Constraint Solving for Interpolation

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

[1] Ecole Polytechnique Fédérale de Lausanne
[2] Max-Planck-Institut für Informatik, Saarbrücken

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 [5] 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. [6,10,11,12,16,17,18,26].

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 [14,21,17]). 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 [11,17]. 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 [7].

We propose an algorithm for the generation of interpolants for the combined theory of linear arithmetic and uninterpreted function symbols that does not

B. Cook and A. Podelski (Eds.): VMCAI 2007, LNCS 4349, pp. 346–362, 2007.

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 LI for the generation of interpolants for linear arithmetic only, which is based on a reduction to constraint solving. The algorithm LI 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 LIUIF for generating interpolants in combination of linear arithmetic with uninterpreted function symbols, following the hierarchical style of [23,24]. It applies the algorithm LI 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 [7] and ARMC [20]. Our experiments with Blast on Windows device drivers provide a direct comparison with the existing tool FOCI [17], 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 [19], 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.

Related work. Our algorithm differs from the existing methods for the interpolant generation [11,12,17,26] in the following key points. Being constraint based, our algorithm does not require a priori constructed resolution 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 [23,24] 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 [24], 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 [26] 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 [12]. 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.

The "split" prover [11] 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. [2,3,4]. 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.

Structure of the paper. We introduce the necessary preliminaries in Section 2. We describe the algorithm LI for the synthesis of constrained interpolants in linear arithmetic in Section 3, and its extension LIUIF for the handling of uninterpreted function symbols in Section 4. We briefly describe our implementation and experimental results in Section 5.

2 Preliminaries

In what follows we will use the following notations:

Linear constraints over rational and real spaces. 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 inequalites 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 $x \leq -1$.

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 similarily. In what follows, the definitions are given for the case of linear rational arithmetic. Similar definitions can also be given for $LI(\mathbb{R})^\Sigma$. A model $\mathcal{M}$ of $LI(\mathbb{Q})^\Sigma$ is a model of $LI(\mathbb{Q})$ with universe M and with 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 (i.e. they are free).

Truth, satisfiability and entailment w.r.t. a theory. 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. 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.

Interpolants. A theory $\mathcal{T}$ has interpolation if, for all formulae ϕ and ψ in the signature of $\mathcal{T}$, if $\phi \models_\mathcal{T} \psi$ then there exists a formula I containing only symbols which occur in both ϕ and ψ such that $\phi \models_\mathcal{T} I$ and $I \models_\mathcal{T} \psi$. An alternative formulation in the model-checking literature is:

> If $\phi \wedge \psi \models_\mathcal{T} \bot$ then there exists a formula I containing only symbols which occur in both ϕ and ψ such that $\phi \models_\mathcal{T} I$ and $\psi \wedge I \models_\mathcal{T} \bot$.

First order logic has interpolation [5]. However, even if ϕ and ψ are very simple (e.g. quantifier-free or conjunctions of atoms), I may still be an arbitrary formula. In many applications it is important to find *simple* interpolants: for instance, if ϕ and ψ are quantifier-free formulae, we are often interested in the existence of *quantifier-free* interpolants. We say that a theory $\mathcal{T}$ has *quantifier-free* interpolants if for all quantifier-free formulae A and B:

> If $\mathsf{A} \wedge \mathsf{B} \models_\mathcal{T} \bot$ there exists a quantifier-free formula I over the common variables of A and B such that $\mathsf{A} \models_\mathcal{T} I$ and $I \wedge \mathsf{B} \models_\mathcal{T} \bot$.

3 Linear Interpolants

In this section we present an algorithm LI (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 LI 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

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
$\Phi := \ \lambda \geq 0 \wedge \mu \geq 0 \wedge i = \lambda A \wedge \delta = \lambda a \wedge \lambda A + \mu B = 0$
if exist λ, μ, i, δ satisfying $\Phi \wedge \lambda a + \mu b \leq -1$ **then**
(∗ 1st branch ∗)
return $ix \leq \delta$
else if exist λ, μ, i, δ satisfying $\Phi \wedge \lambda a + \mu b \leq 0 \wedge \lambda^{\mathsf{lt}} \neq 0$ **then**
(∗ 2nd branch ∗)
return $ix < \delta$
else if exist λ, μ, i, δ satisfying $\Phi \wedge \lambda a + \mu b \leq 0 \wedge \mu^{\mathsf{lt}} \neq 0$ **then**
(∗ 3rd branch ∗)
return $ix \leq \delta$
end.

x	A	a	λ	A^{lt}	A^{le}	a^{lt}	a^{le}	λ^{lt}	λ^{le}
$n \times 1$	$m_A \times n$	$m_A \times 1$	$1 \times m_A$	$m_{A^{\mathsf{lt}}} \times n$	$m_{A^{\mathsf{le}}} \times n$	$m_{A^{\mathsf{lt}}} \times 1$	$m_{A^{\mathsf{le}}} \times 1$	$1 \times m_{A^{\mathsf{lt}}}$	$1 \times m_{A^{\mathsf{le}}}$

Fig. 1. Algorithm LI for the synthesis of linear interpolants. The table shows dimensions of matrices and vectors used in the algorithm. The dimensions for $Bx \leqslant b$, μ, μ^{lt}, and μ^{le} are fixed in a similar fashion.

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 Motzkin's transposition theorem [22]. The proofs of Theorems 2 and 3 provide a formal explanation of the correspondence.

Theorem 1 ((Motzkin's) transposition theorem [22]). *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 1. We simulate the algorithm LI 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 LI 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 2 (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 $\lhd \in \{\leq, <\}$.*

The correctness of the algorithm LI is stated in the following theorem.

Theorem 3 (Algorithm LI: Soundness and completeness). *The algorithm* LI *is sound and complete: It always produces a linear interpolant, by taking the 1st, 2nd or 3rd branch.*

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 [17] or [26]: in a DPLL-style procedure partial interpolants are generated for the unsatisfiable branches and then recombined using ideas of Pudlák [21].

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*) [23]. 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 [24].

We begin by giving the main idea of the hierarchical calculus for extensions with free function symbols in [23] (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 $c_1, \ldots, c_n$[1] 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 [23], 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 4.

Theorem 4 ([23]). *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 4 is a satisfiability problem for quantifier-free clauses in linear rational arithmetic. We thus reduced, hierarchically, the problem of testing

[1] 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".

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 2. 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:

$$\begin{array}{llll} \mathsf{A}_0: & a_1 = c + 5 \wedge a_2 \geq c + 1, & D_A: & a_1 = g(a) \wedge a_2 = f(a_1), \\ \mathsf{B}_0: & b_1 = d + 4 \wedge d = c + 1 \wedge b_2 < c + 1, & D_B: & b_1 = h(b) \wedge b_2 = f(b_1). \end{array}$$

Step 2: Hierarchical reasoning. By Theorem 4 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:	$a_1{=}g(a) \wedge a_2{=}f(a_1)$	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:	$b_1{=}h(b) \wedge b_2{=}f(b_1)$	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:

(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).

Flattening and purification do not influence the existence of interpolants [24]: 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 4, $\mathsf{A}_0 \wedge D_A \wedge \mathsf{B}_0 \wedge D_B \models_{LI(\mathbb{Q})^{\Sigma}} \bot$ if and only if $\mathsf{A}_0 \wedge \mathsf{B}_0 \wedge N_0 \models_{LI(\mathbb{Q})} \bot$, 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 4 cannot be used directly for generating a ground interpolant.

Example 3. 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 4. 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. 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. The correctness of the method relies on the following properties of linear arithmetic: convexity with respect to equality atoms (Lemma 1) and separability of entailed inequalities (Lemma 2).

Lemma 1. *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 2. *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 $t \leq x_j$ [26].
(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 $t = x_j$.

We show that N_{mix} can be replaced 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, 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 5 ([24]). *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 $f(t_1, \ldots, t_n)$.

A direct consequence of Theorem 5 is the possibility of hierarchically generating interpolants in $LI(\mathbb{Q})^{\Sigma}$.

Corollary 1 ([24]). *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 5 and Corollary 1 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 [24], 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.

We now present an alternative approach, in which the computation of the interpolant is interleaved with the separation process. The idea is described in the algorithm in Figure 2. The algorithm is based on Theorem 5 and Corollary 1, but 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 what follows.

Theorem 6. *The algorithm in Figure 2 terminates and returns an interpolant I of $\mathsf{A} \wedge \mathsf{B}$.*

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 2 is comparable to that of other methods for interpolant generation which construct interpolants from proofs [11,12,17,26]. 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 7. *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 [22] and in NP for sets of clauses [25]. 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 [13], for which satisfiability can be tested in polynomial time.

5 Experiments

We implemented the presented algorithms in a tool called CLP-Prover.[2] 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 [8]. CLP-Prover is built in SICStus Prolog [15], which is a Constraint

[2] CLP-Prover homepage: `http://mtc.epfl.ch/~rybalche/clp-prover/`.

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 : definitions for fresh variables created by flattening and purification of A and B
N_0 : instances of functionality axioms for functions from D
output
I: the resulting interpolant
local vars
I_0, I_1, I_2 : partial interpolants; t_i^-, t_i^+: separating terms
begin
if $N_0 \neq \emptyset$ **then**
choose $C : \bigwedge_{i=1}^n c_i = d_i \rightarrow c = d$ from N_0
such that $Ax \leq a \wedge Bx \leq b \models_{LI(\mathbb{Q})} \bigwedge_{i=1}^n c_i = d_i$
(assume C is an instance of the functionality axiom for $f \in \Sigma$)
for each $i \in \{1, \ldots, n\}$ **do**
compute t_i^+ and t_i^- over A-B-common variables such that
$Ax \leq a \models_{LI(\mathbb{Q})} c_i \leq t_i^+$ and $Bx \leq b \models_{LI(\mathbb{Q})} t_i^+ \leq d_i$ and
$Ax \leq a \models_{LI(\mathbb{Q})} c_i \geq t_i^-$ and $Bx \leq b \models_{LI(\mathbb{Q})} t_i^- \geq d_i$
done
$I_0 := \mathsf{false}$
$I_1 := \mathsf{true}$
for each $k :=$ index within $\{1, \ldots, n\}$ such that $t_k^+ \neq t_k^-$ **do**
$I_0 := I_0 \vee t_k^+ > t_k^-$
$I_1 := I_1 \wedge t_k^+ = t_k^-$
$Ax \leq a := Ax \leq a \wedge t_k^+ = t_k^-$
$Bx \leq b := Bx \leq b \wedge t_k^- = t_k^+$
done
$t :=$ fresh variable; $D := D \cup \{t = f(t_1^+, \ldots, t_n^+)\}$
$Ax \leq a := Ax \leq a \wedge c = t$
$Bx \leq b := Bx \leq b \wedge t = d$
$I_2 :=$ result of recursively applying the procedure
for the new $Ax \leq a$, $Bx \leq b$ and D, and $N_0 \backslash \{C\}$
$I := I_0 \vee (I_1 \wedge I_2)$ where each definition from D is applied
else
$ix \leq \delta :=$ result of applying LI on $Ax \leq a$ and $Bx \leq b$
$I :=$ $ix \leq \delta$ where each definition from D is applied
endif
return "interpolant I"
end.

Fig. 2. Algorithm LIUIF for the synthesis of constrained interpolants for linear arithmetic and uninterpreted function symbols. LIUIF uses the algorithm LI as a subroutine.

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.

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

Logic Programming (CLP) system [9]. 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 2. We integrated CLP-PROVER into the predicate discovery procedure of the software verification tools BLAST [7] and ARMC [20]. 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 [17].

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 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 [19]. 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. [1].

Acknowledgements. We thank Friedrich Eisenbrand for valuable discussions.

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.

References

1. 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.
2. 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.
3. 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.
4. 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.
5. W. Craig. Linear reasoning. A new form of the Herbrand-Gentzen theorem. *J. Symb. Log.*, 22(3):250–268, 1957.
6. 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.
7. 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.

8. C. Holzbaur. *OFAI clp(q,r) Manual, Edition 1.3.3*. Austrian Research Institute for Artificial Intelligence, Vienna, 1995. TR-95-09.
9. J. Jaffar and S. Michaylov. Methodology and implementation of a CLP system. In *ICLP'1987: Int. Conf. on Logic Programming*, volume 1. MIT Press, 1987.
10. 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.
11. 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.
12. D. Kapur, R. Majumdar, and C. G. Zarba. Interpolation for data structures. In *FSE'2006: Foundations of Software Engineering*. ACM, 2006. To appear.
13. M. Koubarakis. Tractable disjunctions of linear constraints: Basic results and applications to temporal reasoning. *Theoretical Computer Science*, 266(1-2): 311–339, 2001.
14. J. Krajícek. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *J. Symb. Log.*, 62(2):457–486, 1997.
15. 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.
16. K. L. McMillan. Interpolation and SAT-based model checking. In *CAV'2003: Computer Aided Verification*, volume 2725 of *LNCS*, pages 1–13. Springer, 2003.
17. K. L. McMillan. An interpolating theorem prover. *Theor. Comput. Sci.*, 345(1):101–121, 2005.
18. K. L. McMillan. Lazy abstraction with interpolants. In *CAV'2006: Computer Aided Verification*, volume 4144 of *LNCS*, pages 123–136. Springer, 2006.
19. 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.
20. 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. Springer, 2007. to appear.
21. P. Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *J. Symb. Log.*, 62(3):981–998, 1997.
22. A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons Ltd., 1986.
23. 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.
24. 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.
25. E. Sontag. Real addition and the polynomial hierarchy. *Information Processing Letters*, 20(3):115–120, 1985.
26. 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.