.

In the following, we will consider two cases for computing the decompositions. We first study a common special case where system A contains rows with no local variables and give a linear-time algorithm for computing the decompositions. We then move to the general case where the rows of A contain local variables and provide a decomposition algorithm based on computing a vector basis for a null space of a matrix obtained from A.

6.1 Trivial Elements

First, consider a situation where there is a linear inequality with no local variables. This means there is a row j in C_A (denoted as $C_{A\,j}$) such that all entries in columns corresponding to local variables are 0, i.e. $L_j = \mathbf{0}^\intercal$. Then, $\{I_1, I_2\}$ for $\mathbf{k_{A,1}} = \mathbf{k_A}[j] \times \mathbf{e_j}$ and $\mathbf{k_{A,2}} = \mathbf{k_A} - \mathbf{k_{A,1}}$ is a decomposition. Intuitively, any linear inequality that contains only shared variables can form a stand-alone element of a decomposition. When looking for finest decomposition, we do this iteratively for all inequalities with no local variables. In the next part, we show how to look for a non-trivial decomposition when dealing with local variables.

6.2 Decomposing in the Presence of Local Variables

For this section, assume that L has no zero rows. (We have shown above how to deal with such rows.) We are going to search for a non-trivial decomposition starting with the following observation:

Observation $\mathbf{k_A}^\intercal L = 0$. *Equivalently, there are no A-local variables in the Farkas interpolant. It follows that* $L^\intercal \mathbf{k_A} = 0$ *and* $\mathbf{k_A}$ *is in the kernel of* $L^\intercal$.

Let us denote by $\mathbb{K} = ker(L^\intercal)$ the kernel of $L^\intercal$.

Theorem 2 *Let* $\mathbf{v_1}, \ldots, \mathbf{v_n}$ *be vectors from* $\mathbb{K}$ *such that* $\exists \alpha_1, \ldots, \alpha_n$ *with* $\alpha_i \mathbf{v_i} \geq \mathbf{0}$ *for all* i *and* $\mathbf{k_A} = \sum_{i=1}^n \alpha_i \mathbf{v_i}$.

Then, $\{\mathbf{w_1}, \ldots, \mathbf{w_n}\}$ *for* $\mathbf{w_i} = \alpha_i \mathbf{v_i}$ *is a decomposition of* $\mathbf{k_A}$ *and* $D = \{I_1, \ldots, I_n\}$ *for* $I_i := \mathbf{w_i} C_A \mathbf{x} \leq \mathbf{c_A}$ *is a decomposition, i.e. the formula* $I^D = \bigwedge_{i=1}^n I_i$ *is a decomposed interpolant.*

Proof The theorem follows from the definition of decomposition (Def. 1). From the assumptions of the theorem, we immediately obtain $\mathbf{k_A} = \sum_{i=1}^n \mathbf{w_i}$ and $\mathbf{w_i} \geq \mathbf{0}$. Moreover, $\mathbf{w_i} \in \mathbb{K}$, since $\mathbf{v_i} \in \mathbb{K}$ and $\mathbf{w_i} = \alpha_i \mathbf{v_i}$. As a consequence, $L^\intercal \mathbf{w_i} = 0$ and it follows that there are no A-local variables in $\mathbf{w_i}^\intercal C_A \mathbf{x}$. □

Note that Theorem 2 permits redundant components of a decomposition. Consider vectors $\mathbf{w_1}, \mathbf{w_2}, \mathbf{w_3} \in \mathbb{K}$ that are part of a decomposition in the sense of Theorem 2 and that $\mathbf{w_3} = \mathbf{w_1} + \mathbf{w_2}$. Then, $I_1 \wedge I_2 \implies I_3$ and I_3 is a *redundant conjunct* in the corresponding decomposed interpolant.

Good candidates that satisfy most of the assumptions of Theorem 2 (and avoid redundancies) are bases of the vector space $\mathbb{K}$. If $B = \{\mathbf{b_1}, \ldots, \mathbf{b_n}\}$ is a basis of $\mathbb{K}$ such that $\mathbf{k_A} = \sum_{i=1}^n \alpha_i \mathbf{b_i}$ with $\alpha_i \mathbf{b_i} \geq \mathbf{0}$ for all i, then $\{\alpha_1 \mathbf{b_1}, \ldots, \alpha_n \mathbf{b_n}\}$ is a decomposition. Our solution for computing the decomposition of Farkas coefficients $\mathbf{k_A}$ is described in Algorithm 1. It is based on the above idea of computing bases of $ker(L^\intercal)$. First, after transforming the matrix to the RREF form, we compute a basis of the kernel using the standard linear-algebra algorithm. The basis is almost what we want, except that some vectors of this basis can have negative coefficients. In such a case, our algorithm gradually updates the basis until all vectors from the basis are nonnegative while preserving all the necessary properties. Such a basis is used to compute the desired decomposition. Now, we describe our algorithm in

input : matrix M, vector $\mathbf{v}$ such that $\mathbf{v} \in ker(M)$ and $\mathbf{v} > \mathbf{0}$
output: $\{\mathbf{w_1}, ..., \mathbf{w_n}\}$, a decomposition of $\mathbf{v}$, such that $\mathbf{w_i} \in ker(M)$, $\mathbf{w_i} \geq \mathbf{0}$ and $\sum \mathbf{w_i} = \mathbf{v}$

1 $M \leftarrow \texttt{RREF}(M)$
2 $n \leftarrow \texttt{Nullity}(M)$
3 **if** $n = 1$ **then return** $\{\mathbf{v}\}$ $(\mathbf{b_1}, \ldots, \mathbf{b_n}) \leftarrow \texttt{KernelBasis}(M)$
4 $(\alpha_1, \ldots, \alpha_n) \leftarrow \texttt{Coordinates}(\mathbf{v}, (\mathbf{b_1}, \ldots, \mathbf{b_n}))$
5 **assert** $\alpha_k > 0$ for each $k = 1, \ldots, n$
6 **while** $\exists i, j$ *such that* $\mathbf{b_i}_j < 0$ **do**
7 $\quad C \leftarrow 1 + \frac{-\mathbf{b_i}_j \alpha_i}{\mathbf{v}_j}$
8 $\quad \mathbf{b_i} \leftarrow \mathbf{b_i} + \frac{-\mathbf{b_i}_j}{\mathbf{v}_j}\mathbf{v}$
9 $\quad (\alpha_1, \ldots, \alpha_n) \leftarrow (\frac{\alpha_1}{C}, \ldots, \frac{\alpha_n}{C})$
10 $\quad$ **assert** $\alpha_k > 0$ for each $k = 1, \ldots, n$
11 $\quad$ **assert** $\mathbf{v} = \sum_{k=1}^{n} \alpha_k \mathbf{b_k}$
12 **end**
13 **assert** $\mathbf{b_k} \geq \mathbf{0}$ for each $k = 1, \ldots, n$
14 **return** $\{\alpha_1 \mathbf{b_1}, \ldots, \alpha_n \mathbf{b_n}\}$

Algorithm 1: Algorithm for decomposition of Farkas coefficients

detail, show its termination and correctness, and discuss its complexity.

The algorithm runs on the matrix $M = L^\intercal$ and vector $\mathbf{v} = \mathbf{k_A}$. At the beginning, the *reduced row echelon form* (RREF) of the matrix is computed. (Recall definition of RREF from Sec. 4.) Importantly, the transformation of a matrix to RREF preserves its kernel. The dimension of the kernel, known as *nullity*, can now be efficiently computed using *rank–nullity theorem*, which states that the nullity of a matrix is equal to the number of its columns minus its rank. For a matrix in RREF, the rank is simply the number of nonzero rows.

We already know that there is a nonzero vector in the kernel; therefore, the nullity of the matrix is at least one. If it is *exactly* one (line 3), then no non-trivial decomposition of the vector exists. Intuitively, this means that the Farkas coefficients represent *the unique way* (up to positive scalar multiples) of summing up the inequalities of A-part to eliminate the A-local variables. However, if the nullity is greater than one, it is possible to compute a decomposition of size equal to the nullity.

Initial basis computation. First, a basis of the kernel of the matrix in RREF is computed by a standard algorithm (see, for example, [5]). This algorithm ensures that the coordinates of $\mathbf{v}$, with respect to the basis it computes, are positive (lines 5, 6). Since this is an important property, we include the description of the algorithm with the proof. Given a matrix M in RREF with m columns, each column is denoted as either pivot or non-pivot. A pivot column contains the first nonzero entry for a particular row; non-pivot column does not. We say that a non-pivot column is *free*. The number of free columns is exactly the nullity of the matrix, i.e. n, and the number of pivot columns is $m - n$. Due to the need to iterate over the pivot and free columns separately, we introduce additional notation: we use $f \in \{1, \ldots, n\}$ to iterate over the free columns, $p \in \{1, \ldots, m - n\}$ to iterate over the pivot columns, and we use mapping functions $F : \{1, \ldots, n\} \to \{1, \ldots, m\}$ and $P : \{1, \ldots, m-n\} \to \{1, \ldots, m\}$ to get the original column indices in M.

Now, for each $f \in \{1, \ldots, n\}$ denote as $\mathbf{b_f}$ the solution obtained by solving the system $M\mathbf{x} = \mathbf{0}$ where all variables corresponding to free columns are set to 0, except for $x_{F(f)}$ which is set to 1. Note that this uniquely determines the value of pivot variables since M is in RREF; thus,

$$x_{P(p)} = \sum_{f=1}^{n} -M_{pF(f)} x_{F(f)}, \forall p \in \{1, \ldots, m-n\} \tag{6}$$

Lemma 2 *$\mathcal{B} = \{\mathbf{b_f} \mid f \in \{1, \ldots, n\}\}$ is a basis of $ker(M)$. Moreover, $\forall \mathbf{v} \in ker(M) : \mathbf{v} = \sum_{f=1}^{n} \mathbf{v}_{F(f)} \mathbf{b_f}$.*

Proof **Linear independence:** For each $f \in \{1, \ldots, n\}$, $\mathbf{b_f}$ has 1 at position $F(f)$, while all other elements of $\mathcal{B}$ have 0 at position $F(f)$. Consequently, $\mathbf{b_f}$ cannot be expressed as a linear combination of other elements of $\mathcal{B}$.

Generators: We show that each vector $\mathbf{v} \in ker(M)$ can be written as a linear combination of elements of $\mathcal{B}$. More precisely, we show that $\mathbf{v} = \sum_{f=1}^{n} \mathbf{v}_{F(f)} \mathbf{b_f}$.

(a) For each $f \in \{1, \ldots, n\} : \mathbf{v}_{F(f)} = \sum_{\hat{f}=1}^{n} \mathbf{v}_{F(\hat{f})} \mathbf{b_{\hat{f}}}_{F(f)}$ as $\mathbf{b_f}_{F(f)} = 1$ and $\mathbf{b_{\hat{f}}}_{F(f)} = 0$ for $\hat{f} \neq f$.
(b) Fix a pivot index $p \in \{1, \ldots, m-n\}$. To see that $\mathbf{v}_{P(p)} = \sum_{f=1}^{n} \mathbf{v}_{F(f)} \mathbf{b_f}_{P(p)}$, note that $\mathbf{v}$ and all elements of $\mathcal{B}$ are solutions to the system $M\mathbf{x} = \mathbf{0}$, so they satisfy Eq. (6). Instantiating Eq. (6) with $\mathbf{b_f}$ for $f \in \{1, \ldots, n\}$, we get

$$\mathbf{b_f}_{P(p)} = \sum_{\hat{f}=1}^{n} -M_{pF(\hat{f})} \mathbf{b_f}_{F(\hat{f})} = -M_{pF(f)} \tag{7}$$

since $\mathbf{b_f}_{F(\hat{f})} = 1$ when $\hat{f} = f$ and 0 otherwise. Now, $\mathbf{v}_{P(p)} = \sum_{f=1}^{n} \mathbf{v}_{F(f)} \mathbf{b_f}_{P(p)}$ is obtained by instantiating Eq. (6) with $\mathbf{v}$ and then replacing $-M_{pF(f)}$ by $\mathbf{b_f}_{P(p)}$ using Eq. (7).

Combining (a) and (b), we have shown that $\mathbf{v}$ can be expressed as a linear combination of $\mathcal{B}$, which together with the linear independence of $\mathcal{B}$ concludes the proof. □

A direct consequence of Lemma 2 is that the *coordinates* of $\mathbf{v} \in ker(M)$ with respect to basis $\mathcal{B}$, i.e. the coefficients of elements of $\mathcal{B}$ in the linear combination expressing $\mathbf{v}$, are positive if $\mathbf{v} > 0$. These coordinates are denoted as $\alpha_1, \ldots, \alpha_n$ in Algorithm 1, and we have just shown that using this standard algorithm for the computation of a kernel's basis the coordinates are guaranteed to be positive (line 6). However, the

elements of the basis $\mathcal{B}$ are not guaranteed to be nonnegative vectors.

Ensuring nonnegativity of the basis. The second part of the algorithm, the loop on lines 7-13, modifies the elements of the basis. It gradually makes all elements nonnegative, while at the same time it keeps the coordinates of vector $\mathbf{v}$, corresponding to the current basis, positive. Given an element of the basis $\mathbf{b_i}$ such that its j-th element is negative, the algorithm replaces the element $\mathbf{b_i}$ with a new element $\mathbf{b'_i} := \mathbf{b_i} + \frac{-\mathbf{b}_{\mathbf{i}\,j}}{\mathbf{v}_j}\mathbf{v}$. After replacing $\mathbf{b_i}$ with $\mathbf{b'_i}$, the resulting set of vectors is still a basis of $ker(M)$.

Lemma 3 *The set of vectors $\mathcal{B}' = (\mathcal{B} \setminus \{\mathbf{b_i}\}) \cup \{\mathbf{b'_i}\}$ is a basis of $ker(M)$.*

Proof We show that $\mathbf{b_i}$ can be expressed as a linear combination of vectors from $\mathcal{B}'$. This is sufficient to show that $\mathcal{B}'$ consists of linearly independent vectors and that it generates $ker(M)$. Let us denote the constant $\frac{-\mathbf{b}_{\mathbf{i}\,j}}{\mathbf{v}_j}$ as K and note that $K > 0$ since $\mathbf{v}_j > 0$ and $\mathbf{b}_{\mathbf{i}\,j} < 0$. We first express $\mathbf{b'_i}$ as

$$\begin{aligned}\mathbf{b'_i} &= \mathbf{b_i} + K\mathbf{v} = \mathbf{b_i} + K\sum_{f=1}^{n}\alpha_f\mathbf{b_f}\\ &= \mathbf{b_i}(1 + K\alpha_i) + K\sum_{f\neq i}\alpha_f\mathbf{b_f}\end{aligned}$$

and now $\mathbf{b_i}$ can be expressed as a linear combination of elements of $\mathcal{B}'$:

$$\mathbf{b_i}(1 + K\alpha_i) = \mathbf{b'_i} - K\sum_{f\neq i}\alpha_f\mathbf{b_f}$$

$$\mathbf{b_i} = \frac{\mathbf{b'_i} + \sum_{f\neq i} -K\alpha_f\mathbf{b_f}}{1 + K\alpha_i}$$

□

After this replacement, (at least) one negative value has been successfully eliminated: as $K > 0$ and $\mathbf{v} > \mathbf{0}$, it follows that $\mathbf{b'_i} > \mathbf{b_i}$ and $\mathbf{b'}_{\mathbf{i}\,j} = 0$.

As the last step, we show that the new coordinates of $\mathbf{v}$ (with respect to the new basis) are still positive.

Lemma 4 *Let $\boldsymbol{\alpha}'$ denote the coordinates of $\mathbf{v}$ with respect to the new basis $\mathcal{B}'$. Then, $\boldsymbol{\alpha}' > \mathbf{0}$.*

Proof First, consider the result of a linear combination of the new basis $\mathcal{B}'$ with the old coefficients $\boldsymbol{\alpha}$:

$$\begin{aligned}&\alpha_1\mathbf{b_1} + \ldots + \alpha_i\mathbf{b'_i} + \ldots + \alpha_n\mathbf{b_n} = \sum_{f=1}^{n}\alpha_f\mathbf{b_f} + \alpha_i K\mathbf{v}\\ &= \mathbf{v} + \alpha_i K\mathbf{v} = \mathbf{v}(1 + \alpha_i K)\end{aligned}$$

Now, set $C := 1 + \alpha_i K$ and note that $C > 1$ since $K > 0$ and $\alpha_i > 0$. It follows that

$$\mathbf{v} = \frac{\alpha_1}{C}\mathbf{b_1} + \ldots + \frac{\alpha_i}{C}\mathbf{b'_i} + \ldots + \frac{\alpha_n}{C}\mathbf{b_n}$$

and that $\boldsymbol{\alpha}' = \frac{\boldsymbol{\alpha}}{C}$ is the vector of coordinates of $\mathbf{v}$ with respect to the new basis $\mathcal{B}'$. Since $\boldsymbol{\alpha} > \mathbf{0}$ and $C > 0$, it follows that $\boldsymbol{\alpha}' > \mathbf{0}$ as required. □

We have shown that the loop on lines 7-13 preserves the invariant that the coordinates of $\mathbf{v}$ with respect to the current basis are all positive (lines 11,12) and that each iteration decreases the number of negative values of the basis vectors. As a result, Algorithm 1 terminates and returns a decomposition of the input vector $\mathbf{v}$ of size equal to the nullity of the input matrix M.

We first simulate the run of the algorithm on an example, then discuss its complexity and finally compare it to other approaches for computing interpolants as a conjunction of inequalities.

Example 2 Consider an unsatisfiable system of inequalities $A \wedge B$ where $A = \{x_1 + x_2 \leq 0, -x_1 + x_3 \leq 0, x_1 + x_4 \leq 0, -x_1 + x_5 \leq 0\}$ and $B = \{-x_2 - x_3 - x_4 - x_5 \leq -1\}$. The vector of Farkas coefficients witnessing the unsatisfiability of $A \wedge B$ is $\mathbf{k} = \begin{pmatrix}1 & 1 & 1 & 1 & 1\end{pmatrix}^\intercal$ and its restriction to A-part is $\mathbf{k_A} = \begin{pmatrix}1 & 1 & 1 & 1\end{pmatrix}^\intercal$. The only A-local variable is x_1, so the matrix of A-local coefficients is $L^\intercal = \begin{pmatrix}1 & -1 & 1 & -1\end{pmatrix}$. We simulate the run of Algorithm 1 on $\mathbf{k_A}$ and $L^\intercal$: since $L^\intercal$ is already in RREF, nothing changes on line 1. Now, the rank of $L^\intercal$ is 1 and it has 4 columns, and thus, its nullity is 3 and we can compute a decomposition of $\mathbf{k_A}$ of size 3. The first column of $L^\intercal$ is pivot, while the other three columns are free. The computation of the initial basis of $ker(L^\intercal)$ (line 4) yields three vectors:

$$b_1 = \begin{pmatrix}1\\1\\0\\0\end{pmatrix}, \quad b_2 = \begin{pmatrix}-1\\0\\1\\0\end{pmatrix}, \quad b_3 = \begin{pmatrix}1\\0\\0\\1\end{pmatrix}.$$

The coordinates of $\mathbf{k_A}$ with respect to this basis are $\boldsymbol{\alpha} = \begin{pmatrix}1 & 1 & 1\end{pmatrix}^\intercal$. As $\mathbf{b_2}_1 < 0$, we enter the loop on line 6 where the new vector $\mathbf{b'_2}$ is computed as $\mathbf{b'_2} = \mathbf{b_2} + \mathbf{k_A} = \begin{pmatrix}0 & 1 & 2 & 1\end{pmatrix}^\intercal$. Then, the coordinates are divided by a constant $C = 2$ to obtain the new coordinates $\boldsymbol{\alpha} = \begin{pmatrix}1/2 & 1/2 & 1/2\end{pmatrix}^\intercal$. Since there are no more negative elements in the vectors of the basis, the decomposition $\mathbf{k_A} = 1/2 * \begin{pmatrix}1 & 1 & 0 & 0\end{pmatrix} + 1/2 * \begin{pmatrix}0 & 1 & 2 & 1\end{pmatrix} + 1/2 * \begin{pmatrix}1 & 0 & 0 & 1\end{pmatrix}$ is returned. This decomposition results in the decomposed interpolant

$$\begin{aligned}I^{Dec} = \ &(x_2 + x_3 \leq 0)\\ &\wedge(x_3 + 2x_4 + x_5 \leq 0) \wedge (x_2 + x_5 \leq 0).\end{aligned}$$

Complexity of Algorithm 1. Considering the matrix of A-local coefficients L for m inequalities and l A-local variables, the algorithm runs on matrix $M = L^\intercal$ with m columns and l rows. When the transformation of M to RREF is done by Gauss–Jordan elimination, it needs to perform $\mathcal{O}(m^2 l)$ arithmetic operations. After the transformation, the number of (nonzero) rows is r, which is the rank of M, and we know that $r \leq l$. With n denoting the nullity of M, rank–nullity theorem implies that $r + n = m$ and consequently that $n < m$. The complexity of the computation of an initial basis is $\mathcal{O}(nm)$ since we are computing n basis vectors, each of size m. Determining the value for every element of each basis vector is immediate: it is 0 or 1 for positions corresponding to the free columns, and it is a negated coefficient from RREF(M) for positions corresponding to the pivot columns, see Eq. (7). Finally, one iteration of the loop that ensures nonnegativity of the basis needs just $\mathcal{O}(m)$ arithmetic operations and the termination can be ensured after $\mathcal{O}(n)$ iteration. To see this, note that a basis vector $\mathbf{b_i}$ can be made nonnegative in one iteration when the index j is used that maximizes $\frac{-\mathbf{b_i}_j}{\mathbf{v}_j}$. The whole loop thus requires $\mathcal{O}(nm)$ arithmetic operations. The complexity of the algorithm is thus dominated by the first part—computing RREF of the input matrix.

6.3 Comparison with other approaches

Given an unsatisfiable system of inequalities (A, B), Cimatti et al. [11] recognized two extreme points in the spectrum of possible interpolants. On one side, there is the Farkas interpolant in the form of single inequality obtained as a weighted sum of inequalities from A with weights given by Farkas coefficients. On the other side, it is possible to employ quantifier elimination to compute the strongest possible interpolant for (A, B) which will result in a conjunction of inequalities (if possible). If all A-local variables are existentially quantified in A and eliminated, then this is guaranteed to yield an interpolant. However, as Cimatti et al. note, quantifier elimination is potentially a very expensive operation.[3] Therefore, they propose modifications to the procedure computing the interpolant from the proof of unsatisfiability. The observation they make is that the only purpose of the summation of inequalities when traversing the proof is to eliminate A-local variables. If the leaves of the proof do not contain A-local variables, no summation is needed, and the conjunction of the inequalities in the leaves is already an interpolant. This corresponds to our notion of *trivial elements* of the decomposition. Based on this observation, they proposed a modification to the proof-based algorithm that performs *only the summations that are necessary for eliminating A-local variables*.

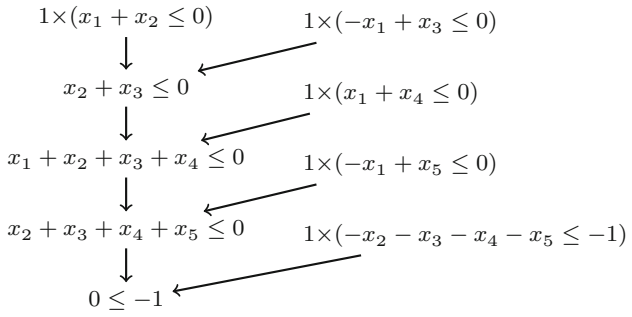

Fig. 2 Proof of unsatisfiability of the system from Example 2

Example 3 Consider the unsatisfiable system of inequalities from Example 2. Figure 2 shows a possible proof of unsatisfiability according to the description of [11].

The computation of Farkas interpolant as described by Eq. (2) can be simulated by replacing the leaves from B with $0 \leq 0$. The resulting Farkas interpolant is

$$I^F = x_2 + x_3 + x_4 + x_5 \leq 0.$$

Applying the modification from [11] avoids one unnecessary sum and results in an interpolant

$$I^M = (x_2 + x_3 \leq 0) \wedge (x_4 + x_5 \leq 0).^4$$

As seen in Example 2, our approach yields interpolant with three conjuncts

$$I^{Dec} = (x_2 + x_3 \leq 0) \wedge (x_3 + 2x_4 + x_5 \leq 0) \wedge (x_2 + x_5 \leq 0).$$

Finally, existentially quantifying x_1 in A and eliminating this quantifier yield interpolant with four conjuncts

$$I^{QE} = (x_2 + x_3 \leq 0) \wedge (x_2 + x_5 \leq 0) \wedge (x_3 + x_4 \leq 0) \wedge (x_4 + x_5 \leq 0).$$

Note that I^{QE} is the strongest and I^F is the weakest interpolant in this quadruple, while I^M and I^{Dec} are incomparable in terms of logical strength. However, the advantage of our algorithm is that even though its result depends on the order of the inequalities (the order of columns of $L^\intercal$), it guarantees to find a decomposition of size 3 in our example. If the first and third inequalities are switched, the decomposed interpolant

[3] Even when restricted to conjunction of inequalities, as is our case. For example, in Fourier–Motzkin procedure eliminating one variable can increase the number of inequalities from m to $m^2/4$ in the worst case. Thus, eliminating n variables increases the number of inequalities to $4(\frac{m}{4})^{2^n}$ in the worst case.

[4] This is indeed the interpolant computed by MATHSAT 5.6.0

computed by Algorithm 1 is

$$I^{Dec'} = (x_4 + x_3 \leq 0) \\ \wedge (x_3 + 2x_2 + x_5 \leq 0) \wedge (x_4 + x_5 \leq 0)$$

while if the first and second inequalities are switched, the computed interpolant is

$$I^{Dec''} = (x_2 + x_4 + 2x_5 \leq 0) \\ \wedge (x_3 + x_4 \leq 0) \wedge (x_3 + x_2 \leq 0).$$

On the other hand, the approach of [11] is, in some sense, even more sensitive to the order of the input inequalities (the shape of the proof) since the order can influence the size of the decomposition. If the second and the third inequalities are switched, then their approach does not detect the opportunity for decomposition and returns the Farkas interpolant I^F. Our algorithm in this situation returns an interpolant equivalent to I^{Dec}.

7 Experiments

We have implemented the computation of decomposed interpolants and their duals using Algorithm 1 in our SMT solver OpenSMT [23], which already provided a variety of interpolation algorithms for propositional logic [24,34], theory of uninterpreted functions [3] and theory of linear real arithmetic [4].

We evaluated the effect of decomposed interpolants in a model-checking scenario using the model checker SALLY [26] with YICES [17] for satisfiability queries and OpenSMT for interpolation queries[5]. We experimented with four LRA interpolation algorithms: the original interpolation algorithms based on Farkas' lemma, (i) Itp_F and (ii) Itp'_F, and the interpolation algorithm computing decomposed interpolants, (iii) Itp_{DI}, and (iv) Itp'_{DI}. OpenSMT computes interpolants from the proof of unsatisfiability. In this approach, the interpolants computed for LRA conflicts are combined based on interpolation rules for propositional logic and the structure of the proof. In our experiments, we fixed the propositional part of the interpolation algorithm to use McMillan's interpolation rules [30]. We split our analysis of the experiments into two parts. In Sec. 7.1, we analyse the performance of the model checker using different LRA interpolation algorithms. We focus specifically on a detailed comparison of Itp_F and Itp_{DI}, i.e. the default algorithm and our proposed algorithm. In Sec. 7.2, we analyse the performance of a portfolio of interpolation algorithms and measure the contribution of our proposed algorithm. For comparison, we also run a version of SALLY using MathSAT as the interpolation engine and compare to the contribution of the decomposing algorithm proposed in [11].

The experiments were run on a large set of benchmarks consisting of several problem sets related to fault-tolerant algorithms (**azadmanesh**, **approxagree**, **om**, **hacms**, **misc**, **ttesynchro**, **ttastartup**,**unifapprox**), software model checking (**cav12**, **ctigar**), simple concurrent programs (**conc**), and a lock-free hash table (**lfht**). A benchmark suite of the KIND model checker is also included (**lustre**). Each benchmark is a transition system with formulas characterizing initial states, a transition relation and a property that should hold. SALLY can finish with two possible answers (or run out of resources with no answer): *valid* means the property holds and an invariant implying the property has been found; *invalid* means the property does not hold and a counterexample leading to a state where the property does not hold has been found. In the plots, we denote the answers as + and ∘, respectively. The benchmarks were run on Linux machines with the Intel E5-2650 v3 processor (2.3 GHz) and 64GB of memory. Each benchmark was restricted to 600 seconds of running time and to 4GB of memory.

7.1 Comparing individual configurations

Table 1 presents the results of the model checker's runs using different interpolation algorithms. The results are summarized *by category* with the name of the category and the number of corresponding benchmarks in the first column. The two columns per interpolation algorithm show the number of benchmarks solved successfully (validated/invalidated) within the resource limits and the total running time for the *solved* benchmarks.

The results suggest that Itp_F interpolation algorithm achieves the best result overall. However, there are certain cases where Itp_{DI} is faring better, for example, the **lfht** category. Before we present a more thorough comparison between these two algorithms, we note that the configuration using Itp'_{DI}, which computes the weakest interpolants, performs very poorly compared to the others. Closer inspection revealed that it did not solve any benchmarks not solvable by other configurations. It did solve a few benchmarks faster than others, but the improvement was negligible. On the other hand, the overall drop in performance is large. We conclude that computing very weak interpolants is a bad strategy in this model-checking scenario.

As mentioned before, the results summarized in Table 1 suggest that Itp_F performs better than Itp_{DI} overall. However, a closer look reveals that the situation is more complicated. Figure 3 illustrates a direct comparison between these two algorithms. Each point represents one benchmark, x-axis corresponds to the runtime (in seconds) of SALLY using Itp_F as

[5] Detailed description of the set-up and specifications of the experiments, together with all the results, can be found at http://verify.inf.usi.ch/content/decomposed-interpolants

Table 1 Performance of SALLY using different interpolation algorithms of OPENSMT

Problem set	Itp_F solved (V/I)	$\sum$ time(s)	Itp'_F solved (V/I)	$\sum$ time(s)	Itp_{DI} solved (V/I)	$\sum$ time(s)	Itp'_{DI} solved (V/I)	$\sum$ time(s)
approxagree (9)	9 (8/1)	127	9 (8/1)	138	**9 (8/1)**	**106**	9 (8/1)	126
azadmanesh (20)	**20 (17/3)**	**418**	20 (17/3)	639	20 (17/3)	422	20 (17/3)	1202
cav12 (99)	**68 (48/20)**	**2097**	67 (48/19)	2580	66 (48/18)	1441	66 (47/19)	2446
conc (6)	3 (3/0)	20	3 (3/0)	22	**5 (5/0)**	**313**	3 (3/0)	21
ctigar (110)	**74 (54/20)**	**3066**	70 (50/20)	1919	71 (51/20)	3077	58 (39/19)	1701
hacms (5)	2 (2/0)	332	**2 (1/1)**	**251**	1 (1/0)	5	1 (1/0)	5
lfht (27)	17 (17/0)	319	18 (18/0)	448	**22 (22/0)**	**2784**	16 (16/0)	26
lustre (790)	**773 (437/336)**	**3530**	769 (436/333)	3,180	766 (433/333)	3990	741 (416/325)	2021
misc (10)	8 (7/1)	154	8 (7/1)	127	**9 (7/2)**	**57**	9 (7/2)	888
om (9)	9 (7/2)	6	**9 (7/2)**	**4**	9 (7/2)	6	**9 (7/2)**	**4**
ttastartup (3)	**2 (1/1)**	**325**	1 (1/0)	7	1 (1/0)	11	1 (1/0)	15
ttesynchro (6)	**6 (3/3)**	**10**	6 (3/3)	11	6 (3/3)	13	6 (3/3)	13
unifapprox (11)	11 (8/3)	71	**11 (8/3)**	**64**	11 (8/3)	71	11 (8/3)	448
Total (1105)	**1002 (612/390)**	**10,475**	993 (607/386)	9390	996 (611/385)	12,296	950 (573/377)	8916

Bold font emphasizes the best interpolation algorithm for each problem set

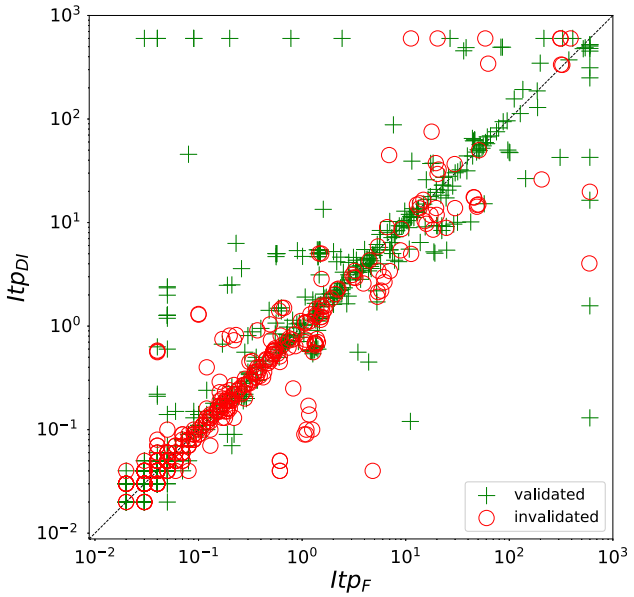

Fig. 3 Evaluation of the decomposed interpolants in model checking scenario: comparison of performance of SALLY using OPENSMT with different interpolation procedures, Itp_F and Itp_{DI}

the interpolation algorithm in OPENSMT, and y-axis corresponds to the runtime of SALLY using Itp_{DI}. The direct comparison shows that in some cases, the use of decomposed interpolants outperforms the original procedure, sometimes by an order of magnitude. Even though Itp_{DI} solved 6 benchmarks less than Itp_F, it still managed to solve 12 benchmarks that Itp_F was not able to solve within the resource limits. Moreover, on a common set of non-trivial (runtime at least 10 seconds) solved benchmarks, it improved the performance by more than 10% on 45 benchmarks (out of 116 such benchmarks).

During the evaluation, we realized that a small modification in the SMT solver sometimes had a huge effect on the performance of the model checker. It made previously unsolved instance easily solvable or the other way around. To confirm that using Itp_{DI} is indeed better than using Itp_F for particular benchmarks, we ran an additional set of experiments. For each of the 12 benchmarks solved by Itp_{DI} but not solved by Itp_F, we ran the model checker 100 times, each time with a different random seed for the interpolating solver. The results are summarized in Table 2. For each of the two configurations, the table reports how many runs (out of 100) of the model checker finished successfully within the resource limits and the average time of the successful runs. This experiment demonstrates that there are indeed benchmarks where the decomposition is necessary, while using the original Farkas algorithm leads to divergence. In other cases, the use of decomposed interpolants leads to a higher chance of a successful result and/or better runtime of the model checker. Note that these benchmarks were picked deliberately to confirm that Itp_{DI} performs better on them than Itp_F, based on our experiments on the whole benchmark set.

For the final aspect of the direct comparison of Itp_F and Itp_{DI}, we collected statistics from the runs of SALLY with Itp_{DI} about how often Itp_{DI} manages to decompose the vector of Farkas coefficients, thus returning a different interpolant than Itp_F would. These results are summarized in Table 3. The column pwd reports the *number* of benchmarks with at least a *single* decomposition (any; with at least one trivial element; with at least one non-trivial element). The next column ("#non-triv. LRA itps") reports the total number of interpo-

Table 2 Aggregated results from 100 runs of the model checker on selected benchmarks

benchmark	Itp_F solved	avg. time	Itp_{DI} solved	avg. time
fib_benc_safe_v1	0	–	100	46.5
fib_benc_safe_v2	0	–	100	0.01
dillig01.c	0	–	100	0.1
dillig03.c	0	–	100	0.1
lifnat.c	17	510	29	471
lfht_2_mini_cleaned.prop1	21	362	57	344
lfht_2_mini_lemma5c	18	257	69	293
lfht_2_mini_lemma5e	0	–	30	347
lfht_2_mini_lemma5f	1	188	39	363
lfht_2_mini_lemma5g	22	284	47	311
DRAGON_12_e2_1618_e2_138	99	25	100	19
mvs_with_timeouts3	73	251	98	64

Table 3 Interpolation statistics—pwd stands for "Number of problems with at least one decomposition". The numbers in parentheses denote "Decompositions with trivial and with non-trivial elements" (trivial/non-trivial)

	Itp_{DI} Problem set	pwd	#non-triv. LRA itps	#decomp. itps
approxagree (9)	1 (1/1)	7	7	(4/3)
azadmanesh (20)	0 (0/0)	1818	0	(0/0)
cav12 (99)	40 (30/29)	707,414	6464	(747/5719)
conc (6)	3 (3/3)	39,135	25,603	(4030/21,033)
ctigar (110)	70 (58/69)	4,064,827	1,106,642	(61,371/1,049,904)
hacms (5)	5 (5/5)	424,532	32,331	(3,628/28,703)
lfht (27)	14 (14/14)	786,837	126,568	(5464/121,104)
lustre (790)	327 (96/299)	2,916,829	2,001,503	(9,115/2,001,058)
misc (10)	8 (7/8)	59,266	12,054	(2363/10,024)
om (9)	6 (6/0)	974	380	(380/0)
ttastartup (3)	3 (2/3)	117,303	12,165	(240/11,925)
ttesynchro (6)	4 (4/4)	90	90	(90/69)
unifapprox (11)	1 (1/0)	1	1	(1/0)

lation problems for theory conflict, excluding those without even theoretical possibility for decomposition. There is no possibility for decomposition if all inequalities are from one part of the problem (resulting in trivial interpolants, either $\top$ or $\bot$) or there is only a single inequality in the A-part (trivially yielding an interpolant equal to that inequality). The last column reports the number of successfully decomposed interpolants (with at least one trivial element; with at least one non-trivial element). Note that it can happen that a successful decomposition contains both trivial and non-trivial elements. We see that at least one decomposition was possible in only less than half of all the benchmarks. This explains why there are many points on the diagonal in Fig. 3. On the other hand, it shows that the test for the *possibility* of decomposition is cheap and does not represent a significant overhead. Another conclusion we can draw is that when the structure of the benchmark enables decomposition, it can often be discovered in many theory conflicts that appear during the solving.

7.2 Analysis of the portfolio

In this part, we present yet another way to measure the contribution of the decomposed interpolants: the contribution to the virtual best configuration. We consider a virtual portfolio consisting of configurations of SALLY using different interpolation algorithms of OPENSMT. In addition, we also consider a separate virtual portfolio of configurations of SALLY using MATHSAT. The result of a virtual portfolio on a benchmark is the best result achieved by any of the configurations of the portfolio. As noted before, the configuration using Itp'_{DI} performed quite poorly on our benchmarks. Since MATHSAT can compute Farkas interpolants and its duals, and restricted form of decomposed interpolants but not its dual, we also exclude Itp'_{DI} from the portfolio of OPENSMT's configurations, with minimal impact on the performance. We denote the heuristic for computing decompositions described in [11] and available in MATHSAT as Itp_M. We use the number of

Table 4 Contribution of the configurations to their respective portfolios

	Configuration	#uniquely solved	PAR-2 regret	
OpenSMT	Itp_F	4	4046	3.5%
	Itp'_F	3	4586	3.9%
	Itp_{DI}	10	10245	8.8%
MathSAT	Itp_F	0	260	0.2%
	Itp'_F	3	3594	3.3%
	Itp_M	6	7754	7%

solved instances and PAR-2 score as a metric of measuring the performance. PAR-2 score is computed as the sum of runtime on solved instances plus two times the timeout for each unsolved instance. Finally, for each configuration we compute the number of uniquely solved instances (not solved by any other configuration in the portfolio) and *regret*, i.e. how much would the PAR-2 score of the portfolio worsen, if that particular configuration was excluded from the portfolio. The results are summarized in Table 4. Note that OpenSMT and MathSAT portfolios are considered separately.

OpenSMT configuration portfolio is able to solve 1,017 benchmarks with PAR-2 score 116,117. MathSAT configuration portfolio is able to solve 1,018 benchmarks with PAR-2 score 110,356. We hypothesize that the better performance of MathSAT can be at least partially attributed to the fact that it supports interpolation in combination with incremental solving while OpenSMT does not. In both portfolios, the ability to compute decomposed interpolants (even in the restricted form) significantly improves the performance of the portfolio. We also see that the contribution of our algorithm based on methods from linear algebra to OpenSMT portfolio is slightly larger than the contribution of the heuristic Itp_M to the MathSAT portfolio. Additionally, our algorithm solves more instances uniquely within its portfolio. Interestingly, the contribution of the configuration computing Farkas interpolants is non-trivial in OpenSMT, but almost non-existent in MathSAT. Our hypothesis is that Itp_M, compared to Itp_{DI}, decomposes less often and the decompositions are of smaller size (e.g. in the situation from Example 3). This would mean that the interpolants from Itp_M are more often similar (or even identical) to Farkas interpolants, which would make the MathSAT portfolio less diverse than the OpenSMT portfolio.

8 Conclusion

In this paper, we have presented a new interpolation algorithm for linear real arithmetic that generalizes the interpolation algorithm based on Farkas' lemma used in modern SMT solvers. We showed that the algorithm is able to compute interpolants in the form of a *conjunction* of inequalities that are logically stronger than the single inequality returned by the original approach. This becomes useful in the IC3-style model-checking algorithms where Farkas interpolants have been shown to be a source of incompleteness. In our experiments, we have demonstrated that the opportunity to decompose Farkas interpolants frequently occurs in practice and that the decomposition often leads to (i) lower solving time and, in some cases, to (ii) solving a problem not solvable by the previous approach.

As the next steps, we plan to investigate how to automatically determine what kind of interpolant would be more useful for the current interpolation query in IC3-style model-checking algorithms. We also plan to investigate other uses of interpolation in model checking where stronger (or weaker) interpolants are desirable [35].

Acknowledgements We would like to thank Dejan Jovanović for providing the benchmarks and for the help with integrating OpenSMT into SALLY. We would also like to thank the anonymous reviewers for their time and helpful comments.

Funding Open Access funding provided by Universitá della Svizzera italiana.

References

1. Albarghouthi, A., McMillan, K.L.: Beautiful interpolants. In: Sharygina, N., Veith, H. (eds.) CAV 2013, LNCS, vol. 8044, pp. 313–329. Springer, Heidelberg (2013)
2. Alt, L., Fedyukovich, G., Hyvärinen, A.E.J., Sharygina, N.: A proof-sensitive approach for small propositional interpolants. In: Gurfinkel, A., Seshia, S.A. (eds.) VSTTE 2015, LNCS, vol. 9593, pp. 1–18. Springer, Cham (2016)

3. Alt, L., Hyvärinen, A.E.J., Asadi, S., Sharygina, N.: Duality-based interpolation for quantifier-free equalities and uninterpreted functions. In: Stewart, D., Weissenbacher, G. (eds.) FMCAD 2017, pp. 39–46. IEEE (2017)
4. Alt, L., Hyvärinen, A.E.J., Sharygina, N.: LRA interpolants from no man's land. In: Strichman, O., Tzoref-Brill, R. (eds.) HVC 2017, LNCS, vol. 10629, pp. 195–210. Springer, Cham (2017)
5. Andrilli, S., Hecker, D.: Elementary Linear Algebra, 5th edn. Academic Press, Cambridge (2016)
6. Barrett, C., de Moura, L.M., Ranise, S., Stump, A., Tinelli, C.: The SMT-LIB initiative and the rise of SMT. In: Barner, S., Harris, I.G., Kroening, D., Raz, O. (eds.) HVC 2010 *LNCS*, vol. 6504, p. 3. Springer, Heidelberg (2011)
7. Barrett, C., Sebastiani, R., Seshia, S., Tinelli, C.: Satisfiability modulo theories, Frontiers in Artificial Intelligence and Applications, 1st edition. vol. 185, pp. 825–885 (2009)
8. Blicha, M., Hyvärinen, A.E.J., Kofroň, J., Sharygina, N.: Decomposing Farkas interpolants. In: Vojnar, T., Zhang, L. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science, p. 3–20. Springer International Publishing, Berlin (2019). https://doi.org/10.1007/978-3-030-17462-0_1
9. Bradley, A.R.: SAT-based model checking without unrolling. In: Jhala, R., Schmidt, D.A. (eds.) VMCAI 2011, LNCS, vol. 6538, pp. 70–87. Springer, Berlin (2011)
10. Bradley, A.R.: Understanding IC3. In: Cimatti, A., Sebastiani, R. (eds.) Theory and Applications of Satisfiability Testing - SAT 2012, pp. 1–14. Springer, Berlin Heidelberg (2012)
11. Cimatti, A., Griggio, A., Sebastiani, R.: Efficient generation of Craig interpolants in satisfiability modulo theories. ACM Trans. Comput. Logic **12**(1), 7:1–7:54 (2010)
12. Craig, W.: Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. J. Symb. Logic **22**(3), 269–285 (1957)
13. Davis, M., Logemann, G., Loveland, D.W.: A machine program for theorem-proving. Commun. ACM **5**(7), 394–397 (1962)
14. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM **7**(3), 201–215 (1960)
15. Detlefs, D., Nelson, G., Saxe, J.B.: Simplify: a theorem prover for program checking. J. ACM **52**(3), 365–473 (2005)
16. D'Silva, V., Kroening, D., Purandare, M., Weissenbacher, G.: Interpolant strength. In: VMCAI 2010. LNCS, vol. 5944, pp. 129–145. Springer, Berlin (2010)
17. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) CAV 2014, LNCS, vol. 8559, pp. 737–744. Springer, Berlin (2014)
18. Dutertre, B., de Moura, L.M.: A fast linear-arithmetic solver for DPLL(T). In: Ball, T., Jones, R.B. (eds.) CAV 2006, LNCS, vol. 4144, pp. 81–94. Springer, Berlin (2006)
19. Farkas, G.: A Fourier-féle mechanikai elv alkalmazásai (Hungarian) [On the applications of the mechanical principle of Fourier] (1894)
20. Gurfinkel, A., Kahsai, T., Komuravelli, A., Navas, J.A.: The SeaHorn verification framework. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015, LNCS, vol. 9206, pp. 343–361. Springer, Cham (2015)
21. Gurfinkel, A., Rollini, S.F., Sharygina, N.: Interpolation properties and SAT-based model checking. In: Van Hung, D., Ogawa, M. (eds.) ATVA 2013, pp. 255–271. Springer, Cham (2013)
22. Huberman, B.A., Lukose, R.M., Hogg, T.: An economics approach to hard computational problems. Science **275**(5296), 51–54 (1997)
23. Hyvärinen, A.E.J., Marescotti, M., Alt, L., Sharygina, N.: OpenSMT2: An SMT solver for multi-core and cloud computing. In: Creignou, N., Le Berre, D. (eds.) SAT 2016, LNCS, vol. 9710, pp. 547–553. Springer, Cham (2016)
24. Jančík, P., Alt, L., Fedyukovich, G., Hyvärinen, A.E.J., Kofroň, J., Sharygina, N.: PVAIR: Partial variable assignment interpolator. In: Stevens, P., Wasowski, A. (eds.) FASE 2016. Springer, Heidelberg (2016)
25. Jančík, P., Kofroň, J., Rollini, S.F., Sharygina, N.: On interpolants and variable assignments. In: FMCAD 2014, pp. 123–130. IEEE (2014)
26. Jovanović, D., Dutertre, B.: Property-directed k-induction. In: 2016 Formal Methods in Computer-Aided Design (FMCAD), pp. 85–92 (2016)
27. Komuravelli, A., Gurfinkel, A., Chaki, S.: SMT-based model checking for recursive programs. In: Biere, A., Bloem, R. (eds.) CAV 2014, pp. 17–34. Springer, Cham (2014)
28. Korovin, K., Tsiskaridze, N., Voronkov, A.: Conflict resolution. In: Gent, I.P. (ed.) CP 2009, pp. 509–523. Springer, Heidelberg (2009)
29. Manna, Z., Pnueli, A.: Temporal Verification of Reactive Systems: Safety. Springer, Heidelberg (1995)
30. McMillan, K.L.: Interpolation and SAT-based model checking. In: Hunt, W.A., Somenzi, F. (eds.) CAV 2013, pp. 1–13. Springer, Heidelberg (2003)
31. McMillan, K.L.: An interpolating theorem prover. Theor. Comput. Sci. **345**(1), 101–121 (2005)
32. Nieuwenhuis, R., Oliveras, A.: Proof-producing congruence closure. In: Giesl, J. (ed.) RTA 2005, LNCS, vol. 3467, pp. 453–468. Springer, Berlin (2005)
33. Pudlák, P.: Lower bounds for resolution and cutting plane proofs and monotone computations. J. Symb. Logic **62**(3), 981–998 (1997)
34. Rollini, S.F., Alt, L., Fedyukovich, G., Hyvärinen, A.E.J., Sharygina, N.: PeRIPLO: a framework for producing effective interpolants in SAT-based software verification. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) LPAR 2013, LNCS, vol. 8312, pp. 683–693. Springer, Heidelberg (2013)
35. Rollini, S.F., Šerý, O., Sharygina, N.: Leveraging interpolant strength in model checking. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012, LNCS, vol. 7358, pp. 193–209. Springer, Heidelberg (2012)
36. Rybalchenko, A., Sofronie-Stokkermans, V.: Constraint solving for interpolation. In: Cook, B., Podelski, A. (eds.) VMCAI 2007, LNCS, vol. 4349, pp. 346–362. Springer, Heidelberg (2007)
37. Schindler, T., Jovanović, D.: Selfless interpolation for infinite-state model checking. In: Dillig, I., Palsberg, J. (eds.) VMCAI 2018, pp. 495–515. Springer, Cham (2018)
38. Scholl, C., Pigorsch, F., Disch, S., Althaus, E.: Simple interpolants for linear arithmetic. In: DATE 2014, pp. 1–6. IEEE (2014)
39. Schrijver, A.: Theory of Linear and Integer Programming. John Wiley and Sons Inc, New York (1998)
40. Sery, O., Fedyukovich, G., Sharygina, N.: Incremental upgrade checking by means of interpolation-based function summaries. In: 2012 Formal Methods in Computer-Aided Design (FMCAD), pp. 114–121 (2012)
41. Sheeran, M., Singh, S., Stålmarck, G.: Checking safety properties using induction and a SAT-solver. In: Hunt, W.A., Johnson, S.D. (eds.) FMCAD 2000, pp. 127–144. Springer, Heidelberg (2000)
42. Silva, J.P.M., Sakallah, K.A.: GRASP: a search algorithm for propositional satisfiability. IEEE Trans. Comput. **48**(5), 506–521 (1999)