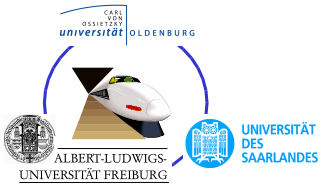

AVACS – Automatic Verification and Analysis of Complex Systems

REPORTS

of SFB/TR 14 AVACS

Editors: Board of SFB/TR 14 AVACS

Constraint Solving for Interpolation

by

Andrey Rybalchenko Viorica Sofronie-Stokkermans

AVACS Technical Report No. 56
December 2009
ISSN: 1860-9821

Publisher: Sonderforschungsbereich/Transregio 14 AVACS
(Automatic Verification and Analysis of Complex Systems)
Editors: Bernd Becker, Werner Damm, Martin Fränzle, Ernst-Rüdiger Olderog,
Andreas Podelski, Reinhard Wilhelm
ATRs (AVACS Technical Reports) are freely downloadable from `www.avacs.org`

Copyright © December 2009 by the author(s)
Author(s) contact: Viorica Sofronie-Stokkermans (`vss@avacs.org`).

Constraint Solving for Interpolation *

Andrey Rybalchenko[1] and Viorica Sofronie-Stokkermans[2]

[1] Max Planck Institute for Software Systems
[2] Max Planck Institute for Computer Science

Abstract. Interpolation is an important component of recent methods for program verification. It provides a natural and effective means for computing separation between the sets of 'good' and 'bad' states. The existing algorithms for interpolant generation are proof-based: They require explicit construction of proofs, from which interpolants can be computed. Construction of such proofs is a difficult task. We propose an algorithm for the generation of interpolants for the combined theory of linear arithmetic and uninterpreted function symbols that does not require a priori constructed proofs to derive interpolants. It uses a reduction of the problem to constraint solving in linear arithmetic, which allows application of existing highly optimized Linear Programming solvers in black-box fashion. We provide experimental evidence of the practical applicability of our algorithm.

1 Introduction

Interpolation [8] is an important component of recent methods for program verification. It provides a natural and effective means for computing separation between the sets of 'good' and 'bad' states. Such separations provide a basis for powerful heuristics for the discovery of relevant predicates for predicate abstraction with refinement and for the over-approximation in model checking, see e.g. [9,14,15,16,22,23,24,35].

The applicability of interpolation-based verification methods crucially depends on the employed procedure for interpolant generation. The existing algorithms for interpolant generation are proof-based: They require explicit construction of proofs, from which interpolants can be computed (resolution proofs in propositional logic, proofs for linear inequalities over the reals, or in the combined theory of linear arithmetic with uninterpreted function symbols [19,28,23]). Explicit construction of such proofs is a difficult task, which hinders the practical applicability of interpolants for verification. In fact, the existing tools for the generation of interpolants over linear arithmetic and uninterpreted function symbols only handle the difference bound-fragment of arithmetic constraints [15,23].

* This work is supported in part by the German Research Foundation (DFG) as a part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS), by the German Federal Ministry of Education and Research (BMBF) in the framework of the Verisoft project under grant 01 IS C38.

One of the consequences of this limitation is that no program whose correctness depends on a predicate over three or more variables can be handled by the method described in [11].

We propose an algorithm for the generation of interpolants for the combined theory of linear arithmetic and uninterpreted function symbols that does not require a priori constructed proofs to derive interpolants. It uses a reduction of the problem to constraint solving in linear arithmetic. Thus, the algorithm allows application of existing highly optimized Linear Programming solvers in black-box fashion, which leads to a practical implementation.

The main contributions of the paper are the following.

- First, we describe an algorithm $\text{INTER}_{LI(\mathbb{Q})}$ for the generation of interpolants for linear arithmetic only, which is based on a reduction to constraint solving. The algorithm $\text{INTER}_{LI(\mathbb{Q})}$ has the following advantages:
  - it allows to handle directly strict and non-strict inequalities,
  - it can be implemented using a Linear Programming solver as a black box.
- Second, we present an algorithm $\text{INTER}_{LI(\mathbb{Q})^\Sigma}$ for generating interpolants in combination of linear arithmetic with uninterpreted function symbols, following the hierarchical style of [31,32,33]. It applies the algorithm $\text{INTER}_{LI(\mathbb{Q})}$ as a subroutine.
- We provide experimental evidence of the applicability of this constraint based interpolant generation.

Our implementation is integrated into the predicate discovery procedure of the software verification tools Blast [11] and ARMC [27]. Our experiments with Blast on Windows device drivers provide a direct comparison with the existing tool FOCI [23], and show promising running times in favour of the constraint based approach. Our method can handle systems which pose problems to other interpolation-based provers: It allowed us, for instance, to apply ARMC to verify safety properties of train controller systems [26], which required inference of predicates with both strict and nonstrict inequalities, and it allows us to verify examples that require predicates over up to four variables.

Furthermore, our algorithm $\text{INTER}_{LI(\mathbb{Q})}$ provides a basis for dealing with linear arithmetic in the interpolation procedure CSIsat [1], which is integrated in Blast and has been successfully applied on software verification benchmarks.

Related work: This paper extends [29] with full proofs, elaborate and exhaustive comparison with recent developments in the related work, and presents an improved algorithm for the combined theories, see Figure 3.

Our algorithm differs from the existing methods for (ground) interpolant generation [15,16,23,35,4,21,5,10] in the following key points. Being constraint based, our algorithm does not require a priori constructed proof to derive an interpolant. However, it is possible to construct such a proof using non-negative linear combinations of inequalities computed by our algorithm.

Our method for the synthesis of interpolants for the combined theory of linear arithmetic and uninterpreted function symbols, follows the hierarchical style of [31,32,33], and uses the interpolant construction method for linear arithmetic as a black-box. The algorithm presented in this paper, on which our implementation is based, differs from that in [32,33], being tuned to our constrained based approach.

In fact, our method for generating interpolants for linear arithmetic can be used as a black box procedure also in other contexts, e.g. for the method for constructing interpolants in combinations of theories over disjoint signatures proposed in [35] or for the interpolant generation method for the combinations of linear arithmetic, uninterpreted function symbols, lists, and sets with cardinality constraints, which uses a reduction to the invariant generation in linear arithmetic and uninterpreted function symbols [16]. Conversely, our method for the synthesis of interpolants for the combined theory of linear arithmetic and uninterpreted function symbols can use any method for interpolant construction for linear arithmetic, including methods which extract interpolants from proofs.

The "split" prover [15] applies a sequent calculus for the synthesis of interpolants whose linear arithmetic part is restricted to difference bounds constraints with a user-defined constraint on the bound. In contrast, our implementation is constraint-based and handles full linear arithmetic. Its extension to accommodate user-defined constraints is a subject of ongoing work.

Our algorithm constructs linear arithmetic interpolants in a way similar to constraint-based invariant and ranking function generation methods based on Farkas' lemma and linear programming, see e.g. [3,6,7]. We use its extension (Motzkin's transposition theorem) to handle strict inequalities. In the terminology of linear programming, interpolants are hyperplanes that separate strictly disjoined convex hulls and contain only common variables of the hulls.

The interpolation procedure CSISAT applies our interpolation algorithm for linear arithmetic and demonstrates its practical efficiency [1].

Finally, in [25,18] interpolants which possibly contain quantifiers are extracted from resolution proofs. The results we present here do not aim at such a general approach, but extensions to more general types of interpolants are interesting for future work.

Structure of the paper: We introduce the necessary preliminaries in Section 2. We describe the algorithm $\text{INTER}_{LI(\mathbb{Q})}$ for the synthesis of constrained interpolants in linear arithmetic in Section 3, and its extension $\text{INTER}_{LI(\mathbb{Q})^E}$ for the handling of uninterpreted function symbols in Section 4. We briefly describe our implementation and experimental results in Section 5.

2 Preliminaries

We introduce the definitions and notation used in the paper.

2.1 Linear constraints over rational and real spaces

In what follows we will use the following notation: We write $Ax < a$ and $Ax \leq a$ for systems (conjunctions) of strict and non-strict inequalities, respectively. We write $Ax \leqslant a$ for a system that may contain inequalities of both kinds. We refer to such systems as *mixed* ones. Given $Ax \leqslant a$, we write $A^{\mathsf{lt}}x < a^{\mathsf{lt}}$ and $A^{\mathsf{le}}x \leq a^{\mathsf{le}}$ for the systems that contain strict and non-strict inequalities from $Ax \leqslant a$, respectively. A row vector λ with m_A elements defines a linear combination of inequalities from $Ax \leqslant a$. The vector λ has sub-vectors λ^{lt} and λ^{le} that correspond to strict and non-strict inequalities from $Ax \leqslant a$, respectively. We have $\lambda A = \lambda^{\mathsf{lt}}A^{\mathsf{lt}} + \lambda^{\mathsf{le}}A^{\mathsf{le}}$ and $\lambda a = \lambda^{\mathsf{lt}}a^{\mathsf{lt}} + \lambda^{\mathsf{le}}a^{\mathsf{le}}$. We write $A_{|k}$ for the k-th column of a matrix A. We write $A^k x \leqslant a^k$ to refer to a system of inequalities with index k. Given a vector i we write i^{T} to denote its transposition.

We note that precise handling of strict inequalities is required for interpolation problems over rationals/reals, which occur in the verification of real time and hybrid systems. Consider the unsatisfiable conjunction $x < 0$ and $x \geq 0$, where x ranges over rationals/reals. The relaxation of $x < 0$ to non-strict inequality $x \leq 0$ leads to the loss of unsatisfiability. The strengthening of $x < 0$ to $x \leq -1$ may result in the interpolant $x \leq -1$ which is not an interpolant for the original problem, since $x < 0$ does not imply $x \leq -1$.

2.2 Extensions with uninterpreted functions

Let Σ be a set of (new) function symbols. Let $\mathcal{T}_0$ be one of the theories $LI(\mathbb{Q})$ (linear rational arithmetic) or $LI(\mathbb{R})$ (linear real arithmetic). with signature $\Pi_0 = (\Sigma_0, \mathsf{Pred})$. We denote by $LI(\mathbb{Q})^{\Sigma}$ the extension of $\mathbb{Q}$ with the uninterpreted function symbols in Σ. $LI(\mathbb{R})^{\Sigma}$ is defined similarly. In what follows, the definitions are given for the case of linear rational arithmetic. Similar definitions can also be given for $LI(\mathbb{R})^{\Sigma}$.

Definition 1 *A model $\mathcal{M} = (M, \{f_{\mathcal{M}}\}_{f \in \Sigma})$ of $LI(\mathbb{Q})^{\Sigma}$ consists of a model M of $LI(\mathbb{Q})$ (with universe $\mathbb{Q}$) and a function $f_{\mathcal{M}} : M^n \to M$ for each $f \in \Sigma$ with arity n. No additional constraints are imposed on the properties of these functions (they are free).*

2.3 Formulae; truth, satisfiability and entailment w.r.t. a theory

For the definitions of terms and first-order formulae over a given signature we refer to any introductory book in first-order logic. A formula is called closed, or a sentence, if it has no free variables. A formula or a term is called ground if it has no occurrences of variables.

Let $\mathcal{T}$ be a theory (that is, a set of models in a given signature Σ). Truth and satisfiability of a first-order formula in a given model are defined in the standard way.

Definition 2 *Let ϕ and ψ be formulae over the signature Σ. We say that:*

- ϕ is true w.r.t. $\mathcal{T}$ *(denoted* $\models_{\mathcal{T}} \phi$*) if* ϕ *is true in all models of* $\mathcal{T}$*;*
- ϕ entails *(or* implies*)* ψ w.r.t. $\mathcal{T}$ *(denoted* $\phi \models_{\mathcal{T}} \psi$*) if* ψ *is true in all models of* $\mathcal{T}$ *in which* ϕ *is true;*
- ϕ is satisfiable w.r.t. $\mathcal{T}$ *if there exists a model of* $\mathcal{T}$ *in which* ϕ *is true.*

If ϕ *is false in all models of* $\mathcal{T}$*, we say that* ϕ *is unsatisfiable.*

Note that ϕ is unsatisfiable iff $\phi \models_{\mathcal{T}} \bot$, where $\bot$ stands for false. $\top$ stands for true.

Remark: In what follows we are concerned with satisfiability of quantifier-free formulae with variables. When checking satisfiability, these variables can be seen as implicitly existentially quantified. In the automated reasoning literature, existential variables are replaced by constants, using skolemization. In what follows we refer to them as variables. However, the notation we chose reminds that these existentially quantified variables can, in fact, be regarded as "constants"; in this case the quantifier-free formulae with variables we consider can be considered to be ground.

2.4 Interpolants

An important result in first-order logic is Craigs Interpolation Theorem [8].

Theorem 3 ([8]) *Let* ϕ, ψ *be closed formulae such that* $\phi \models \psi$*. Then there exists a closed formula* I *containing only predicate symbols, function symbols and constants occurring in both* ϕ *and* ψ *such that* $\phi \models I$ *and* $I \models \psi$*.*

Every formula I satisfying the property in Theorem 3 will be called an interpolant of ϕ and ψ. More generally, we can talk about interpolants w.r.t. a theory and interpolation properties of theories. In what follows, as in [35], we will refer to theories $\mathcal{T}$ with signature Σ in which we distinguish a subset $\Sigma_T \subseteq \Sigma$ of *interpreted symbols*; all the other symbols in $\Sigma \backslash \Sigma_T$ are considered to be *uninterpreted*.

Definition 4 (Theory specific interpolant) *Let* $\mathcal{T}$ *be a theory with signature* Σ*, and* $\Sigma_T \subseteq \Sigma$ *be the set of symbols we consider to be interpreted in* $\mathcal{T}$*. Let* ϕ *and* ψ *be formulae in the signature of* $\mathcal{T}$ *such that* $\phi \models_{\mathcal{T}} \psi$*. We say that a formula* I *is a (theory-specific)* interpolant *of* ϕ *and* ψ *if (i)* $\phi \models_{\mathcal{T}} I$*, (ii)* $I \models_{\mathcal{T}} \psi$*, and (iii)* I *contains interpreted symbols (i.e. any symbol in* Σ_T*), only uninterpreted symbols common to* ϕ *and* ψ*, and only free variables occurring freely in both* ϕ *and* ψ*.*

An alternative definition of interpolants used in the model-checking literature (called *reverse interpolants* in [18]) is presented below:

Definition 5 *Let* $\mathcal{T}$ *be a theory with signature* Σ*, and let* $\Sigma_T \subseteq \Sigma$ *be the set of interpreted symbols in* $\mathcal{T}$*. Let* ϕ *and* ψ *be formulae in the signature of* $\mathcal{T}$ *such that* $\phi \wedge \psi \models_{\mathcal{T}} \bot$*. We say that a formula* I *is a* reverse interpolant *of* ϕ *and* ψ *if (i)* $\phi \models_{\mathcal{T}} I$*, (ii)* $I \wedge \psi \models_{\mathcal{T}} \bot$*, and (iii)* I *contains interpreted symbols (i.e. any symbol in* Σ_T*), only uninterpreted symbols common to* ϕ *and* ψ*, and only free variables occurring freely in both* ϕ *and* ψ*.*

In this paper we study the existence of *reverse interpolants* (cf. Def. 5). If ψ is closed then $\phi \models_{\mathcal{T}} \psi$ iff $\phi \wedge \neg\psi \models_{\mathcal{T}} \bot$, so I is an interpolant of ϕ and ψ iff it is a reverse interpolant of ϕ and $\neg\psi$. Therefore, in order to be consistent with the terminology from the verification literature, we will use the name "interpolant" also for "reverse interpolant" (the meaning will be clear from the context).

By Thm. 3, first order logic has interpolation. It can be proved that in Thm. 3, if ϕ and ψ are ground then they have a ground interpolant. However, when considering theory-specific interpolants w.r.t. a theory $\mathcal{T}$, even if ϕ and ψ are very simple (e.g. quantifier-free or conjunctions of atoms), I may be a formula containing quantifiers. In many applications it is important to find *simple* interpolants: e.g., if ϕ and ψ are quantifier-free formulae, we are interested in the existence of *quantifier-free* interpolants. In the light of the remark in Section 2.3, we will sometimes consider an equivalent formulation of this problem: if ϕ and ψ are *ground* formulae, we are often interested in the existence of *ground* interpolants (below, this equivalent formulation is indicated between brackets).

Definition 6 *We say that a theory $\mathcal{T}$ (with set of interpreted symbols Σ_T) has* quantifier-free *(ground) interpolants if for all quantifier-free (ground) formulae* A *and* B*: If* $\mathsf{A} \wedge \mathsf{B} \models_{\mathcal{T}} \bot$ *there exists a quantifier-free (ground) formula I containing interpreted symbols (in Σ_T) and only uninterpreted symbols and variables (constants)*[3] *occurring in both* A *and* B *such that* $\mathsf{A} \models_{\mathcal{T}} I$ *and* $I \wedge \mathsf{B} \models_{\mathcal{T}} \bot$.

In this paper we present a method for generating quantifier-free (ground) interpolants for ground formulae w.r.t. the theory $LI(\mathbb{Q})$, and its extension $LI(\mathbb{Q})^{\Sigma}$ with free function symbols in a set Σ. In what follows, we will consider that the set Σ_T of interpreted symbols is the full signature of $LI(\mathbb{Q})$. The free function symbols in Σ as well as the existentially quantified variables in the formula (which after skolemization can be regarded as constants) will be considered to be uninterpreted. Analogous results can be obtained for $LI(\mathbb{R})$ and $LI(\mathbb{R})^{\Sigma}$.

3 Linear interpolants

In this section we present an algorithm $\textsc{Inter}_{LI(\mathbb{Q})}$ (Linear Interpolation) for the interpolant generation for linear arithmetic (with both strict and non-strict inequalities). We show the algorithm in Figure 1.

The input of $\textsc{Inter}_{LI(\mathbb{Q})}$ consists of two mixed systems of inequalities $Ax \leqslant a$ and $Bx \leqslant b$ that are mutually disjoint, i.e. the conjunction $Ax \leqslant a \wedge Bx \leqslant b$ is not satisfiable. The output of the algorithm is a linear interpolant $ix \lhd \delta$, where $\lhd \in \{\leq, <\}$.

The algorithm proceeds by constructing linear programming problems and solving these problems using an off-the-shelf linear programming solver. The solver is treated as a black box. The structure of the problems reflects the different cases why the conjunction $Ax \leqslant a \wedge Bx \leqslant b$ is unsatisfiable, following

[3] Thus, I contains only variables common to A and B (these are existentially quantified, so after skolemization they can be regarded as free constants).

algorithm $\text{INTER}_{LI(\mathbb{Q})}$
input
$Ax \leqslant a$ and $Bx \leqslant b$: systems of strict and non-strict inequalities,
where $Ax \leqslant a \wedge Bx \leqslant b$ is unsatisfiable
output
$ix \lhd \delta$: interpolant, where $\lhd \in \{\leq, <\}$
vars
Φ: auxiliary constraint
λ: vector defining linear combination of inequalities in $Ax \leqslant a$
$\lambda^{\mathsf{lt}}, \lambda^{\mathsf{le}}$: sub-vectors of λ defining linear combination of
strict and *non-strict* inequalities in $Ax \leqslant a$, respectively
(in particular, $\lambda A = \lambda^{\mathsf{lt}} A^{\mathsf{lt}} + \lambda^{\mathsf{le}} A^{\mathsf{le}}$ and $\lambda a = \lambda^{\mathsf{lt}} a^{\mathsf{lt}} + \lambda^{\mathsf{le}} a^{\mathsf{le}}$)
$\mu, \mu^{\mathsf{lt}}, \mu^{\mathsf{le}}$: analogous to $\lambda, \lambda^{\mathsf{lt}}$, and λ^{le}
begin
1 $\Phi := \ \lambda \geq 0 \wedge \mu \geq 0 \wedge i = \lambda A \wedge \delta = \lambda a \wedge \lambda A + \mu B = 0$
2 **if** exist λ, μ, i, δ satisfying $\Phi \wedge \lambda a + \mu b \leq -1$ **then**
3 $(*$ 1st branch $*)$
4 **return** $ix \leq \delta$
5 **else if** exist λ, μ, i, δ satisfying $\Phi \wedge \lambda a + \mu b \leq 0 \wedge \lambda^{\mathsf{lt}} \neq 0$ **then**
6 $(*$ 2nd branch $*)$
7 **return** $ix < \delta$
8 **else if** exist λ, μ, i, δ satisfying $\Phi \wedge \lambda a + \mu b \leq 0 \wedge \mu^{\mathsf{lt}} \neq 0$ **then**
9 $(*$ 3rd branch $*)$
10 **return** $ix \leq \delta$
end.

Fig. 1. Algorithm $\text{INTER}_{LI(\mathbb{Q})}$ for the synthesis of linear interpolants. We fix the following dimensions of matrices and vectors used in the algorithm: $A : m_A \times n$, $x : n \times 1$, $a : m_A \times 1$, $\lambda : 1 \times m_A$, $A^{\mathsf{lt}} : m_{A^{\mathsf{lt}}} \times n$, $A^{\mathsf{le}} : m_{A^{\mathsf{le}}} \times n$, $a^{\mathsf{lt}} : m_{A^{\mathsf{lt}}} \times 1$, $a^{\mathsf{le}} : m_{A^{\mathsf{le}}} \times 1$, $\lambda^{\mathsf{lt}} : 1 \times m_{A^{\mathsf{lt}}}$, $\lambda^{\mathsf{le}} : 1 \times m_{A^{\mathsf{le}}}$. The dimensions for $Bx \leqslant b$, μ, μ^{lt}, and μ^{le} are fixed in a similar fashion.

Motzkin's transposition theorem [30]. The proofs of Theorems 9 and 10 provide a formal explanation of the correspondence.

Theorem 7 ((Motzkin's) transposition theorem [30]) *Let A and B be matrices and let a and b be column vectors. Then there exists a vector x with $Ax < a$ and $Bx \leq b$, if and only if for all row vectors $y, z \geq 0$:*

- *if $yA + zB = 0$ then $ya + zb \geq 0$; and*
- *if $yA + zB = 0$ and $y \neq 0$ then $ya + zb > 0$.*

Example 8 *We simulate the algorithm $\text{INTER}_{LI(\mathbb{Q})}$ on the following unsatisfiable conjunction of mixed systems of inequalities*

$$z < 0 \wedge x \leq z \wedge y \leq x \quad \text{and} \quad y \leq 0 \wedge x + y \geq 0.$$

We assume an additional constraint that the resulting interpolant must not contain the variable y. We translate the inequalities into the matrix representation.

$$\underbrace{\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix}}_{A} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \leqslant \underbrace{\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}}_{a} \qquad \underbrace{\begin{pmatrix} 0 & 1 & 0 \\ -1 & -1 & 0 \end{pmatrix}}_{B} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \leqslant \underbrace{\begin{pmatrix} 0 \\ 0 \end{pmatrix}}_{b}$$

We split the mixed system $Ax \leqslant a$ into the strict part $A^{\mathsf{lt}}x < a^{\mathsf{lt}}$ and the non-strict part $A^{\mathsf{le}}x \leq a^{\mathsf{le}}$.

$$\underbrace{\begin{pmatrix} 0 & 0 & 1 \end{pmatrix}}_{A^{\mathsf{lt}}} \begin{pmatrix} x \\ y \\ z \end{pmatrix} < \underbrace{\begin{pmatrix} 0 \end{pmatrix}}_{a^{\mathsf{lt}}} \qquad \underbrace{\begin{pmatrix} 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix}}_{A^{\mathsf{le}}} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \leq \underbrace{\begin{pmatrix} 0 \end{pmatrix}}_{a^{\mathsf{le}}}$$

The system $Bx \leqslant b$ is equal to its non-strict part $B^{\mathsf{le}}x \leq b^{\mathsf{le}}$. The strict part of $Bx \leqslant b$ is empty.

Let $i = (i_x\ i_y\ i_z)$ and δ be the unknown coefficients that define the interpolant $i\left(\begin{smallmatrix} x \\ y \\ z \end{smallmatrix}\right) \lhd \delta$, where $\lhd$ is either the strict $<$ or non-strict $\leq$ inequality relation symbol. The algorithm computes the values for the unknown coefficients and determines the relation $\lhd$.

Let $\lambda = (\lambda_1\ \lambda_2\ \lambda_3)$ and $\mu = (\mu_1\ \mu_2)$ be the linear combinations of the inequalities of the first and the second system, respectively. We have $\lambda^{\mathsf{lt}} = (\lambda_1)$ and $\lambda^{\mathsf{le}} = (\lambda_2\ \lambda_3)$. The values of λ and μ determine the interpolant.

The guard of the first branch of $\textsc{Inter}_{LI(\mathbb{Q})}$ is unsatisfiable. The guard of the second branch is satisfiable; we compute the valuations $\lambda = (1\ 1\ 0)$ and $\mu = (1\ 1)$ together with the interpolant's coefficients $i = (1\ 0\ 0)$ and $\delta = 0$. Since $\lambda^{\mathsf{lt}} = (1)$, we have that $\lhd$ is the strict inequality relation symbol. The resulting interpolant is $x < 0$. □

For the completeness of the exposition, we show that two mutually unsatisfiable systems of mixed inequalities have an interpolant that is a single inequality. This inequality may be strict or non-strict.

Theorem 9 (Linear interpolants for mixed linear inequalities) *Given mutually unsatisfiable systems $Ax \leqslant a$ and $Bx \leqslant b$ of strict and non-strict inequalities, there exists a linear inequality interpolant $ix \lhd \delta$, where $\lhd \in \{\leq, <\}$.*

Proof. Let $Ax \leqslant a$ consist of strict inequalities $A^{\mathsf{lt}}x < a^{\mathsf{lt}}$ and non-strict inequalities $A^{\mathsf{le}}x \leq a^{\mathsf{le}}$. Let $B^{\mathsf{lt}}x < b^{\mathsf{lt}}$ and $B^{\mathsf{le}}x \leq b^{\mathsf{le}}$ be the corresponding partitioning of $Bx \leqslant b$. Since $Ax \leqslant a \wedge Bx \leqslant b$ is unsatisfiable, there exist non-negative row vectors $\lambda = (\lambda^{\mathsf{lt}}\ \lambda^{\mathsf{le}})$ and $\mu = (\mu^{\mathsf{lt}}\ \mu^{\mathsf{le}})$ such that $\lambda A + \mu B = 0$ and (i) $\lambda a + \mu b < 0$ or $\lambda a + \mu b \leq 0$ if either (ii) $\lambda^{\mathsf{lt}} \neq 0$ or (iii) $\mu^{\mathsf{lt}} \neq 0$. We define $i = \lambda A$ and $\delta = \lambda a$. We prove that $ix \lhd \delta$ is an interpolant. We choose the relation symbol $\lhd$ as follows: for cases (i) and (iii) it is the non-strict inequality $\leq$, and for case (ii) it is the strict inequality $<$.

We show that $ix \lhd \delta$ is expressed over the common variables of $Ax \leqslant a$ and $Bx \leqslant b$. Assume that a variable x_k in x is not common to both systems. This means that $A_{|k} = 0$ or $B_{|k} = 0$ holds. From $\lambda A + \mu B = 0$ follows that $i_k = 0$, since $i_k = \lambda A_{|k} = -\mu B_{|k} = 0$.

The proof of the implication between $Ax \leqslant a$ and $ix \leq \delta$ for cases (i) and (iii) is straightforward. We show that $Ax \leqslant a$ implies the strict inequality $ix < \delta$ if case (ii) applies. Let $(\lambda\ 1)$ be the row vector obtained from λ by extending it with the constant 1. We have $(\lambda\ 1)\left(\begin{smallmatrix} A \\ -i \end{smallmatrix}\right) = 0$ and $(\lambda\ 1)\left(\begin{smallmatrix} a \\ -\delta \end{smallmatrix}\right) = 0$. In case (ii) we have $\lambda^{\mathsf{lt}} \neq 0$. By Motzkin's transposition theorem we have $Ax \leqslant a \wedge -ix \leq -\delta$ is unsatisfiable. We conclude that $Ax \leqslant a$ implies $ix < \delta$ in case (ii).

We prove that $Bx \leqslant b$ and $ix \lhd \delta$ are mutually unsatisfiable in a way similar to the reasoning above. We have $(\mu\ 1)\left(\begin{smallmatrix} B \\ i \end{smallmatrix}\right) = 0$ and $(\mu\ 1)\left(\begin{smallmatrix} b \\ \delta \end{smallmatrix}\right) < 0$ or $(\mu\ 1)\left(\begin{smallmatrix} b \\ \delta \end{smallmatrix}\right) \leq 0$ if $\lambda^{\mathsf{lt}} \neq 0$ or $\mu^{\mathsf{lt}} \neq 0$. If $(\mu\ 1)\left(\begin{smallmatrix} b \\ \delta \end{smallmatrix}\right) < 0$, i.e. in case (i), by Motzkin's transposition theorem we have that $Bx \leqslant b$ and $ix \leq \delta$ are mutually unsatisfiable. The reasoning for the cases (ii) and (iii) is analogous.

The correctness of the algorithm $\textsc{Inter}_{LI(\mathbb{Q})}$ is stated in the following theorem.

Theorem 10 (Algorithm $\textsc{Inter}_{LI(\mathbb{Q})}$: Soundness and completeness) *The algorithm $\textsc{Inter}_{LI(\mathbb{Q})}$ is sound and complete: It always produces a linear interpolant, by taking the 1st, 2nd or 3rd branch.*

Proof. **Soundness** Let $Ax \leqslant a$ and $Bx \leqslant b$ be mutually unsatisfiable systems of inequalities. Let $ix \lhd \delta$, where $\lhd \in \{\leq, <\}$, be an inequality computed by (taking one of the branches of) the algorithm $\textsc{Inter}_{LI(\mathbb{Q})}$. We proof that the inequality $ix \lhd \delta$ is an interpolant.

We show that $ix \lhd \delta$ is expressed over the common variables of $Ax \leqslant a$ and $Bx \leqslant b$. For contradiction, assume that a variable x_k in x, where $k \in \{1, \ldots, n\}$, appears in the interpolant and is local wrt. one of the systems. This means that $i_k \neq 0$, and either $A_{|k} = 0$ or $B_{|k} = 0$ holds.[4] The constraint Φ in the algorithm $\textsc{Inter}_{LI(\mathbb{Q})}$ enforces that $\lambda A + \mu B = 0$. We have $i_k = 0$, since $i_k = \lambda A_{|k} = -\mu B_{|k} = 0$, which is a contradiction.

We show the implication between $Ax \leqslant a$ and $ix \lhd \delta$. The proof is a straightforward application of Farkas' lemma [30] if the algorithm produces the result by taking either the first or the third branch, i.e. $\lhd$ is a non-strict inequality, since $ix \lhd \delta$ is a non-negative linear combination of the inequalities in $Ax \leqslant a$. Now we consider the case when the second branch is taken, i.e. $\lhd$ is a strict inequality. Let $(\lambda\ 1)$ be the row vector obtained from λ by extending it with the constant 1. We have $(\lambda\ 1)\left(\begin{smallmatrix} A \\ -i \end{smallmatrix}\right) = 0$ and $(\lambda\ 1)\left(\begin{smallmatrix} a \\ -\delta \end{smallmatrix}\right) = 0$, which is enforced by the constraint Φ. We rewrite these equalities as

$$\lambda^{\mathsf{lt}} A^{\mathsf{lt}} + (\lambda^{\mathsf{le}}\ 1)\left(\begin{smallmatrix} A^{\mathsf{le}} \\ -i \end{smallmatrix}\right) = 0 \quad \text{and} \quad \lambda^{\mathsf{lt}} a^{\mathsf{lt}} + (\lambda^{\mathsf{le}}\ 1)\left(\begin{smallmatrix} a^{\mathsf{le}} \\ -\delta \end{smallmatrix}\right) = 0.$$

[4] Recall that $A_{|k}$ denotes the k-th column of a matrix A.

Furthermore, we have $\lambda^{\mathsf{lt}} \neq 0$ when the second branch is taken. By Motzkin's transposition theorem we have that

$$A^{\mathsf{lt}} x < a^{\mathsf{lt}} \wedge \begin{pmatrix} A^{\mathsf{le}} \\ -i \end{pmatrix} x \leq \begin{pmatrix} a^{\mathsf{le}} \\ -\delta \end{pmatrix}$$

is unsatisfiable, which we rewrite as $Ax \leqslant a \wedge -ix \leq -\delta$. We conclude that $Ax \leqslant a$ implies $ix < \delta$.

We show that $Bx \leqslant b$ and $ix \lhd \delta$ are mutually unsatisfiable. We have that $(\mu\ 1) \binom{B}{i} = 0$ is enforced by Φ. If the first branch is taken then $(\mu\ 1) \binom{b}{\delta} < 0$. Hence by Motzkin's transposition theorem, by failing its first condition, we have that $Bx \leqslant b$ and $ix \leq \delta$ are mutually unsatisfiable. If the second branch is taken then we have $(\mu\ 1) \binom{b}{\delta} \leq 0$, which we rewrite as

$$(\mu^{\mathsf{lt}}\ 1) \begin{pmatrix} B^{\mathsf{lt}} \\ i \end{pmatrix} + \mu^{\mathsf{le}} B^{\mathsf{le}} = 0 \quad \text{and} \quad (\mu^{\mathsf{lt}}\ 1) \begin{pmatrix} b^{\mathsf{lt}} \\ \delta \end{pmatrix} + \mu^{\mathsf{le}} b^{\mathsf{le}} \leq 0.$$

Since $(\mu^{\mathsf{lt}}\ 1) \neq 0$, the Motzkin's transposition theorem applies and proves $Bx \leqslant b \wedge ix < \delta$ unsatisfiable. The reasoning about the third branch of the algorithm is similar.

Completeness Assume that there exists a linear interpolant $ix \lhd \delta$, where $\lhd \in \{\leq, <\}$, for the mutually unsatisfiable systems $Ax \leqslant a$ and $Bx \leqslant b$. We prove that one of the three branches of the algorithm succeeds. In the proof we make a distinction whether the interpolant is a strict or a non-strict inequality.

Non-strict: We prove that suitable values λ and μ, which satisfy the guard of either the first or the third branch, do exist.

By Motzkin's transposition theorem, from the mutual unsatisfiability of $Ax \leqslant a$ and $-ix < -\delta$ follows that there exist row vectors $\lambda^{\mathsf{lt}}, \lambda^{\mathsf{le}} \geq 0$ and a constant $\alpha \geq 0$ such that $(\lambda^{\mathsf{lt}}\ \alpha) \begin{pmatrix} A^{\mathsf{lt}} \\ -i \end{pmatrix} + \lambda^{\mathsf{le}} A^{\mathsf{le}} = 0$ and $(\lambda^{\mathsf{lt}}\ \alpha) \begin{pmatrix} a^{\mathsf{lt}} \\ -\delta \end{pmatrix} + \lambda^{\mathsf{le}} a^{\mathsf{le}} \leq 0$. Furthermore, we have $\alpha > 0$, since the system $Ax \leqslant a$ is satisfiable. We define λ to be a combination of inequalities from $Ax \leqslant a$ such that the coefficient from strict inequalities are λ^{lt}, and the coefficients for non-strict inequalities are λ^{le}. We have $\lambda A = \lambda^{\mathsf{lt}} A^{\mathsf{lt}} + \lambda^{\mathsf{le}} A^{\mathsf{le}} = \alpha i$ and $\lambda a = \lambda^{\mathsf{lt}} a^{\mathsf{lt}} + \lambda^{\mathsf{le}} a^{\mathsf{le}} \leq \alpha\delta$.

Analogously, from the unsatisfiability of $Bx \leqslant b \wedge ix \leq \delta$ we obtain row vectors $\mu^{\mathsf{lt}}, \mu^{\mathsf{le}} \geq 0$ and a constant $\beta \geq 0$ such that $\mu^{\mathsf{lt}} B^{\mathsf{lt}} + (\mu^{\mathsf{le}}\ \beta) \begin{pmatrix} B^{\mathsf{le}} \\ i \end{pmatrix} = 0$ and $\mu^{\mathsf{lt}} b^{\mathsf{lt}} + (\mu^{\mathsf{le}}\ \beta) \begin{pmatrix} b^{\mathsf{le}} \\ \delta \end{pmatrix} < 0$ or $\mu^{\mathsf{lt}} b^{\mathsf{lt}} + (\mu^{\mathsf{le}}\ \beta) \begin{pmatrix} b^{\mathsf{le}} \\ \delta \end{pmatrix} \leq 0$ if $\mu^{\mathsf{lt}} \neq 0$. Here, we have $\beta > 0$. We construct μ from $\mu^{\mathsf{lt}} = \frac{\alpha}{\beta}\mu^{\mathsf{lt}}$ and $\frac{\alpha}{\beta}\mu^{\mathsf{le}}$ in a way similar to the construction of λ. We have $\mu B = \frac{\alpha}{\beta}(\mu^{\mathsf{lt}} B^{\mathsf{lt}} + \mu^{\mathsf{le}} B^{\mathsf{le}}) = -\alpha i$ and $\mu b = \frac{\alpha}{\beta}(\mu^{\mathsf{lt}} b^{\mathsf{lt}} + \mu^{\mathsf{le}} b^{\mathsf{le}}) < -\alpha\delta$ or $\mu b \leq -\alpha\delta$ if $\mu^{\mathsf{lt}} \neq 0$.

We have $\lambda A + \mu B = 0$ and i) $\lambda a + \mu b < 0$ or ii) $\lambda a + \mu b \leq 0$ if $\mu^{\mathsf{lt}} \neq 0$. Let us consider case (i). Let $\gamma = \lambda a + \mu b$. We scale, i.e. redefine, λ and μ to be $-\frac{1}{\gamma}\lambda$ and $-\frac{1}{\gamma}\mu$, respectively. Then, we have $\lambda a + \mu b \leq -1$ for the scaled λ and μ. The first branch of the algorithm computes the interpolant. In case (ii) the third branch of the algorithm computes the interpolant.

Strict: Let $ix < \delta$ be the existing interpolant. The proof that there exist suitable values λ and μ that satisfy the guard of the second branch follows the lines of the reasoning above.

3.1 Interpolants for disjunctions

We obtain an algorithm for the synthesis of constrained interpolants for disjunctions of mixed systems

$$\bigvee_k A_k x \leqslant a_k \quad \text{and} \quad \bigvee_l B_l x \leqslant b_l$$

by taking the disjunction of convex hulls

$$\bigvee_k \bigwedge_l i_{kl} x \leqslant \delta_{kl}$$

that consists of interpolants $i_{kl}x \leqslant \delta_{kl}$ for each pair of disjuncts $A_k x \leqslant a_k$ and $B_l x \leqslant b_l$. The constraints above are in *disjunctive normal form*: both formulae for which the interpolant is computed are disjunctions of conjunctions. In applications we sometimes need to compute constrained interpolants for conjunctions of non-unit clauses, i.e. for formulae in *conjunctive normal form*. For this we can use standard methods discussed e.g. in [23] or [35]: in a DPLL-style procedure partial interpolants are generated for the unsatisfiable branches and then recombined using ideas of Pudlák [28].

An alternative approach for dealing disjunctions involves the construction of a *conjunctive* unsatisfiable core that is responsible for the mutual unsatisfiability of the input formulae [1]. Then, $\textsc{Inter}_{LI(\mathbb{Q})}$ is applied on the formulae that appear in the core.

4 Extension with uninterpreted function symbols

So far, we have presented an algorithm for the generation of interpolants in the theory of linear arithmetic. Our application domains mentioned in the introduction include software model checking and verification of timed and hybrid systems. They naturally motivate an extension of the interpolant-generation algorithm to combination of linear arithmetic with additional theories.

In this section we consider the extension with free functions, which is useful for conservative approximation of non-arithmetic expressions. The algorithm we propose is based on a hierarchical calculus for reasoning in certain extensions of theories (which we called *local extensions*) [31]. This calculus makes it possible to reduce checking satisfiability of quantifier-free formulae w.r.t. the extension, to checking satisfiability of formulae in the 'base theory'. Any extension of a theory with uninterpreted function symbols falls into this class. As the notion of local theory extension is not needed in the present context, all relevant results will be presented for the special case of extensions of linear rational and real arithmetic with free function symbols. For the sake of simplicity, we will use as running example $LI(\mathbb{Q})$. All results can be used as well for $LI(\mathbb{R})$. However, many of the results presented here also hold for more general extensions. Such generalizations were presented in [32,33].

We begin by giving the main idea of the hierarchical calculus for extensions with free function symbols in [31] (Section 4.1). Based on this, in Section 4.2 we present a hierarchical method for generating interpolants in such extensions.

Notation: Everywhere in what follows let Σ be a set of (new) function symbols. We denote by $LI(\mathbb{Q})^{\Sigma}$ the extension of $\mathbb{Q}$ with the uninterpreted function symbols in Σ. We refer to the function symbols in Σ as extension functions. To distinguish them from constraints in linear arithmetic, we denote conjunctions of (unit) literals over this extended signature using a special font ($\mathsf{A} \wedge \mathsf{B}$).

4.1 A hierarchical calculus

Let Σ be a set of uninterpreted function symbols, let $LI(\mathbb{Q})^{\Sigma}$ be the extension of linear rational arithmetic $LI(\mathbb{Q})$ with the uninterpreted function symbols in Σ. Given a disjunction $\phi(x_1, \ldots, x_n)$ of conjunctions of atomic formulae over the signature of $LI(\mathbb{Q})^{\Sigma}$, we want to check whether

$$LI(\mathbb{Q})^{\Sigma} \models \forall x_1 \ldots \forall x_n \phi(x_1, \ldots, x_n),$$

i.e., whether ϕ holds in each model of $LI(\mathbb{Q})^{\Sigma}$ and for all possible assignments of values in this model to the variables $x_1, \ldots, x_n$. Equivalently, we can test whether there exists a model and a possible assignment to the variables $x_1, \ldots, x_n$ in it for which $\neg\phi$ becomes true, i.e. checking whether $\neg\phi(x_1, \ldots, x_n)$ is satisfiable. Thus, proving truth w.r.t. all models and valuations can be reduced to proving satisfiability of sets of clauses w.r.t. $LI(\mathbb{Q})^{\Sigma}$.

Let $G(c_1, \ldots, c_n)$ be a set of quantifier-free clauses with variables $c_1, \ldots, c_n$[5] in the signature of $LI(\mathbb{Q})^{\Sigma}$. To check the satisfiability of $G(c_1, \ldots, c_n)$ w.r.t. $LI(\mathbb{Q})^{\Sigma}$ we can proceed as follows:

Step 1: Flattening and purification. G is purified and flattened by introducing fresh variables for the arguments of the extension functions as well as for the subterms $t = f(g_1, \ldots, g_n)$ starting with extension functions $f \in \Sigma$, together with corresponding definitions $c_t = t$. We obtain a set of clauses $G_0 \wedge D$, where D consists of unit clauses of the form $f(c_1, \ldots, c_n) = c$, where $c_1, \ldots, c_n, c$ are variables and $f \in \Sigma$, and G_0 contains clauses without function symbols in Σ.

Step 2: Reduction to testing satisfiability in $LI(\mathbb{Q})$. By the locality of any extension with free function symbols [31], we know that we can reduce the problem of testing satisfiability of G w.r.t. $LI(\mathbb{Q})^{\Sigma}$ to a satisfiability test in $LI(\mathbb{Q})$ as shown in Theorem 11.

[5] In what follows we are concerned with satisfiability of such clauses; the variables in $G(c_1, \ldots, c_n)$ are implicitly existentially quantified. In the automated reasoning literature, existential variables are replaced by constants, using skolemization; thus one can replace the variables in $G(c_1, \ldots, c_n)$ by (Skolem) constants. In what follows we refer to them as variables. However, the notation we chose reminds that these existentially quantified variables can, in fact, be regarded as "constants".

Theorem 11 ([31]) *With the notations above, the following are equivalent:*
(1) $G \models_{LI(\mathbb{Q})^\Sigma} \bot$,
(2) $G_0 \wedge D \models_{LI(\mathbb{Q})^\Sigma} \bot$,
(3) $G_0 \wedge N_0 \models_{LI(\mathbb{Q})} \bot$, *where*

$$N_0 = \bigwedge\{\bigwedge_{i=1}^{n} c_i = d_i \rightarrow c = d \mid f(c_1, \ldots, c_n) = c \in D, f(d_1, \ldots, d_n) = d \in D\}.$$

is the set of functionality axioms corresponding to the terms occurring in D.

Problem (3) in Theorem 11 is a satisfiability problem for quantifier-free clauses in linear rational arithmetic. We thus reduced, hierarchically, the problem of testing the satisfiability of the set of quantifier-free clauses G in $LI(\mathbb{Q})^\Sigma$ to the problem of testing the satisfiability of a set of quantifier-free constraints in $LI(\mathbb{Q})$.

Complexity. Flattening and purification can be done in linear time; the growth of the formulae is linear. The size of the satisfiability problem in $LI(\mathbb{Q})$ obtained by the translation above is quadratic in the number of extension terms in the input formula. Hence, the complexity of the procedure is of order $k(n^2)$, where n is the size of the input formula and $k(m)$ is the complexity of the problem of testing the satisfiability of sets of ground clauses in $LI(\mathbb{Q})$ for an input of size m.

Remark. If G is a set of unit clauses then the procedure mimics the Nelson-Oppen procedure for combination of $LI(\mathbb{Q})$ with the theory of free function symbols in Σ within the prover for linear arithmetic. (Due to the convexity of linear arithmetic, we can always find a clause in N_0 whose premises are implied by G. The clause is replaced with its conclusion and the procedure is repeated until a set of unit clauses is obtained.) Thus, exchange of equalities between shared variables needs not be done explicitly. The complexity of the method is similar to that of the Nelson-Oppen combination of convex theories.

The following example illustrates the method.

Example 12 *Let $G = \mathsf{A} \wedge \mathsf{B}$, where*

$$\begin{aligned} \mathsf{A}: &\quad g(a) = c + 5 \wedge f(g(a)) \geq c + 1, \\ \mathsf{B}: &\quad h(b) = d + 4 \wedge d = c + 1 \wedge f(h(b)) < c + 1. \end{aligned}$$

We show that $\mathsf{A} \wedge \mathsf{B}$ is unsatisfiable in $LI(\mathbb{Q})^{\{f,g,h\}}$ as follows:

Step 1: Flattening and purification. *We purify and flatten the formulae A and B by replacing the terms starting with f with new variables. We obtain the following purified form:*

$$\begin{aligned} \mathsf{A_0}: &\quad a_1 = c + 5 \wedge a_2 \geq c + 1, & D_A: &\quad a_1 = g(a) \wedge a_2 = f(a_1), \\ \mathsf{B_0}: &\quad b_1 = d + 4 \wedge d = c + 1 \wedge b_2 < c + 1, & D_B: &\quad b_1 = h(b) \wedge b_2 = f(b_1). \end{aligned}$$

Step 2: Hierarchical reasoning. *By Theorem 11 we have that $\mathsf{A} \wedge \mathsf{B}$ is unsatisfiable in $LI(\mathbb{Q})^{\{f,g,h\}}$ iff $\mathsf{A_0} \wedge \mathsf{B_0} \wedge N_0$ is unsatisfiable in $LI(\mathbb{Q})$, where N_0 corresponds*

to the consequences of the congruence axioms for those ground terms which occur in the definitions $D_A \wedge D_B$ for the newly introduced variables.

Def	G_0	N_0
$D_A : a_1{=}g(a) \wedge a_2{=}f(a_1)$	$\mathsf{A}_0 : a_1 = c+5 \wedge a_2 \geq c+1$	$N_0 :\ b_1{=}a_1 \rightarrow b_2{=}a_2$
$D_B : b_1{=}h(b) \wedge b_2{=}f(b_1)$	$\mathsf{B}_0 : b_1 = d+4 \wedge d = c+1 \wedge b_2 < c+1$	

To prove that $\mathsf{A}_0 \wedge \mathsf{B}_0 \wedge N_0$ is unsatisfiable, note that $\mathsf{A}_0 \wedge \mathsf{B}_0 \models a_1 = b_1$. Hence, $\mathsf{A}_0 \wedge \mathsf{B}_0 \wedge N_0$ entails $a_2 = b_2 \wedge a_2 \geq c+1 \wedge b_2 < c+1$, which is inconsistent.

4.2 Hierarchical Interpolation in $LI(\mathbb{Q})^{\Sigma}$

We show how this hierarchical calculus can be used to generate interpolants for extensions with free function symbols.

Assume that $\mathsf{A} \wedge \mathsf{B} \models_{LI(\mathbb{Q})^{\Sigma}} \bot$, where A and B are two sets of ground clauses. Our goal is to find an *interpolant*, that is a quantifier-free formula I, containing only variables and uninterpreted function symbols which are common to A and B such that

$$\mathsf{A} \models_{LI(\mathbb{Q})^{\Sigma}} I \quad \text{and} \quad I \wedge \mathsf{B} \models_{LI(\mathbb{Q})^{\Sigma}} \bot\ .$$

For the sake of simplicity we first restrict to sets A and B of unit clauses, i.e. to conjunctions of ground literals. Our goal is to reduce the search for the interpolant of $\mathsf{A} \wedge \mathsf{B}$ in $LI(\mathbb{Q})^{\Sigma}$ to the task of computing one or more interpolants in $LI(\mathbb{Q})$. We with present two methods of doing so, namely:

(1) An approach described in [32,33] in which the computation of the interpolant is reduced to the following tasks:
  (i) constructing an interpolant I_0 in $LI(\mathbb{Q})$,
  (ii) using I_0 to construct an interpolant for $\mathsf{A} \wedge \mathsf{B}$ (by appropriate substitutions).

(2) An approach in which the computation of the interpolant is interleaved with a separation process for the atoms occurring in the problem into a pure A abd a pure B part.

As explained before, $\mathsf{A} \wedge \mathsf{B}$ is purified and flattened by introducing fresh variables for the arguments of the extension functions as well as for the subterms $t = f(g_1, \ldots, g_n)$ starting with extension functions $f \in \Sigma$, together with corresponding definitions $c_t = t$. We obtain a set of clauses $\mathsf{A}_0 \wedge D_A \wedge \mathsf{B}_0 \wedge D_B$, where D_A (resp. D_B) consists of unit clauses of the form $f(c_1, \ldots, c_n) = c$, where $c_1, \ldots, c_n, c$ are constants (extistentially quantified variables) and $f \in \Sigma$ obtained from flattening and purifying the terms in A (resp. B), and $\mathsf{A}_0 \wedge \mathsf{B}_0$ contains clauses without function symbols in Σ. Flattening and purification do not influence the existence of interpolants.

Theorem 13 ([32,33]) *If I_0 is an interpolant of the flattened forms $(\mathsf{A}_0 \wedge D_A) \wedge (\mathsf{B}_0 \wedge D_B)$ of A_0 and B_0, then the formula $\overline{I}_0$, obtained from I_0 by replacing, recursively, all newly introduced variables with the terms in the original signature which they represent, is an interpolant for $\mathsf{A} \wedge \mathsf{B}$.*

Therefore we can restrict w.l.o.g. to finding interpolants for the *purified and flattened* set of formulae $(\mathsf{A}_0 \wedge D_A) \wedge (\mathsf{B}_0 \wedge D_B)$.

By Theorem 11, the following holds:

$$\mathsf{A}_0 \wedge D_A \wedge \mathsf{B}_0 \wedge D_B \models_{LI(\mathbb{Q})^\Sigma} \bot \quad \text{if and only if} \quad \mathsf{A}_0 \wedge \mathsf{B}_0 \wedge N_0 \models_{LI(\mathbb{Q})} \bot,$$

where $N_0 = \bigwedge\{\bigwedge_{i=1}^n c_i{=}d_i \rightarrow c{=}d \mid f(c_1, \ldots, c_n) = c, f(d_1, \ldots, d_n) = d \in D\}$.

By definition, $N_0 = N^A \wedge N^B \wedge N_{\mathsf{mix}}$, where:

- N^A only contains variables from A_0 (it is A-pure),
- N^B only contains variables from B_0 (it is B-pure), and
- $N_{\mathsf{mix}} = \bigwedge\{\bigwedge_{i=1}^n a_i = b_i \rightarrow a = b \mid f(a_1, \ldots, a_n) = a \in (D_A \backslash D_B), f(b_1, \ldots, b_n) = b \in (D_B \backslash D_A)\}$.

The clauses in N_{mix} are mixed, i.e. contain combinations of A-local and B-local variables. Thus, the equivalence in Theorem 11 cannot be used directly for generating a ground interpolant.

Example 14 *Consider the reduction to the base theory in the previous example. The clause $a_1 = b_1 \rightarrow a_2 = b_2$ of N_0 contains both A-local and B-local variables.*

Idea The idea of our approach is to separate mixed instances N_{mix} of congruence axioms in N_0, into an A-part and a B-part. We show that if $\mathsf{A} \wedge \mathsf{B} \models_{LI(\mathbb{Q})^\Sigma} \bot$ then we find a set T of terms in the signature of $LI(\mathbb{Q})^\Sigma$ containing only variables and extension functions common to A and B, which allows us to separate the instances of functionality axioms in N_{mix} into a part N^A_{sep} consisting of instances of functionality axioms for extension terms occurring in A and T, and a part N^B_{sep} consisting of instances with terms occurring in B and T. We show that such a separation does not lead to the loss of unsatisfiability, i.e. that the conjunction

$$(\mathsf{A}_0 \wedge N_A \wedge N^A_{\mathsf{sep}}) \wedge (\mathsf{B}_0 \wedge N_B \wedge N^B_{\mathsf{sep}})$$

has no model where the extension functions may be partial, but in which all terms in D_A, D_B, and T are defined.

Example 15 *Consider the reduction to the base theory in the example given in Section 4.1. The clause $a_1 = b_1 \rightarrow a_2 = b_2$ of N_{mix} can be replaced with a conjunction of A-pure and B-pure clauses as follows:*

Note that $\mathsf{A}_0 \wedge \mathsf{B}_0 \models a_1 = b_1$. It is easy to see that there exists a term t (namely $t = c + 5$) containing only variables common to A_0 and B_0 such that $\mathsf{A}_0 \models_{LI(\mathbb{Q})} a_1 = t$ and $\mathsf{B}_0 \models_{LI(\mathbb{Q})} t = b_1$. Let $T = \{t\} = \{c + 5\}$. We show that instead of using the mixed clause $a_1 = b_1 \rightarrow a_2 = b_2$, we can use, without loss of unsatisfiability, the flattened and purified instances N^A_{sep} and N^B_{sep} of the functionality axioms corresponding to terms in A and T, resp. B and T:

$$N^A_{\mathsf{sep}} = \{a_1 = c + 5 \rightarrow a_2 = c_{f(c+5)}\}, \quad N^B_{\mathsf{sep}} = \{c + 5 = b_1 \rightarrow c_{f(c+5)} = b_2\}.$$

(We introduced a new constant $c_{f(c+5)}$ for $f(c+5)$, together with its definition $D_T : \ c_{f(c+5)} = f(c+5)$.) We can thus replace N_0 with the instances of the congruence axioms N^A_{sep} and N^B_{sep}, now separated into an A-part and a B-part. It is now sufficient to compute an interpolant in $LI(\mathbb{Q})$ for

$$(\mathsf{A}_0 \wedge N^A_{\mathsf{sep}}) \wedge (\mathsf{B}_0 \wedge N^B_{\mathsf{sep}}).$$

To compute the interpolant, note that $\mathsf{A}_0 \wedge N^A_{\mathsf{sep}}$ is logically equivalent to $\mathsf{A}_0 \wedge a_2{=}c_{f(c+5)}$, and $\mathsf{B}_0 \wedge N^B_{\mathsf{sep}}$ is logically equivalent to $\mathsf{B}_0 \wedge b_2{=}c_{f(c+5)}$. The conjunction $(\mathsf{A}_0 \wedge a_2{=}c_{f(c+5)}) \wedge (\mathsf{B}_0 \wedge b_2{=}c_{f(c+5)})$ is unsatisfiable. An interpolant is $I_0 : c_{f(c+5)} \geq c+1$. Thus, $\mathsf{A}_0 \wedge a_2{=}c_{f(c+5)} \models I_0$ and $\mathsf{B}_0 \wedge b_2{=}c_{f(c+5)} \wedge I_0 \models \bot$.

Let $I = (f(c+5) \geq c+1)$ be obtained by replacing the newly introduced constant $c_{f(c+5)}$ with the term it denotes (namely $f(c+5)$). It is easy to see that:

$$\mathsf{A}_0 \wedge D_A \models_{LI(\mathbb{Q})^{\{f,g,h\}}} \mathsf{A}_0 \wedge (a_2{=}f(c+5)) \models_{LI(\mathbb{Q})^{\{f,g,h\}}} I,$$
$$\mathsf{B}_0 \wedge D_B \models_{LI(\mathbb{Q})^{\{f,g,h\}}} \mathsf{B}_0 \wedge (b_2{=}f(c+5)) \models_{LI(\mathbb{Q})^{\{f,g,h\}}} \neg I.$$

Thus, I is an interpolant for $(\mathsf{A}_0 \wedge D_A) \wedge (\mathsf{B}_0 \wedge D_B)$, hence also for $\mathsf{A} \wedge \mathsf{B}$.

The method With the notation used before, assume that

$$\mathsf{A}_0 \wedge D_A \wedge \mathsf{B}_0 \wedge D_B \models_{LI(\mathbb{Q})^{\Sigma}} \bot \ .$$

Then $\mathsf{A}_0 \wedge \mathsf{B}_0 \wedge N_0 \models_{LI(\mathbb{Q})} \bot$, where $N_0 = N^A \wedge N^B \wedge N_{\mathsf{mix}}$, the clauses in N^A are A-pure, those in N^B are B-pure, and those in $N_{\mathsf{mix}} = \bigwedge\{\bigwedge_{i=1}^n a_i = b_i \rightarrow a = b \mid f(a_1, \ldots, a_n) = a \in (D_A \backslash D_B), f(b_1, \ldots, b_n) = b \in (D_B \backslash D_A)\}$ contain combinations of A-local and B-local variables.

Our goal is to replace N_{mix} with the conjunction of an A-pure and a B-pure part, $N^A_{\mathsf{sep}} \wedge N^B_{\mathsf{sep}}$, of instances of the functionality axioms. We show that this is possible at the price of having to take into account additional terms over the shared signature of A and B not occurring in $\mathsf{A} \wedge \mathsf{B}$.

Theorem 16 ([32,33]) *Let A_0 and B_0 be conjunctions of literals in the signature of $LI(\mathbb{Q})$ such that $\mathsf{A}_0 \wedge \mathsf{B}_0 \wedge N \models_{\mathcal{T}_0} \bot$, for a set $N = N^A \cup N^B \cup N_{\mathsf{mix}}$ of flattened instances of congruence axioms. There exists a set T of $\Sigma_{LI(\mathbb{Q})}$-terms containing only variables common to A_0 and B_0, and possibly common newly introduced variables in a set Σ_c such that*

$$\mathsf{A}_0 \wedge \mathsf{B}_0 \wedge (N^A \wedge N^B) \wedge N_{\mathsf{sep}} \models_{\mathcal{T}_0} \bot,$$

where $N_{\mathsf{sep}} = \ \bigwedge\{(\bigwedge_{i=1}^n c_i{=}t_i \rightarrow c{=}c_{f(t_1,\ldots,t_n)}) \wedge (\bigwedge_{i=1}^n t_i{=}d_i \rightarrow c_{f(t_1,\ldots,t_n)}{=}d) \mid \bigwedge_{i=1}^n c_i = d_i \rightarrow c = d \in N_{\mathsf{mix}}\} \ = \ N^A_{\mathsf{sep}} \wedge N^B_{\mathsf{sep}}$

and $c_{f(t_1,\ldots,t_n)}$ are new variables in Σ_c (considered to be common) introduced for the terms $f(t_1, \ldots, t_n)$.

Idea of the proof. The proof is by induction on the number of clauses in N. If $N = \emptyset$ the result is trivially true. Otherwise we select a clause $C = \bigwedge_{i=1}^{n} c_i = d_i \rightarrow c = d$ of N with the property that $\mathsf{A}_0 \wedge \mathsf{B}_0$ entails (w.r.t. $LI(\mathbb{Q})$) all the premises $c_i = d_i$. If the clause C is pure, we can simply replace N with $(N \backslash \{C\}) \wedge (c = d)$ and repeat the procedure.

If the clause C is mixed, we compute terms t_i containing only variables common to A and B such that $\mathsf{A}_0 \wedge \mathsf{B}_0 \models_{LI(\mathbb{Q})} c_i = t_i \wedge t_i = d_i$. Then we can replace the clause C with the conjunction of the clauses C_A and C_B, where:

$$C_A: \quad \bigwedge_{i=1}^{n} c_i{=}t_i \rightarrow c{=}c_{f(t_1,\ldots,t_n)} \qquad \text{and} \qquad C_B: \quad \bigwedge_{i=1}^{n} t_i{=}d_i \rightarrow c_{f(t_1,\ldots,t_n)}{=}d,$$

where $c_{f(t_1,\ldots,t_n)}$ is a new constant, denoting the term $f(t_1, \ldots, t_n)$. We now repeat the procedure for $(N \backslash C) \wedge C_A \wedge C_B$ (taking into account that $\mathsf{A}_0 \wedge \mathsf{B}_0 \wedge N$ is logically equivalent to $(N \backslash C) \wedge c = c_{f(t_1,\ldots,t_n)} \wedge c_{f(t_1,\ldots,t_n)} = d$). □

A direct consequence of Theorem 16 is the possibility of hierarchically generating interpolants in $LI(\mathbb{Q})^{\Sigma}$.

Corollary 17 ([32,33]) *Assume that* $(\mathsf{A}_0 \wedge D_A) \wedge (\mathsf{B}_0 \wedge D_B) \models_{LI(\mathbb{Q})^{\Sigma}} \bot$, *and let* $N_0, N^A, N^B, N_{\mathsf{mix}}, N^A_{\mathsf{sep}}, N^B_{\mathsf{sep}}$ *be as before. Then:*

(1) There exists a formula I_0 *containing only variables which occur both in* A_0 *and* B_0 *such that* $(\mathsf{A}_0 \wedge N^A \wedge N^A_{\mathsf{sep}}) \models_{LI(\mathbb{Q})} I_0$ *and* $(\mathsf{B}_0 \wedge N^B \wedge N^B_{\mathsf{sep}}) \wedge I_0 \models_{LI(\mathbb{Q})} \bot$.

(2) The ground formula I *obtained from* I_0 *by recursively replacing every variable* c_t *introduced in the separation process with the term* t *is an interpolant for* $(\mathsf{A}_0 \wedge D_A) \wedge (\mathsf{B}_0 \wedge D_B)$, *i.e.:*

(i) I *contains only variables and extension functions common to* A *and* B;

(ii) $\mathsf{A}_0 \wedge D_A \models_{LI(\mathbb{Q})^{\Sigma}} I$ *and* $\mathsf{B}_0 \wedge D_B \wedge I \models_{LI(\mathbb{Q})^{\Sigma}} \bot$.

By Theorem 16 and Corollary 17 we know that if A and B are conjunctions of literals in linear arithmetic and uninterpreted function symbols such that $\mathsf{A} \wedge \mathsf{B}$ is unsatisfiable then there exists an interpolant; its existence is not influenced by the choice of the separating terms in the set T. The method terminates; its complexity is discussed in [32,33], and depends on the complexity of computing separating terms in linear arithmetic, and on the complexity of computing interpolants for conjunctions of *clauses* in LI. In order to compute an interpolant for $(\mathsf{A}_0 \wedge N^A \wedge N^A_{\mathsf{sep}}) \wedge (\mathsf{B}_0 \wedge N^B \wedge N^B_{\mathsf{sep}})$ one can use, for instance, the method discussed in Section 3.1.

Example 18 *We illustrate the ideas on a simple example. Consider the unsatisfiable conjunction* $A \wedge B$, *where:*

$$\mathsf{A}: \quad p \leq c \wedge c \leq q \wedge f(c) = 1 \qquad \mathsf{B}: \quad q \leq d \wedge d \leq p \wedge f(d) = 0.$$

We can compute an interpolant for $\mathsf{A} \wedge \mathsf{B}$ *using Theorem 16 and Corollary 17 as follows:*

Step 1: Purification. *After purification and flattening (with* $D = \{c_{f(c)} = f(c), c_{f(d)} = f(d)\}$*) and adding the corresponding instances of the congruence axioms we obtain:*

$$\mathsf{A}_0: \quad p \leq c \wedge c \leq q \wedge c_{f(c)} = 1 \qquad \mathsf{B}_0: \quad q \leq d \wedge d \leq p \wedge c_{f(d)} = 0$$
$$N_0: \quad c = d \rightarrow c_{f(c)} = c_{f(d)}$$

Step 2: Separation. *The clause in* N_0 *is mixed. We separate it as follow. We notice that* $\mathsf{A}_0 \wedge \mathsf{B}_0 \models c = d$*, and there is a term in the shared signature of* A *and* B*, namely* p *such that* $\mathsf{A}_0 \wedge \mathsf{B}_0 \models c = p \wedge p = d$*. We replace the clause in* N_0 *with the clauses:*

$$C_A: \quad c = p \rightarrow c_{f(c)} = c_{f(p)} \qquad C_B: \quad p = d \rightarrow c_{f(p)} = c_{f(d)}.$$

Step 3: Interpolant computation. *We now use a method for computing interpolants for sets of clauses in* $LI(\mathbb{Q})$ *to compute an interpolant* I_0 *for* $(\mathsf{A} \wedge C_A) \wedge (\mathsf{B} \wedge C_B)$*, i.e. for*

$$(p{\leq}c \wedge c{\leq}q \wedge c_{f(c)}{=}1 \wedge (c{=}p \rightarrow c_{f(c)}{=}c_{f(p)})) \wedge (q{\leq}d \wedge d{\leq}p \wedge c_{f(d)}{=}0 \wedge (p{=}d \rightarrow c_{f(p)}{=}c_{f(d)}))$$

i.e. – writing the formulae in disjunctive form – for $A_1 \vee A_2$ *and* $B_1 \vee B_2$*, where:*

$$A_1: \quad p{\leq}c \wedge c{<}q \wedge c_{f(c)}{=}1 \qquad A_2: \quad p{\leq}c \wedge c{\leq}q \wedge c_{f(c)}{=}1 \wedge c_{f(c)}{=}c_{f(p)}$$
$$B_1: \quad q{\leq}d \wedge d{<}p \wedge c_{f(d)}{=}0 \qquad B_2: \quad q{\leq}d \wedge d{\leq}p \wedge c_{f(d)}{=}0 \wedge c_{f(p)}{=}c_{f(d)}$$

By the remarks in Section 3.1, $I_0 = \bigvee_{j=1,2} \bigwedge_{i=1,2} I_{ij}$*, where* I_{ij} *is the interpolant of* A_i *and* B_j*. Such interpolants are e.g.:*

$$I_{11}: \quad p < q; \qquad I_{12}: \quad p < q; \qquad I_{21}: \quad p \leq q; \qquad I_{22}: \quad c_{f(p)} = 1$$

Thus, $I_0 = p < q \vee (p \leq q \wedge c_{f(p)} = 1)$*. From* I_0 *we compute the interpolant for* $\mathsf{A} \wedge \mathsf{B}$*:*

$$I = p < q \vee (p \leq q \wedge f(p) = 1)$$

by replacing the variable $c_{f(p)}$ *with the term* $f(p)$*.*

In Section 4.3 we will present an approach in which separation and interpolant computation are interleaved.

The proof of Theorem 16 and the correctness of the methods relies on the following properties of linear arithmetic: convexity with respect to equality atoms (Lemma 19) and separability of entailed inequalities (Lemma 20).

Lemma 19 *Linear arithmetic over* $\mathbb{R}$ *or over* $\mathbb{Q}$ *is convex w.r.t. equality atoms, i.e. for each conjunction* Γ *of literals and for every set of equalities* $s_i = t_i$*,* $i \in \{1, \ldots, n\}$*, if* $\Gamma \models \bigvee_{i=1}^{n} s_i = t_i$ *then* $\Gamma \models s_j = t_j$ *for some* $j \in \{1, \ldots, n\}$*.*

Lemma 20 *Let* $Ax \leq a$ *and* $Bx \leq b$ *be two conjunctions of constraints in linear arithmetic, and let* x_i *and* x_j*, where* $i, j \in \{1, \ldots, n\}$*, appear in* $Ax \leq a$ *and* $Bx \leq b$*, respectively.*

(1) If $Ax \leq a \wedge Bx \leq b$ implies $x_i \leq x_j$ then there exists a linear expression t over variables that are common to $Ax \leq a$ and $Bx \leq b$ such that $Ax \leq a$ implies $x_i \leq t$ and $Bx \leq b$ implies $t \leq x_j$ [35].
(2) If $Ax \leq a \wedge Bx \leq b$ implies $x_i = x_j$ then there exists a linear expression t over variables that are common to $Ax \leq a$ and $Bx \leq b$ such that $Ax \leq a \wedge Bx \leq b$ implies $x_i = t$ and $t = x_j$.

Proof. A proof is given in [35]. We present an alternative proof, based on Farkas' lemma. We represent the inequality $x_i \leq x_j$ using matrix notation as $cx \leq 0$, where c is a row vector such that $c_i = 1$, $c_j = -1$, and $c_k = 0$ for each $k \in \{1, \ldots, n\} \setminus \{i, j\}$. Since the conjunction $Ax \leq a \wedge Bx \leq b$ implies $x_i \leq x_j$, by Farkas' lemma [30] there exist non-negative row vectors λ and μ such that

$$\lambda A + \mu B = c, \qquad (*)$$
$$\lambda a + \mu b \leq 0. \qquad (**)$$

We define the term t to be the expression $-\lambda Ax + x_i + \lambda a$. Let t_k denote the coefficient before the variable x_k in the term t, for $k \in \{1, \ldots, n\}$. We show that t satisfies the required conditions.

First, we show that t does not contain local variables. Let x_k be local to $Ax \leq a$, which implies $k \neq j$ and $B_{|k} = 0$. If $k = i$ then we have $t_k = \lambda A_{|k} + 1$, and $\lambda A_{|k} + \mu B_{|k} = 1$, which follows from $(*)$. We conclude $t_k = 0$. If $k \neq i$ then from $(*)$ we have $\lambda A_{|k} + \mu B_{|k} = 0$, and hence $t_k = 0$. Let x_k be local to $Bx \leq b$, and hence $k \neq i$ and $A_{|k} = 0$. We have $t_k = A_{|k} = 0$.

The proof that $Ax \leq a$ implies $x_i \leq t$ is an application of Farkas' lemma, since $x_i \leq t$ is equivalent to $\lambda Ax \leq \lambda a$.

From $(*)$ we have $\mu B_{|i} = 1 - \lambda A_{|i}$, $\mu B_{|j} = -1 - \lambda A_{|j}$, and $\mu B_{|k} = -\lambda A_{|k}$ for each $k \in \{1, \ldots, n\} \setminus \{i, j\}$. Hence, μBx is equivalent to $-\lambda Ax + x_i - x_j$. Together with $(**)$ the implication between $Bx \leq b$ and $t \leq x_j$ follows from Farkas' lemma.

4.3 An interleaved hierarchical method

We now present an approach in which the computation of the interpolant is interleaved with the separation process. The idea is described in the algorithm in Figure 3. The algorithm contains several optimizations, which allow performing simultaneously the separation into an A-pure and a B-pure part and the interpolant construction. Termination and correctness of the algorithm are proved in Theorem 22.

We illustrate the algorithm $\textsc{Inter}_{LI(\mathbb{Q})^\Sigma}$ on the previous example:

$$\mathsf{A}_0 : \quad p \leq c \wedge c \leq q \wedge c_{f(c)} = 1 \qquad \mathsf{B}_0 : \quad q \leq d \wedge d \leq p \wedge c_{f(d)} = 0$$
$$N_0 : \quad c = d \rightarrow c_{f(c)} = c_{f(d)}$$

Now, instead of separating the mixed congruence axiom instance in N_0, we compute the terms t^+ and t^- using the algorithm Sep. We obtain

$$t^+ : \quad q \qquad t^- : \quad p \qquad I_0 : \quad q > p \qquad I_1 : \quad q \geq p \qquad t : \quad f(q)\ .$$

algorithm SEP
input
$Ax \leq a$ and $Bx \leq b$: constraints in matrix form
x_i, x_j : variables from x such that $B_{|i} = 0$, $A_{|j} = 0$, and $Ax \leq a \wedge Bx \leq b \models_{LI(\mathbb{Q})} x_i = x_j$
output
t^-, t^+ : expressions over common variables of $Ax \leq a$ and $Bx \leq b$ such that
$Ax \leq a \models_{LI(\mathbb{Q})} t^- \leq x_i \leq t^+$ and $Bx \leq b \models_{LI(\mathbb{Q})} t^+ \leq x_j \leq t^-$
vars
e^+, e^- : expression vectors
$\lambda^+, \mu^+, \lambda^-, \mu^-$: non-negative linear combinations
1 **begin**
2 $e^+ := \lambda^+ A + \mu^+ B$
3 $e^- := \lambda^- A + \mu^- B$
4 $\lambda^+, \mu^+ :=$ solution for $\lambda^+ \geq 0 \wedge \mu^+ \geq 0 \wedge \lambda^+ a + \mu^+ b \leq 0 \wedge$
5 $e_i^+ = 1 \wedge e_j^+ = -1 \wedge \bigwedge_{k \in \{1,\ldots,n\} \setminus \{i,j\}} e_k^+ = 0$
6 $\lambda^-, \mu^- :=$ solution for $\lambda^- \geq 0 \wedge \mu^- \geq 0 \wedge \lambda^- a + \mu^- b \leq 0 \wedge$
7 $e_i^- = -1 \wedge e_j^- = 1 \wedge \bigwedge_{k \in \{1,\ldots,n\} \setminus \{i,j\}} e_k^- = 0$
8
9 $t^+ := \mu^+ Bx + x_j - \mu^+ b$
$t^- := \lambda^- Ax + x_i - \lambda^- a$
return t^+, t^-
end.

Fig. 2. Algorithm SEP for the computation of separating terms.

The recursive call $\text{INTER}_{LI(\mathbb{Q})^\Sigma}(\mathsf{A}_0 \wedge c = t, \mathsf{B}_0 \wedge t = d, D \cup \{t = f(q)\}, \emptyset)$ returns $t \geq 1$. We combine it with I_0 and I_1 and obtain the following interpolant

$$q > p \vee (q \geq p \wedge f(q) \geq 1) \ .$$

Notation: Everywhere in this section we write $\models$ for the implication in the theory of linear arithmetic over rationals (reals) $\models_{LI(\mathbb{Q})}$.

Theorem 21 *The algorithm* SEP *in Figure 2 computes the claimed output.*

Proof. Lines 3 and 5 of the algorithm always succeed, since the implication $Ax \leq a \wedge Bx \leq b \models x_i = x_j$ holds, and hence, by Farkas' lemma, $x_i - x_j \leq 0$ and $-x_i + x_j \leq 0$ are deriviable from $Ax \leq a \wedge Bx \leq b$ using $(\lambda^+ \ \mu^+)$ and $(\lambda^- \ \mu^-)$, respectively.

We show 1) $Ax \leq a \models x_i \leq t^+$ and 2) $Bx \leq b \models t^+ \leq x_j$. The proof of the remaining two implications is similar.

By definition, $x_i \leq t^+$ is equivalent to $x_i - x_j - \mu^+ Bx \leq -\mu^+ b$. From lines 3 and 4 follows $\lambda^+ Ax + \mu^+ Bx = x_i - x_j$ and $\lambda^+ a \leq -\mu^+ b$, which proves 1). $t^+ \leq x_j$ is equivalent to $x_j + \mu^+ Bx - x_j \leq \mu^+ b$, which is derivable from $Bx \leq b$ using μ^+.

algorithm $\textsc{Inter}_{LI(\mathbb{Q})^\Sigma}$
input
$Ax \leq a$ and $Bx \leq b$: constraints in matrix form (obtained from flattening and purifying conjunctions A and B of (unit) literals in linear arithmetic and uninterpreted function symbols such that $\mathsf{A} \wedge \mathsf{B}$ is unsatisfiable)
D : definition of fresh variables created by flattening and purification of A and B
N_0 : instances of functionality axioms for functions from D
output
I: resulting interpolant
vars
I_0, I_1 : partial interpolants; t_i^-'s, t_i^+'s: separating terms

1 **begin**
2 **if** exists $(\bigwedge_{i=1}^n c_i = d_i \rightarrow c = d) \in N_0$ such that
3 $Ax \leq a \wedge Bx \leq b \models_{LI(\mathbb{Q})} \bigwedge_{i=1}^n c_i = d_i$ **then**
4 **if** c is A-local and d is B-local **then**
5 **for** each $i \in \{1, \ldots, n\}$ **do**
6 $t_i^+, t_i^- := \textsc{Sep}(Ax \leq a, Bx \leq b, c_i, d_i)$
7 **done**
8 $I_0 := \bigvee_{i=1}^n t_i^+ > t_i^-$
9 $I_1 := \bigwedge_{i=1}^n t_i^+ \geq t_i^-$
10 $f :=$ function symbol corresponding to $\bigwedge_{i=1}^n c_i = d_i \rightarrow c = d$
11 $t :=$ fresh variable
12 $D := D \cup \{t = f(t_1^+, \ldots, t_n^+)\}$
13 $Ax \leq a := Ax \leq a \wedge c = t$
14 $Bx \leq b := Bx \leq b \wedge t = d$
15 **else**
16 **if** c and d are A-local **then**
17 $I_0 := \textsc{Inter}_{LI(\mathbb{Q})}(Ax \leq a \wedge \neg(\bigwedge_{i=1}^n c_i = d_i),\ Bx \leq b)$
18 $I_1 := \top$
19 $Ax \leq a := Ax \leq a \wedge c = d$
20 **else** ($*$ c and d are B-local $*$)
21 $I_0 := \bot$
22 $I_1 := \textsc{Inter}_{LI(\mathbb{Q})}(Ax \leq a,\ Bx \leq b \wedge \neg(\bigwedge_{i=1}^n c_i = d_i))$
23 $Bx \leq b := Bx \leq b \wedge c = d$
24 **endif**
25 **endif**
26 $I := I_0 \vee (I_1 \wedge \textsc{Inter}_{LI(\mathbb{Q})^\Sigma}(Ax \leq a, Bx \leq b, D, N_0 \backslash \{\bigwedge_{i=1}^n c_i = d_i \rightarrow c = d\}))$
27 **else**
28 $I := \textsc{Inter}_{LI(\mathbb{Q})}(Ax \leq a, Bx \leq b)$
29 **endif**
return "interpolant I with expanded D"
end.

Fig. 3. Recursive algorithm $\textsc{Inter}_{LI(\mathbb{Q})^\Sigma}$ for the interpolation for linear arithmetic and uninterpreted function symbols using the algorithms Sep and $\textsc{Inter}_{LI(\mathbb{Q})}$ as subroutines.

Theorem 22 *The algorithm* $\textsc{Inter}_{LI(\mathbb{Q})^\Sigma}$ *in Figure 3 terminates and returns an interpolant* I *of* $\mathsf{A} \wedge \mathsf{B}$.

Proof. We prove that $\textsc{Inter}_{LI(\mathbb{Q})^\Sigma}$ terminates and computes an interpolant.

Termination follows from the finiteness of the set N_0. We prove that the returned formula I (the result of expanding the definitions in D in I) is an interpolant by applying well-founded induction over the number of recursive steps.

For the base step (see line 27), correctness of $\textsc{Inter}_{LI(\mathbb{Q})}$ implies that I, the result of expanding the definitions in D in I, is an interpolant.

For the induction step, we consider the three cases following the branches of the algorithm in lines 3 and 15. Let I_2 be the result of the recursive application of $\textsc{Inter}_{LI(\mathbb{Q})^\Sigma}$ in line 25, and I_0 and I_1 be the result of expanding definitions D in I_0 and I_1, respectively. In each case we prove that:

1) $\mathsf{A} \wedge \neg \mathsf{I}_0 \models \mathsf{I}_1$,
2) $\mathsf{A} \wedge \neg \mathsf{I}_0 \models \mathsf{I}_2$,
3) $\mathsf{I}_0 \wedge \mathsf{B} \models \bot$,
4) $\mathsf{I}_1 \wedge \mathsf{I}_2 \wedge \mathsf{B} \models \bot$, and
5) I is over A-B common symbols.

Case 1: We first consider the case when c is A-local and d is B-local. Then:

$$\mathsf{A} \wedge \neg \mathsf{I}_0 \models \bigwedge_{i=1}^{n} t_i^- \leq t_i^+ \wedge \bigwedge_{i=1}^{n} t_i^+ \leq t_i^- \;\models\; \bigwedge_{i=1}^{n} t_i^+ = t_i^-$$

which in turn implies 1). Due to the axiom instance $\bigwedge_{i=1}^{n} c_i = t_i^+ \rightarrow c = t$, and the fact that $\mathsf{A} \wedge \neg \mathsf{I}_0 \models \bigwedge_{i=1}^{n} t_i^+ = c_i \wedge \bigwedge_{i=1}^{n} c_i = t_i^-$, we know that $\mathsf{A} \wedge \neg \mathsf{I}_0 \models c = t$. It follows therefore that $\mathsf{A} \wedge \neg \mathsf{I}_0 \models (\mathsf{A} \wedge c = t) \wedge \neg \mathsf{I}_0 \models \mathsf{I}_2$ (since the interpolant I_2 is recursively computed for $\mathsf{A} \wedge c = t$). Hence, by induction hypothesis we have 2).

Line 5 of the algorithm establishes 3), since it establishes $\mathsf{B} \models_{LI(\mathbb{Q})^\Sigma} \neg \mathsf{I}_0$. Furthermore, by line 5 of the algorithm, $\mathsf{I}_1 \wedge \mathsf{B} \models \bigwedge_{i=1}^{n} t_i^+ = d = t_i^-$. Thus, $\mathsf{I}_1 \wedge \mathsf{B} \models t = d$. It follows therefore that $\mathsf{I}_1 \wedge \mathsf{B} \models \mathsf{I}_1 \wedge (\mathsf{B} \wedge t = d)$. Hence, using the fact that the interpolant I_2 is recursively computed for $\mathsf{B} \wedge t = d$, $\mathsf{I}_2 \wedge \mathsf{I}_1 \wedge \mathsf{B} \models \bot$. Thus, by induction hypothesis we have 4). Since $t_1^+, \ldots, t_n^+$ and I_2 are over A-B common symbols, so is I. This proves 5)

Case 2: We now consider the case when c and d are A-local. Then, 1) holds vacuously. From line 16 we have

$$\mathsf{A} \wedge \neg \mathsf{I}_0 \models \bigwedge_{i=1}^{n} c_i = d_i \models c = d.$$

Then, 2) follows from line 18 and the induction hypothesis. Line 16 and 25 establish 3) and 4), respectively. Since, c and d are A-local, expansion of $c_1, \ldots, c_n$ and $d_1, \ldots, d_n$ does not contain B-local symbols. Hence, I_0 is over A-B common symbols, which implies 5).

Case 3: The case when c and d are B-local is similar to the above case.

In spite of the fact that the procedure for computing interpolants for linear arithmetic is called as a "black box", and that our method does not require the existence of an a priori constructed resolution proof for building the interpolant, the complexity of the algorithm described in Figure 3 is comparable to that of other methods for interpolant generation which construct interpolants from proofs [15,16,23,35]. The complexity depends linearly on the length of the proof (which in this case is built 'online'). In addition, the complexity of the procedure used for "separating" equalities needs to be taken into account.

Theorem 23 *Assume that we start from an implementation such that in $LI(\mathbb{Q})$ for a formula of length m:*
(a) interpolants can be computed in time $g(m)$,
(b) P-interpolating terms can be computed in time $h(m)$,
(c) entailment can be checked in time $k(m)$.
Then the method described above allows to compute an interpolant in time of order $n^2 \cdot (k(n^2)+h(n^2))+g(n^2)+l$.

Problems (a)–(c) can be solved in polynomial time for sets of unit clauses [30] and in NP for sets of clauses [34]. Due to the specific form of the axioms in N_0 which need to be taken into account (Horn, with all premises being equalities), the sets of clauses which occur in the problems we consider may fall into tractable classes [17], for which satisfiability can be tested in polynomial time.

5 Experiments

This section presents our own experiments with the presented algorithms. For an additional evaluation we refer to [1], which presents the results of using INTER$_{LI(\mathbb{Q})}$ within the interpolation procedure CSISAT and demonstrates its practicality.

We implemented the presented algorithms in a tool called CLP-PROVER. [6] Although the presented algorithms are correct for both rational and real spaces, our implementation handles only rationals, which is due to the applied constraint solver [12]. CLP-PROVER is built in SICStus Prolog [20], which is a Constraint Logic Programming (CLP) system [13]. In particular, the CLP scheme requires that the constraint solver infers all equalities that are implied by the constraint store. This allows for an efficient implementation of the instantiation of functionality axioms, see the "choose C" step in Figure 3. We integrated CLP-PROVER into the predicate discovery procedure of the software verification tools BLAST [11] and ARMC [27]. The integration with ARMC is two-way, namely, interpolants generated by CLP-PROVER are used by ARMC to compute abstraction. The interface to BLAST is only used for comparing with the existing interpolating theorem prover FOCI [23].

Our experiments with BLAST on Windows device drivers provide a direct comparison with the FOCI tool, which is also integrated into BLAST. We used a

[6] CLP-PROVER homepage: `http://www.mpi-sws.org/~rybal/clp-prover/`.

Example	Number of queries	CLP-Prover time (s)				FOCI time (s)
		Solving LI part	Applying axioms	Total solving	Total	
`ntdrivers/kbfiltr.i`	139	0.13	0.02	0.15	0.46	0.55
`ntdrivers/diskperf.i`	747	0.38	0.21	0.59	2.68	3.72
`ntdrivers/floppy.i`	1082	0.61	0.36	0.97	3.97	4.91
`ntdrivers/cdaudio.i`	1060	2.23	0.20	2.43	4.92	4.80

Table 1. Experimental evaluation on examples from Blast distribution. (Memory consumption was not an issue.) 'Solving LI-part' is the time spent on solving the system of constraints that defines an interpolant in linear arithmetic. 'Applying axioms' is the time spent on testing entailment of premises of functionality axiom instances. 'Total solving' is the total time spent on constraint solving. 'Total' is the total time spent in CLP-Prover, which includes parsing, computation of constraint systems, constraint solving, etc.

3 GHz Linux PC, Blast 2.0 and applied CLP-Prover on 3,000 interpolation problems that are also passed to FOCI. The table shows that a constraint-based implementation can provide support for full linear arithmetic with competitive running time.

We applied ARMC to verify safety properties of train controller systems [26]. These examples depend crucially on the ability of our algorithm to handle strict inequalities directly. The running times were similar to the experiments with Blast. Additionally, we applied ARMC to verify absence of array bounds violations (90 checks) for a compact (200 LOC) but intricate C program that performs singular value decomposition. CLP-Prover spends 190 ms on constraint solving for 457 interpolation problems, and computes interpolants over up to four variables. Unfortunately, we could not compare the running times for these experiments with FOCI since the latter does not support strict inequalities (whose relaxation immediately leads to unacceptable loss of precision), and is restricted to the difference bounds fragment of linear arithmetic (i.e. predicates containing four variables cannot be discovered).

6 Conclusion and ongoing work

We presented a constraint-based algorithm for the synthesis of interpolants in linear arithmetic and interpreted function symbols. Our algorithm does not require a priori constructed proofs to derive interpolants, which is a difficult task. The algorithm uses a reduction to constraint solving problem in linear arithmetic, which can be efficiently solved by using a Linear Programming tools in a black-box fashion. Our experiments provide evidence for the practical applicability of the algorithm.

In ongoing work, we are exploring the constraint based setup to accommodate user-defined constraints on the form of the generated interpolant, which has promising applications in software verification. In particular, we would like to

compute interpolants that are elements of a predefined abstract domain relevant for static analysis, see e.g. [2].

Acknowledgements We thank Friedrich Eisenbrand for valuable discussions.

References

1. D. Beyer, D. Zufferey, and R. Majumdar. CSIsat: Interpolation for LA+EUF. In *Computer Aided Verification, 20th International Conference, CAV 2008. Proceedings. LNCS 5123*, pages 304–308, 2008.
2. B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *PLDI'2003: Programming Language Design and Implementation*, pages 196–207. ACM Press, June 7–14 2003.
3. A. R. Bradley, Z. Manna, and H. B. Sipma. Linear ranking with reachability. In *CAV'2005: Computer Aided Verification*, volume 3576 of *LNCS*, pages 491–504. Springer, 2005.
4. A. Cimatti, A. Griggio, and R. Sebastiani. Efficient interpolant generation in satisfiability modulo theories. In C. R. Ramakrishnan and J. Rehof, editors, *ools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008. LNCS 4963*, pages 397–412, 2008.
5. A. Cimatti, A. Griggio, and R. Sebastiani. Interpolant generation for utvpi. In R. A. Schmidt, editor, *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction. Proceedings. LNCS 5663*, pages 167–182, 2009.
6. M. Colón, S. Sankaranarayanan, and H. Sipma. Linear invariant generation using non-linear constraint solving. 420-432. In *CAV'2003: Computer Aided Verification*, volume 2725 of *LNCS*, pages 420–432. Springer, 2003.
7. P. Cousot. Proving program invariance and termination by parametric abstraction, lagrangian relaxation and semidefinite programming. In *VMCAI'2005: Verification, Model Checking, and Abstract Interpretation*, volume 3385 of *LNCS*, pages 1–24. Springer, 2005.
8. W. Craig. Linear reasoning. A new form of the Herbrand-Gentzen theorem. *J. Symb. Log.*, 22(3):250–268, 1957.
9. J. Esparza, S. Kiefer, and S. Schwoon. Abstraction refinement with Craig interpolation and symbolic pushdown systems. In *TACAS'2006: Tools and Algorithms for the Construction and Analysis of Systems*, volume 3920 of *LNCS*, pages 489–503. Springer, 2006.
10. A. Goel, S. Krstic, and C. Tinelli. Ground interpolation for combined theories. In R. A. Schmidt, editor, *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction. Proceedings. LNCS 5663*, pages 183–198, 2009.
11. T. A. Henzinger, R. Jhala, R. Majumdar, and K. L. McMillan. Abstractions from proofs. In *POPL'2004: Principles of Programming Languages*, pages 232–244. ACM Press, 2004.
12. C. Holzbaur. *OFAI clp(q,r) Manual, Edition 1.3.3.* Austrian Research Institute for Artificial Intelligence, Vienna, 1995. TR-95-09.

13. J. Jaffar and S. Michaylov. Methodology and implementation of a CLP system. In *ICLP'1987: Int. Conf. on Logic Programming*, volume 1, pages 196–218. MIT Press, 1987.
14. R. Jhala and K. L. McMillan. Interpolant-based transition relation approximation. In *CAV'2005: Computer Aided Verification*, volume 3576 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2005.
15. R. Jhala and K. L. McMillan. A practical and complete approach to predicate refinement. In *TACAS'2006: Tools and Algorithms for the Construction and Analysis of Systems*, volume 3920 of *LNCS*, pages 459–473. Springer, 2006.
16. D. Kapur, R. Majumdar, and C. G. Zarba. Interpolation for data structures. In *FSE'2006: Foundations of Software Engineering*, pages 105–116. ACM, 2006.
17. M. Koubarakis. Tractable disjunctions of linear constraints: basic results and applications to temporal reasoning. *Theor. Comput. Sci.*, 266(1-2):311–339, 2001.
18. L. Kovács and A. Voronkov. Interpolation and symbol elimination. In R. A. Schmidt, editor, *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction. Proceedings. LNCS 5663*, pages 199–213, 2009.
19. J. Krajícek. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *J. Symb. Log.*, 62(2):457–486, 1997.
20. T. I. S. Laboratory. *SICStus Prolog User's Manual*. Swedish Institute of Computer Science, PO Box 1263 SE-164 29 Kista, Sweden, October 2001. Release 3.8.7.
21. C. Lynch and Y. Tang. Interpolants for linear arithmetic in smt. In *Automated Technology for Verification and Analysis, 6th International Symposium, ATVA 2008. Proceedings. LNCS 5311*, pages 156–170, 2008.
22. K. L. McMillan. Interpolation and SAT-based model checking. In *CAV'2003: Computer Aided Verification*, volume 2725 of *LNCS*, pages 1–13. Springer, 2003.
23. K. L. McMillan. An interpolating theorem prover. *Theor. Comput. Sci.*, 345(1):101–121, 2005.
24. K. L. McMillan. Lazy abstraction with interpolants. In *CAV'2006: Computer Aided Verification*, volume 4144 of *LNCS*, pages 123–136. Springer, 2006.
25. K. L. McMillan. Quantified invariant generation using an interpolating saturation prover. In C. R. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008. LNCS 4963*, pages 413–427, 2008.
26. R. Meyer, J. Faber, and A. Rybalchenko. Model checking duration calculus: A practical approach. In *ICTAC'2006: Int. Colloq. on Theoretical Aspects of Computing*, volume 4281 of *LNCS*, pages 332–346. Springer, 2006.
27. A. Podelski and A. Rybalchenko. ARMC: the logical choice for software model checking with abstraction refinement. In *PADL'2007: Practical Aspects of Declarative Languages, LNCS 4354*, pages 245–259. Springer, 2007.
28. P. Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *J. Symb. Log.*, 62(3):981–998, 1997.
29. A. Rybalchenko and V. Sofronie-Stokkermans. Constraint solving for interpolation. In *VMCAI*, pages 346–362, 2007.
30. A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons Ltd., 1986.
31. V. Sofronie-Stokkermans. Hierarchic reasoning in local theory extensions. In *CADE'2005: Int. Conf. on Automated Deduction*, volume 3632 of *LNCS*, pages 219–234. Springer, 2005.

32. V. Sofronie-Stokkermans. Interpolation in local theory extensions. In *IJCAR'2006: Int. Joint Conf. on Automated Reasoning*, volume 4130 of *LNCS*, pages 235–250. Springer, 2006.
33. V. Sofronie-Stokkermans. Interpolation in local theory extensions. *Logical Methods in Computer Science*, 4(4), 2008.
34. E. Sontag. Real addition and the polynomial hierarchy. *Information Processing Letters*, 20(3):115–120, 1985.
35. G. Yorsh and M. Musuvathi. A combination method for generating interpolants. In *CADE'2005: Int. Conf. on Automated Deduction*, volume 3632 of *LNCS*, pages 353–368. Springer, 2005.