J Autom Reasoning (2011) 47:341–367
DOI 10.1007/s10817-011-9237-y

An Interpolating Sequent Calculus for Quantifier-Free Presburger Arithmetic

Angelo Brillout · Daniel Kroening · Philipp Rümmer · Thomas Wahl

Received: 24 July 2011 / Accepted: 12 August 2011 / Published online: 9 September 2011

Abstract Craig interpolation has become a versatile tool in formal verification, used for instance to generate program assertions that serve as candidates for loop invariants. In this paper, we consider Craig interpolation for *quantifier-free Presburger arithmetic* (QFPA). Until recently, quantifier elimination was the only available interpolation method for this theory, which is, however, known to be potentially costly and inflexible. We introduce an interpolation approach based on a sequent calculus for QFPA that determines interpolants by annotating the steps of an unsatisfiability proof with *partial interpolants*. We prove our calculus to be sound and complete. We have extended the Princess theorem prover to generate interpolating proofs, and applied it to a large number of publicly available Presburger arithmetic benchmarks. The results document the robustness and efficiency of our interpolation procedure. Finally, we compare the procedure against alternative interpolation methods, both for QFPA and linear rational arithmetic.

Keywords Sequent calculus · Presburger arithmetic · Craig interpolation

Mathematics Subject Classification (2010) 03B70

This paper is an extended version of a publication that appeared at IJCAR [3].

Supported by the Engineering and Physical Sciences Research Council (EPSRC) under grant no. EP/G026254/1, by the EU FP7 STREP MOGENTES, by the EU FP7 STREP PINCETTE, and by the EU ARTEMIS project CESAR.

A. Brillout
ETH Zurich, Zurich, Switzerland

D. Kroening · T. Wahl
Computer Science Department, Oxford University, Oxford, UK

P. Rümmer (✉)
Department of Information Technology, Uppsala University, Uppsala, Sweden
e-mail: ph_r@gmx.net

1 Introduction

Craig interpolation, a technique known to logicians since the 1950s [6], has recently emerged in formal verification as a practical approximation method. Its applications range from accelerating convergence of fixpoint calculations for finite-state or infinite-state systems, to generating assertions at intermediate program points that serve as candidates for loop invariants and thus assist the safety analysis of programs. Given two formulae A and C such that A implies C, written $A \Rightarrow C$, an interpolant is a formula I such that $A \Rightarrow I$, $I \Rightarrow C$, and all of I's symbolic constants occur in both A and C. Interpolants exist for any two first-order formulae A and C such that $A \Rightarrow C$.

In program verification, we typically consider the special case $C = \neg B$ of the above formulation, so that the validity of $A \Rightarrow C$ is tantamount to the unsatisfiability of $A \wedge B$. This latter formula is used to encode the reachability of an error condition along a length-bounded path of the program; its unsatisfiability thus proves the program correct along paths up to that length. In order to support expressive programming languages, much effort has been invested in algorithms that compute interpolants for various theories. As a result, efficient interpolation methods are known for propositional logic, linear arithmetic over the reals with uninterpreted functions [1, 15, 21], datastructures like arrays and sets [12], and fragments of integer arithmetic such as difference-bound logic and logics with linear equalities and constant-divisibility predicates. For these fragments, an interpolant can be derived in time polynomial in the size of the input formulae.

Given these encouraging results and the significance of integer arithmetic in software analysis, there have been several attempts recently to interpolate the full range of quantifier-free linear integer arithmetic, also known as *Presburger arithmetic* and denoted QFPA in this paper. This theory has been used, for example, to model the behavior of infinite-state programs and of hardware designs. While interpolation for QFPA can in principle be achieved by quantifying out variables local to one of the input formulae, followed by quantifier elimination, this method suffers from various serious practical impediments, including the high complexity of the elimination procedure (see Section 3 for a detailed illustration). An incomplete interpolation procedure based on linear programming techniques was introduced in [14]. The first complete proof-based interpolation procedure for QFPA was introduced in our IJCAR 2010 paper [3], which we extend in the present article by a refined discussion, and by a significantly expanded experimental analysis.

The interpolation method in this article extracts interpolants directly from an unsatisfiability proof for $A \wedge B$, as suggested e.g. in [1, 10, 14, 15]. We first present a sound and complete proof system for QFPA based on a *sequent calculus*. We then augment the proof rules with labeled formulae and *partial interpolants*—proof annotations that, at the root of a closed proof, reduce to interpolants. In practice, the resulting interpolating proof system can be used to extend an existing unsatisfiability proof to one that interpolates. It can also serve as a replacement of the non-interpolating proof system, allowing the calculation of an interpolant on the fly. We prove our interpolating calculus to be *sound and complete* for QFPA. Our completeness result states that, for any valid implication, there exists a proof of its validity in our calculus, and the proof can be annotated with partial interpolants satisfying the proof rules. This theorem can be generalized to a result stating the existence of *interpolant chains* for conjunctions with arbitrarily many conjuncts.

In the case of QFPA, the primary difficulty when extracting interpolants from a proof is the treatment of *mixed cuts*: applications of a cut rule (such as Gomory cuts [22] or the Omega rule [17]) to inequalities that have been derived as linear combinations of inequalities from both A and B. Our work extends earlier interpolation procedures for linear arithmetic, in particular [14, 15], by defining an interpolating cut rule called STRENGTHEN that can handle even mixed cuts. The rule subsumes a variety of cut rules for integer linear programming, including the above-mentioned Gomory cuts and the Omega rule, so that interpolants can be extracted from proofs using either of those rules by reduction to STRENGTHEN.

To implement our interpolation method, we have extended the PRINCESS theorem prover [20] to generate proofs, using the proof rules presented in this paper. We have applied the interpolating prover to a large number of publicly available linear integer arithmetic benchmarks, taken from the QF-LIA category of the SMT library. We compare the efficiency of the prover to alternative QFPA interpolation procedures, including those based on quantifier elimination (QE) and on linear programming [13]. We also compare the prover with CSISAT [1], a constraint-based [21] interpolation procedure for linear *rational* arithmetic: since interpolation methods for rational arithmetic have been known and developed for several years, it is interesting to measure the overhead imposed by integer reasoning on problems that can also be interpolated using rational methods. Our results demonstrate the strengths of our interpolation approach in terms of both time and interpolant size.

2 Preliminaries

Presburger Arithmetic We assume familiarity with classical first-order logic (see, e.g., [8]). Let x range over an infinite set X of variables, c over an infinite set C of constant symbols, and α over the integers $\mathbb{Z}$. The syntax of *Presburger arithmetic* is defined by the following grammar:

$$\begin{aligned} \phi &::= \quad t \doteq 0 \mid t \leq 0 \mid \alpha \mid t \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \forall x.\phi \mid \exists x.\phi \\ t &::= \quad \alpha \mid c \mid x \mid \alpha t + \cdots + \alpha t \end{aligned}$$

The symbol t denotes terms of linear arithmetic. To enforce canonicity, we assume that terms are implicitly simplified to 0 or to the form $\alpha_1 t_1 + \cdots + \alpha_n t_n$, in which $0 \notin \{\alpha_1, \ldots, \alpha_n\}$, and $t_1, \ldots, t_n$ are pairwise distinct variables, constants, or 1. Further, we only allow 0 as the right-hand side of equalities and inequalities.

This paper is concerned with interpolation for quantifier-free arithmetic and therefore permits quantifiers only in restricted forms: (i) universal (resp., existential) quantifiers under an even (resp., odd) number of negations. Such quantifiers effectively limit the theory to the existential fragment of Presburger arithmetic and can thus be eliminated by Skolemization; and (ii) implicitly to express divisibility by integer constants, namely in the form of the *stride constraint* $\alpha \mid t$, which is equivalent to $\exists s.\, \alpha s - t \doteq 0$. We permit such expressions to enable quantifier-free interpolants for formulae such as $y - 2x \doteq 0 \wedge y - 2z - 1 \doteq 0$, with interpolant $2 \mid y$. QFPA denotes the logic generated by the above grammar, restricted to the aforementioned quantification rules.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{\ldots, \mathit{false} \vdash}\,\text{CLOSE-LEFT}'}{\ldots, 1 \le 0 \vdash}\,\text{SIMP}'}{\ldots, -x \le 0, x+1 \le 0 \vdash}\,\text{FM-ELIM}'}{\ldots, -x \le 0, -b+x \le 0, 3b-2x+1 \le 0 \vdash}\,\text{FM-ELIM}'}{\ldots, -2x \le 0, -2b+2x-1 \le 0, 3b-2x+1 \le 0 \vdash}\,\text{SIMP}'^{+}}{\ldots, -2x \le 0, -2b+2x-1 \le 0, c-3b-1 \doteq 0, c-2x \le 0 \vdash}\,\text{RED}'}{a-2x \doteq 0, -a \le 0, 2b-a \le 0, -2b+a-1 \le 0, c-3b-1 \doteq 0, c-a \le 0 \vdash}\,\text{RED}'^{+}}{a-2x \doteq 0 \wedge -a \le 0 \wedge 2b-a \le 0 \wedge -2b+a-1 \le 0 \wedge c-3b-1 \doteq 0 \wedge c-a \le 0 \vdash}\,\text{AND-LEFT}'^{+}$$

Fig. 1 Unsatisfiability proof for the examples of Section 3

We use the abbreviations *true* and *false* for the equalities $0 \doteq 0$ and $1 \doteq 0$, and $\phi \to \psi$ as abbreviation for $\neg\phi \vee \psi$. Simultaneous substitution of terms $t_1, \ldots, t_n$ for variables $x_1, \ldots, x_n$ in ϕ is denoted by $[x_1/t_1, \ldots, x_n/t_n]\phi$; we assume that variable capture is avoided by renaming bound variables as necessary. As a short-hand notation, we sometimes quantify over constants (as in $\forall c.\phi$) and assume that the constants are implicitly replaced by fresh variables. The semantics of Presburger arithmetic is defined over the universe $\mathbb{Z}$ of integers in the standard way [8].

Gentzen-Style Sequent Calculi If Γ, Δ are finite sets of formulae without free variables, then $\Gamma \vdash \Delta$ is a *sequent*. The sequent is *valid* if the formula $\bigwedge \Gamma \to \bigvee \Delta$ is valid. A calculus *rule* is a binary relation between a finite set of sequents called the premises, and a sequent called the conclusion. A sequent calculus rule is *sound* if, for all instances

$$\frac{\Gamma_1 \vdash \Delta_1 \quad \cdots \quad \Gamma_n \vdash \Delta_n}{\Gamma \vdash \Delta}$$

whose premises $\Gamma_1 \vdash \Delta_1, \ldots, \Gamma_n \vdash \Delta_n$ are valid, the conclusion $\Gamma \vdash \Delta$ is valid, too. Proofs are trees growing upwards, in which each node is labeled with a sequent, and each non-leaf node is related to the node(s) directly above it through an instance of a calculus rule. A proof is *closed* if it is finite and all leaves are justified by an instance of a rule without premises. The interpolating sequent calculus presented in this paper extends the ground fragment of the sound and complete sequent calculus for QFPA in [20]. As an example, Fig. 2 shows most of the rules used in the proof in Fig. 1 (the proof is discussed below in Section 3).

3 A Motivating Example

Consider the following program with variables ranging over unbounded integers:

```
if (a == 2*x && a >= 0) {
  b = a / 2; c = 3*b + 1;
  assert (c > a);
}
```

We would like to verify the assertion in the program. To this end, the program is translated into the QFPA formula below. Note that `b = a / 2` is converted into a conjunction of two inequalities, and that the assertion is negated:

$$a - 2x \doteq 0 \wedge -a \leq 0 \wedge 2b - a \leq 0 \wedge -2b + a - 1 \leq 0 \wedge c - 3b - 1 \doteq 0 \wedge c - a \leq 0 \quad (1)$$

The unsatisfiability of (1) implies that no run of the program violates the assertion. Figure 1 shows a refutation of (1) in the Gentzen-style sequent calculus used in this paper (the right-hand side Δ happens to be empty in all sequents, which is not true in general). We add the prime symbol $'$ to the rule names to distinguish them from the interpolating rules introduced later. The proof starts with the conjunction (1) in the bottom sequent of the tree. Repeatedly applying the rule AND-LEFT$'$ (denoted AND-LEFT$'^{+}$) splits the conjunction into a list of arithmetic literals. The equality $a - 2x \doteq 0$ is used to reduce the inequalities $-a \leq 0$, $-2b + a - 1 \leq 0$, and $c - a \leq 0$ by means of substitution (rule RED$'$). Similarly, $c - 3b - 1 \doteq 0$ is used to reduce $c - 2x \leq 0$. The inequalities $-2x \leq 0$ and $-2b + 2x - 1 \leq 0$ are simplified (rule SIMP$'$) by eliminating the coefficient 2; in the latter inequality, this requires rounding. Unsatisfiability of the remaining inequalities follows from two applications of the Fourier–Motzkin rule FM-ELIM$'$, and the proof can be closed. A selection of the rules applied in Fig. 1 are shown in Fig. 2.

Interpolants for unsatisfiable formulae like (1) can reveal additional information about the program being investigated, for instance intermediate assertions. Suppose we want to compute an invariant for the program point immediately after `b = a / 2`. Let A denote the part of formula (1) encoding the program up to this point; B the rest. We can obtain an interpolant for the unsatisfiable formula $A \wedge B$ by quantifying out the A-local variables, i.e., variable x, from A:

$$\exists x.\, (a - 2x \doteq 0 \wedge -a \leq 0 \wedge 2b - a \leq 0 \wedge -2b + a - 1 \leq 0)\,,$$

which simplifies via *quantifier elimination* (QE) to $-a \leq 0 \wedge 2b - a \doteq 0$. Existentially quantifying out the local variables from A (or, universally, the local variables from B) always returns the strongest (respectively, weakest) interpolant for an unsatisfiable formula. These "extremal" interpolants may be very large, however. Suppose we modify the conditional in the program by adding further conjuncts that are unnecessary for the safety of the program:

$$\texttt{if (a == 2*x \&\& a >= 0 \&\& a >= } n\texttt{*y - } \frac{n}{2} \texttt{ \&\& a <= } n\texttt{*y) \{} \quad (2)$$

$$\frac{\Gamma, \phi, \psi \vdash \Delta}{\Gamma, \phi \wedge \psi \vdash \Delta}\ \text{AND-LEFT}' \qquad \frac{\Gamma \vdash \phi, \psi, \Delta}{\Gamma \vdash \phi \vee \psi, \Delta}\ \text{OR-RIGHT}'$$

$$\frac{*}{\Gamma, \mathit{false} \vdash \Delta}\ \text{CLOSE-LEFT}' \qquad \frac{*}{\Gamma \vdash \mathit{true}, \Delta}\ \text{CLOSE-RIGHT}'$$

$$\frac{\Gamma, t \doteq 0 \vdash \phi[s + \alpha \cdot t], \Delta}{\Gamma, t \doteq 0 \vdash \phi[s], \Delta}\ \text{RED}' \qquad \frac{\Gamma, s \leq 0, t \leq 0, \alpha s + \beta t \leq 0 \vdash \Delta}{\Gamma, s \leq 0, t \leq 0 \vdash \Delta}\ \text{FM-ELIM}'$$

Fig. 2 Some rules of the (non-interpolating) calculus for QFPA from [20]

where $n \in \mathbb{Z}$ is a parameter such that $2 \mid n$. The strongest (quantifier-free) interpolant, denoted I_s^n, grows linearly in n and thus exponentially in the program size:

$$I_s^n \equiv -a \leq 0 \wedge 2b - a \doteq 0 \wedge \left(n \mid a \vee n \mid (a+1) \vee \cdots \vee n \mid \left(a + \frac{n}{2}\right)\right).$$

A weaker but much more succinct interpolant is the inequality $-3b + a \leq 0$. We demonstrate in this paper that *proof-based* interpolation provides a way of obtaining such succinct interpolants. Proofs can compactly encode the unsatisfiability of a formula and abstract away irrelevant facts, enabling the extraction of succinct interpolants. This is of particular importance for program verification, where interpolants carrying unnecessary details can delay or prevent the discovery of inductive invariants (e.g., [16]). We therefore propose to *lift* proofs of unsatisfiability to *interpolating proofs*. This way, we avoid many disadvantages of QE-based interpolation, namely (i) its high complexity, (ii) its inflexibility in always returning a strongest or weakest interpolant, and (iii) the need to restart from scratch in order to consider a new partitioning of the unsatisfiable formula into A and B (in contrast, a proof-based method can extract many interpolants from a single proof).

4 An Interpolating Sequent Calculus for QFPA

In order to extract interpolants from proofs of unsatisfiable conjunctions $A \wedge B$, we introduce *interpolating sequents* as an extension of the Gentzen-style sequents defined in Section 2. Formulae in interpolating sequents are labeled either with the letter L to indicate that they are derived purely from A, the letter R for formulae derived purely from B, or with *partial interpolants* (PIs) that record the A-contribution to a formula obtained jointly from A and B. Similarly as in [8], the labels L/R will be used to handle analytic rules that operate only on subformulae of the input formulae, while rewriting rules for arithmetic may mix parts of A and B and therefore require partial interpolants (as in [15]).

More formally, if ϕ is a formula and t, t^A are terms, all without free variables, then ϕ and $\lfloor\phi\rfloor_R$ are L/R-labeled formulae and $t \doteq 0\,[t^A \doteq 0]$, $t \doteq 0\,[t^A \not\doteq 0]$, and $t \leq 0\,[t^A \leq 0]$ are formulae labeled with the partial interpolants $t^A \doteq 0$, $t^A \not\doteq 0$, and $t^A \leq 0$, respectively. We formally define interpolating sequents as follows:

Definition 1 If Γ, Δ are finite sets of labeled formulae, and I is an unlabeled formula without free variables, then $\Gamma \vdash \Delta \blacktriangleright I$ is an *interpolating sequent* if

(i) Γ only contains formulae $\lfloor\phi\rfloor_L$, $\lfloor\phi\rfloor_R$, $t \doteq 0\,[t^A \doteq 0]$, or $t \leq 0\,[t^A \leq 0]$, and
(ii) Δ only contains formulae $\lfloor\phi\rfloor_L$, $\lfloor\phi\rfloor_R$, $t \doteq 0\,[t^A \doteq 0]$, or $t \doteq 0\,[t^A \not\doteq 0]$.

Note that formulae in interpolating sequents may not contain free variables.

The semantics of interpolating sequents is defined with the help of projections $\Gamma_L =_{\text{def}} \{\phi \mid \lfloor\phi\rfloor_L \in \Gamma\}$ and $\Gamma_R =_{\text{def}} \{\phi \mid \lfloor\phi\rfloor_R \in \Gamma\}$ that extract the L/R-parts of a set Γ of labeled formulae.

Definition 2 An interpolating sequent $\Gamma \vdash \Delta \blacktriangleright I$ is *valid* if

(i) the sequent $\Gamma_L \vdash I, \Delta_L$ is valid,
(ii) the sequent $\Gamma_R, I \vdash \Delta_R$ is valid, and
(iii) the constants in I occur in both $\Gamma_L \cup \Delta_L$ and $\Gamma_R \cup \Delta_R$.

Note that formulae annotated with PIs are irrelevant for deciding whether an interpolating sequent is valid; this only depends on L/R-formulae. The semantics of PIs is made precise in Section 5; intuitively, a labeled formula $\phi\,[\phi^A]$ in an interpolation problem $A \wedge B$ expresses the implications $A \Rightarrow \phi^A$ and $B \wedge \phi^A \Rightarrow \phi$. This implies that ϕ^A is in fact an interpolant of the conjunction $A \wedge B$ if ϕ is unsatisfiable.

As special cases, $\lfloor A \rfloor_L \vdash \lfloor C \rfloor_R \blacktriangleright I$ reduces to I being an interpolant of the implication $A \Rightarrow C$, while $\lfloor A \rfloor_L, \lfloor B \rfloor_R \vdash \blacktriangleright I$ captures the concept of interpolants I for conjunctions $A \wedge B$ common in formal verification.

Example We illustrate the concept of interpolating sequents with the proof in Fig. 3, which is the interpolating version of the proof in Fig. 1 and will serve as a running example in the whole section. For sake of brevity, we omit the subproofs $\mathcal{A}$ and $\mathcal{B}$. Due to the soundness of the applied calculus (stated in Section 5), the root sequent of the proof is valid, which implies that $I_2 \equiv (-3b + a \leq 0)$ is an interpolant for the unsatisfiable conjunction (1). Note that I_2 is the inequality discussed in Section 3 as a succinct interpolant and intermediate program assertion.

In the remainder of Section 4, we explain the rules of our interpolating calculus given in Figs. 4, 5. As usual in sequent calculi, the rules are applied in the upward direction, starting from a sequent $\Gamma \vdash \Delta \blacktriangleright ?$ with unknown interpolant that is to be proven (the proof root), and applying rules to successively decompose and simplify the sequent until a closure rule becomes applicable. The unknown interpolants of sequents have to be left open while building a proof and can only be filled in once all proof branches are closed.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{\ldots, 2 \leq 0\,[-6b + 2a \leq 0] \vdash \blacktriangleright I_1}\,\text{CLOSE-INEQ}}{\ldots, -2x \leq 0\,[-2x \leq 0], 2x + 2 \leq 0\,[-6b + 2a + 2x \leq 0] \vdash \blacktriangleright I_1}\,\text{FM-ELIM}}{\mathcal{A}\quad\mathcal{B}\quad \ldots, -2b + 2x \leq 0\,[-2b + 2x - 1 \leq 0], 3b - 2x + 1 \leq 0\,[a - 2x \leq 0] \vdash \blacktriangleright I_1}\,\text{FM-ELIM}}{\ldots, -2b + 2x - 1 \leq 0\,[-2b + 2x - 1 \leq 0], 3b - 2x + 1 \leq 0\,[a - 2x \leq 0] \vdash \blacktriangleright I_2}\,\text{STRENGTHEN}}{\ldots, -2x \leq 0\,[-2x \leq 0], -2b + 2x - 1 \leq 0\,[-2b + 2x - 1 \leq 0],\ c - 3b - 1 \doteq 0\,[0 \doteq 0], c - 2x \leq 0\,[a - 2x \leq 0] \vdash \blacktriangleright I_2}\,\text{RED-LEFT}}{\ldots, a - 2x \doteq 0\,[a - 2x \doteq 0], -2b + a - 1 \leq 0\,[-2b + a - 1 \leq 0],\ -a \leq 0\,[-a \leq 0], c - 3b - 1 \doteq 0\,[0 \doteq 0], c - a \leq 0\,[0 \leq 0] \vdash \blacktriangleright I_2}\,\text{RED-LEFT}^+}{\lfloor a - 2x \doteq 0 \rfloor_L, \lfloor -a \leq 0 \rfloor_L, \ldots, \lfloor -2b + a - 1 \leq 0 \rfloor_L,\ \lfloor c - 3b - 1 \doteq 0 \rfloor_R, \lfloor c - a \leq 0 \rfloor_R \vdash \blacktriangleright I_2}\,\text{IPI}^+}{\lfloor a - 2x \doteq 0 \wedge -a \leq 0 \wedge 2b - a \leq 0 \wedge -2b + a - 1 \leq 0 \rfloor_L,\ \lfloor c - 3b - 1 \doteq 0 \wedge c - a \leq 0 \rfloor_R \vdash \blacktriangleright I_2}\,\text{AND-LEFT}^+
$$

Fig. 3 The interpolating version of Fig. 1. The initial interpolant generated by CLOSE-INEQ is $I_1 = (-6b + 2a \leq 0) \equiv (-3b + a \leq 0)$, which is by STRENGTHEN combined with the interpolants *false* and ϕ from the subproofs $\mathcal{A}$ and $\mathcal{B}$ to form the final interpolant $I_2 = (I_1 \vee (\mathit{false} \wedge \phi)) \equiv I_1$

$$\frac{\Gamma, \lfloor\phi\rfloor_L \vdash \Delta \blacktriangleright I \quad \Gamma, \lfloor\psi\rfloor_L \vdash \Delta \blacktriangleright J}{\Gamma, \lfloor\phi \vee \psi\rfloor_L \vdash \Delta \blacktriangleright I \vee J}\ \text{OR-LEFT-L} \qquad \frac{\Gamma, \lfloor\phi\rfloor_R \vdash \Delta \blacktriangleright I \quad \Gamma, \lfloor\psi\rfloor_R \vdash \Delta \blacktriangleright J}{\Gamma, \lfloor\phi \vee \psi\rfloor_R \vdash \Delta \blacktriangleright I \wedge J}\ \text{OR-LEFT-R}$$

$$\frac{\Gamma \vdash \lfloor\phi\rfloor_L, \Delta \blacktriangleright I \quad \Gamma \vdash \lfloor\psi\rfloor_L, \Delta \blacktriangleright J}{\Gamma \vdash \lfloor\phi \wedge \psi\rfloor_L, \Delta \blacktriangleright I \vee J}\ \text{AND-RIGHT-L} \qquad \frac{\Gamma \vdash \lfloor\phi\rfloor_R, \Delta \blacktriangleright I \quad \Gamma \vdash \lfloor\psi\rfloor_R, \Delta \blacktriangleright J}{\Gamma \vdash \lfloor\phi \wedge \psi\rfloor_R, \Delta \blacktriangleright I \wedge J}\ \text{AND-RIGHT-R}$$

$$\frac{\Gamma, \lfloor\phi\rfloor_D, \lfloor\psi\rfloor_D \vdash \Delta \blacktriangleright I}{\Gamma, \lfloor\phi \wedge \psi\rfloor_D \vdash \Delta \blacktriangleright I}\ \text{AND-LEFT} \qquad \frac{\Gamma \vdash \lfloor\phi\rfloor_D, \lfloor\psi\rfloor_D, \Delta \blacktriangleright I}{\Gamma \vdash \lfloor\phi \vee \psi\rfloor_D, \Delta \blacktriangleright I}\ \text{OR-RIGHT}$$

$$\frac{\Gamma \vdash \lfloor\phi\rfloor_D, \Delta \blacktriangleright I}{\Gamma, \lfloor\neg\phi\rfloor_D \vdash \Delta \blacktriangleright I}\ \text{NOT-LEFT} \qquad \frac{\Gamma, \lfloor\phi\rfloor_D \vdash \Delta \blacktriangleright I}{\Gamma \vdash \lfloor\neg\phi\rfloor_D, \Delta \blacktriangleright I}\ \text{NOT-RIGHT}$$

$$\frac{\Gamma, \lfloor[x/c]\phi\rfloor_D \vdash \Delta \blacktriangleright I}{\Gamma, \lfloor\exists x.\phi\rfloor_D \vdash \Delta \blacktriangleright I}\ \text{EX-LEFT} \qquad \frac{\Gamma \vdash \lfloor[x/c]\phi\rfloor_D, \Delta \blacktriangleright I}{\Gamma \vdash \lfloor\forall x.\phi\rfloor_D, \Delta \blacktriangleright I}\ \text{ALL-RIGHT}$$

$$\frac{\Gamma, t \circ 0\,[t \circ 0], \lfloor t \circ 0\rfloor_L \vdash \Delta \blacktriangleright I}{\Gamma, \lfloor t \circ 0\rfloor_L \vdash \Delta \blacktriangleright I}\ \text{IPI-LEFT-L} \qquad \frac{\Gamma \vdash t \doteq 0\,[t \doteq 0], \lfloor t \doteq 0\rfloor_L, \Delta \blacktriangleright I}{\Gamma \vdash \lfloor t \doteq 0\rfloor_L, \Delta \blacktriangleright I}\ \text{IPI-RIGHT-L}$$

$$\frac{\Gamma, t \circ 0\,[0 \circ 0], \lfloor t \circ 0\rfloor_R \vdash \Delta \blacktriangleright I}{\Gamma, \lfloor t \circ 0\rfloor_R \vdash \Delta \blacktriangleright I}\ \text{IPI-LEFT-R} \qquad \frac{\Gamma \vdash t \doteq 0\,[0 \not\doteq 0], \lfloor t \doteq 0\rfloor_R, \Delta \blacktriangleright I}{\Gamma \vdash \lfloor t \doteq 0\rfloor_R, \Delta \blacktriangleright I}\ \text{IPI-RIGHT-R}$$

$$\frac{*}{\Gamma, t \doteq 0\,[t^A \doteq 0] \vdash \Delta \blacktriangleright \exists_{LA}\, t^A \doteq 0}\ \text{CLOSE-EQ-LEFT} \qquad (t \doteq 0 \text{ unsat.})$$

$$\frac{*}{\Gamma, \alpha \leq 0\,[t^A \leq 0] \vdash \Delta \blacktriangleright t^A \leq 0}\ \text{CLOSE-INEQ} \qquad (\alpha > 0)$$

$$\frac{*}{\Gamma \vdash 0 \doteq 0\,[t^A \doteq 0], \Delta \blacktriangleright t^A \not\doteq 0}\ \text{CLOSE-EQ-RIGHT}$$

$$\frac{*}{\Gamma \vdash 0 \doteq 0\,[t^A \not\doteq 0], \Delta \blacktriangleright t^A \doteq 0}\ \text{CLOSE-NEQ-RIGHT}$$

Fig. 4 Propositional, Skolemization, initialization, and closure rules. In the propositional rules, $D \in \{L, R\}$. In the rules IPI-LEFT-L/R, $\circ \in \{\doteq, \leq\}$ denotes a relation symbol. In the rule CLOSE-EQ-LEFT, $\exists_{LA}$ denotes existential quantification $\exists c_1, \ldots, c_n.$, over constants c_i that occur in Γ_L, Δ_L but not in Γ_R, Δ_R. An equality $t^A \doteq 0$ is unsatisfiable if and only if it is of the form $\alpha_1 d_1 + \cdots + \alpha_n d_n + \alpha_0 \doteq 0$ and $\gcd(\alpha_1, \ldots, \alpha_n) \nmid \alpha_0$ (with the convention $\gcd() = 0$)

4.1 Propositional, Initialization, and Closure Rules

To construct a proof for an interpolation problem $A \wedge B$, we start with a sequent $\lfloor A\rfloor_L, \lfloor B\rfloor_R \vdash \blacktriangleright ?$ that only contains L/R-labeled formulae and then apply propositional rules and Skolemization rules to decompose A and B (the applications of rule AND-LEFT in Fig. 3). Propositional rules are shown in the topmost part of Fig. 4. Splitting over L-disjunctions in the antecedent (OR-LEFT-L) requires forming the disjunction of the interpolants derived in the subproofs. Analogously, R-disjunctions

$$\frac{\Gamma, t \doteq 0\,[t^A \doteq 0], s + \alpha \cdot t \circ 0\,[s^A + \alpha \cdot t^A \circ 0] \vdash \Delta \blacktriangleright I}{\Gamma, t \doteq 0\,[t^A \doteq 0], s \circ 0\,[s^A \circ 0] \vdash \Delta \blacktriangleright I}\ \textsc{red-left}$$

$$\frac{\Gamma, t \doteq 0\,[t^A \doteq 0] \vdash s + \alpha \cdot t \doteq 0\,[s^A + \alpha \cdot t^A \circ 0], \Delta \blacktriangleright I}{\Gamma, t \doteq 0\,[t^A \doteq 0] \vdash s \doteq 0\,[s^A \circ 0], \Delta \blacktriangleright I}\ \textsc{red-right}$$

$$\frac{\Gamma, \lfloor u - c \doteq 0 \rfloor_L \vdash \Delta \blacktriangleright I}{\Gamma \vdash \Delta \blacktriangleright I}\ \textsc{col-red-l} \qquad \frac{\Gamma, \alpha \cdot t \circ 0\,[\alpha \cdot t^A \circ 0] \vdash \Delta \blacktriangleright I}{\Gamma, t \circ 0\,[t^A \circ 0] \vdash \Delta \blacktriangleright I}\ \textsc{mul-left}$$

$$\frac{\Gamma, \lfloor u - c \doteq 0 \rfloor_R \vdash \Delta \blacktriangleright I}{\Gamma \vdash \Delta \blacktriangleright I}\ \textsc{col-red-r} \qquad \frac{\Gamma \vdash \alpha \cdot t \doteq 0\,[\alpha \cdot t^A \circ 0], \Delta \blacktriangleright I}{\Gamma \vdash t \doteq 0\,[t^A \circ 0], \Delta \blacktriangleright I}\ \textsc{mul-right}$$

$$\frac{\Gamma, \lfloor \exists x.\, \alpha x + t \doteq 0 \rfloor_D \vdash \Delta \blacktriangleright I}{\Gamma, \lfloor \alpha \mid t \rfloor_D \vdash \Delta \blacktriangleright I}\ \textsc{div-left}$$

$$\frac{\Gamma, \lfloor \exists x.\, (1 - x \leq 0 \wedge x - \alpha + 1 \leq 0 \wedge (\alpha \mid t + x)) \rfloor_D \vdash \Delta \blacktriangleright I}{\Gamma \vdash \lfloor \alpha \mid t \rfloor_D, \Delta \blacktriangleright I}\ \textsc{div-right}$$

$$\frac{\Gamma, s \leq 0\,[s^A \leq 0], t \leq 0\,[t^A \leq 0], \alpha s + \beta t \leq 0\,[\alpha s^A + \beta t^A \leq 0] \vdash \Delta \blacktriangleright I}{\Gamma, s \leq 0\,[s^A \leq 0], t \leq 0\,[t^A \leq 0] \vdash \Delta \blacktriangleright I}\ \textsc{fm-elim}$$

$$\frac{\begin{array}{c}\Gamma, t \doteq 0\,[t^A \doteq 0] \vdash \Delta \blacktriangleright E \\ \Gamma, t + 1 \leq 0\,[t^A \leq 0] \vdash \Delta \blacktriangleright I^0 \\ \Gamma, t + 1 \leq 0\,[t^A + 1 \leq 0] \vdash \Delta \blacktriangleright I^1\end{array}}{\Gamma, t \leq 0\,[t^A \leq 0] \vdash \Delta \blacktriangleright I^1 \vee (E \wedge I^0)}\ \textsc{strengthen}$$

$$\frac{\Gamma, t + 1 \leq 0\,[t^A + 1 \leq 0] \vdash \Delta \blacktriangleright I \qquad \Gamma, -t + 1 \leq 0\,[-t^A + 1 \leq 0] \vdash \Delta \blacktriangleright J}{\Gamma \vdash t \doteq 0\,[t^A \doteq 0], \Delta \blacktriangleright I \vee J}\ \textsc{split-eq}$$

$$\frac{\Gamma, t + 1 \leq 0\,[t^A \leq 0] \vdash \Delta \blacktriangleright I \qquad \Gamma, -t + 1 \leq 0\,[-t^A \leq 0] \vdash \Delta \blacktriangleright J}{\Gamma \vdash t \doteq 0\,[t^A \not\doteq 0], \Delta \blacktriangleright I \wedge J}\ \textsc{split-neq}$$

Fig. 5 Rules for equality/divisibility and inequality constraints. In RED-LEFT and MUL-LEFT, $\circ \in \{\doteq, \leq\}$, while in RED-RIGHT and MUL-RIGHT, $\circ \in \{\doteq, \not\doteq\}$. In COL-RED-L and COL-RED-R, c is a constant not occurring in the conclusion nor in u. The term u in COL-RED-L must only contain constants from $\Gamma_L \cup \Delta_L$, while u in COL-RED-R must only contain constants from $\Gamma_R \cup \Delta_R$. In MUL-LEFT and MUL-RIGHT, $\alpha > 0$ is a positive literal. In DIV-LEFT and DIV-RIGHT, $D \in \{L, R\}$, x is a fresh variable, and $\alpha > 0$. In FM-ELIM, $\alpha > 0$ and $\beta > 0$ are positive integers

yield conjunctive interpolants. All propositional rules propagate the L/R-label of formulae to their subformulae, unchanged. For brevity, we have omitted rules to move inequalities from the succedent to the antecedent.

Once the decomposition of formulae results in arithmetic literals, the *initialization* rules in the middle part of Fig. 4 are used to turn L/R-formulae into formulae with PIs, to prepare them for later rewriting (the applications IPI in Fig. 3). Generally, PIs for L-literals are chosen to be the literals themselves, while empty PIs are introduced for R-literals: the intuition is that L-formulae are fully contributed by A, while R-formulae do not contain any A-contribution at all.

We observe that the IPI rules do not remove the L/R-formula to which they are applied (the formula occurs both in the conclusion and in the premise). The reason is that L/R-formulae in sequents, besides their logical meaning, track the vocabulary of

symbols occurring in the input formulae A, B; the vocabulary is used in condition (iii) of the definition of valid interpolating sequents, but also in the closure rules discussed next. To achieve completeness, it is never necessary to apply IPI rules twice on a proof branch to the same L/R-formula.

Finally, once rewriting (discussed in Section 4.2) has produced an unsatisfiable literal in an antecedent (or a valid literal in a succedent), a closure rule can be used to close the proof branch and to derive an interpolant from the PI of the unsatisfiable literal (the application CLOSE-INEQ in Fig. 3). Closure rules are given in the lower part of Fig. 4. Because PIs can still contain local symbols that occur only in $\Gamma_L \cup \Delta_L$ (and are not allowed in interpolants), it may be necessary to introduce existential quantifiers at this point. We note, however, that quantifiers in quantified literals can be eliminated in polynomial time; e.g., $\exists c_1, \ldots, c_n.\ \alpha_1 c_1 + \cdots + \alpha_n c_n + t \doteq 0$ is equivalent to the divisibility judgment $\gcd(\alpha_1, \ldots, \alpha_n) \mid t$.

4.2 Rewriting Rules for Equality, Inequality and Divisibility

Our arithmetic rewriting rules, shown in Fig. 5, form a calculus to solve systems of equalities by means of Gaussian elimination and Euclid's algorithm (the upper part of Fig. 5), as well as a calculus for systems of inequalities that enables us to introduce linear combinations of inequalities, and to strengthen inequalities by means of cuts (the lower part of Fig. 5). Decision procedures for QFPA in terms of the corresponding non-interpolating rules have been introduced in [18, 20] and directly carry over to the interpolating case. In particular, the rules can be used to simulate various procedures for linear integer arithmetic, e.g.:

- *The Omega Test* [17], which is a quantifier elimination procedure for Presburger arithmetic that combines the Fourier–Motzkin method with a splitting rule (the Omega rule) to achieve completeness over the integers. In the quantifier-free case (corresponding to the existential fragment of PA), our calculus can implement the Fourier–Motzkin method through the rule FM-ELIM, and can simulate the Omega rule straightforwardly with the help of the rule STRENGTHEN (see [19] for a discussion).
- *Linear Programming (LP) with Gomory cuts:* given a set of (in)equalities that is unsatisfiable over rational numbers, LP methods are able to provide a *witness* in terms of a (non-negative) linear combination of the inequalities that certifies unsatisfiability (the existence of such witnesses is guaranteed by Farkas' lemma [22]). Once such a witness has been derived, the rule FM-ELIM of our calculus can be used to derive a contradictory inequality, so that LP methods can guide our calculus. The simulation of rules specific to integer linear programming, including Gomory cuts, is discussed in Section 6.2.

The rules RED-LEFT/RIGHT rewrite (in)equalities with equalities in the antecedent; in both cases, PIs are simply propagated along with the literals (RED-LEFT is applied repeatedly in Fig. 3). The RED rules alone are not sufficient for transforming a system of integer equations into a solved form; they are therefore complemented with COL-RED-L/R to introduce fresh constants defined in terms of existing constants (the rules resemble *col*umn *red*uctions when encoding systems of equalities as matrices). In combination, RED and COL-RED are able to simulate the equality elimination

procedure in [17], as well as standard procedures to transform sets of equalities (or matrices) to Hermite and Smith normalform [10, 11]. Because COL-RED-L/R only introduce local L/R-constants, it is guaranteed that the new constants do not occur in interpolants.

The calculi in [18, 20] include a rule SIMP′ that is responsible for rounding inequalities $\alpha t + \beta \leq 0$ to $\alpha t + \alpha \lceil \frac{\beta}{\alpha} \rceil \leq 0$, as well as for eliminating common factors from coefficients of non-constant terms, simplifying equalities $\alpha t \doteq 0$ to $t \doteq 0$ or inequalities $\alpha t \leq 0$ to $t \leq 0$ (provided $\alpha > 0$). The rule SIMP′ corresponds to a variety of more elementary rules in our interpolating calculus. Rounding of inequalities is handled by the STRENGTHEN rule discussed below and in Section 6. In contrast, elimination of common factors in coefficients is not always possible in the presence of PIs: for instance, unlike in [18, 20], the equality $2x \doteq 0\,[a \doteq 0]$ cannot be simplified to the form $x \doteq 0\,[t^A \doteq 0]$, because the factor 2 does not occur in the PI. This means that terms αx cannot be rewritten to 0 with the help of $2x \doteq 0$ if α is odd. Following the principle of pseudo-division, we therefore introduce the rules MUL-LEFT/RIGHT to multiply terms with positive integers prior to rewriting.

Similar to rewriting with equalities, inequalities can be added up with the help of the rule FM-ELIM. The STRENGTHEN rule is introduced to achieve completeness over the integers and splits inequalities $t \leq 0$ into the cases $t \doteq 0$ and $t + 1 \leq 0$ (Fig. 3 shows applications of FM-ELIM and STRENGTHEN). Compared to the calculi in [18, 20], the use of STRENGTHEN in our interpolating calculus is threefold: (i) STRENGTHEN can simulate rules such as OMEGA-ELIM [20] or Gomory cuts, (ii) as shown in Fig. 3, repeated application of STRENGTHEN can be used to round inequalities $\alpha t + \beta \leq 0$ to $\alpha t + \alpha \lceil \frac{\beta}{\alpha} \rceil \leq 0$ (which is done by SIMP′ in [18, 20]), and (iii) STRENGTHEN can simulate the law of anti-symmetry that is implemented by the rule ANTI-SYMM′ in [18, 20]. As STRENGTHEN is the most central rule in our calculus, we provide a detailed discussion in Section 6.

Finally, when reasoning about formulae that contain both equalities and inequalities, it can be necessary to split equalities $t \doteq 0$ into two inequalities $t \leq 0$ and $-t \leq 0$; this is done by the rules SPLIT-EQ and SPLIT-NEQ.

5 Properties of the Calculus

In this section we discuss properties of our calculus that are essential for turning it from a mere set of proof rules into a practical interpolation algorithm. We establish that *repartitioning* the set of input formulae into new L and R parts is largely orthogonal to the interpolating proof. That is, given an interpolating proof for a particular partitioning, an interpolant for a new partitioning can be obtained by suitably adjusting the occurrence of labels across the proof. This property of the calculus has two important consequences. First, our calculus has the *chain interpolation* property, which is essential in model checking based on *lazy abstraction with interpolants* [16] (Section 5.2). Second, the relative insensitivity of the proof procedure to repartitioning can be used to show the completeness of the interpolating calculus. Together with the soundness result that we present below, our calculus is therefore suitable to be implemented in an interpolating decision procedure for quantifier-free Presburger arithmetic.

5.1 Soundness of the Calculus

The most important property of any calculus is soundness: the existence of a proof for a formula or sequent should imply that the formula is indeed valid. Since the correctness of interpolants is built into our definition of valid (interpolating) sequents, soundness also means that a calculus only generates valid interpolants: whenever a sequent $\lfloor A \rfloor_L \vdash \lfloor C \rfloor_R \blacktriangleright I$ is derived, the implications $A \Rightarrow I$ and $I \Rightarrow C$ are valid, and all constants in I occur in both A and C. More generally:

Theorem 1 (Soundness) *If an interpolating sequent* $\Gamma \vdash \Delta \blacktriangleright I$ *without any PIs is provable in the calculus, then it is valid. This implies, in particular, that the sequent* $\Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_R$ *is valid.*

To prove this theorem, we first need to define the semantics of PIs, since—although the sequent $\Gamma \vdash \Delta \blacktriangleright I$ in the theorem does not contain any PIs—they are normally introduced in the course of a proof. We say that a PI (as in Definition 1) occurring in a sequent is *correct* if the sequents (i) and (ii) given in Fig. 6 are valid, t^A only contains constants that occur in $\Gamma_L \cup \Delta_L$, and $t - t^A$ only contains constants that occur in $\Gamma_R \cup \Delta_R$.

Theorem 1 is then proven in two steps:

1. In Section 5.1.1, we show that all PIs in a closed proof are correct by induction on the distance of a sequent from the root of the proof: assuming that all PIs in the conclusion of a rule application are correct, we prove that the PIs in the rule premises are correct.
2. In Section 5.1.2, we show the validity of all sequents in a closed proof by induction on the size of sub-proofs: assuming that all premises of a rule are valid, we prove that the conclusion is valid, too.

In order to show the two properties, we annotate some of the rules by introducing further *auxiliary formulae* in the premises (Fig. 7). The calculus can also be proven sound directly with the original rules, which requires, however, more intricate inductive properties. Auxiliary formulae are not needed for completeness (it is never necessary to apply any rules to the formulae), and are therefore not used in an implementation. Similarly, soundness of the rules with auxiliary formulae directly implies the soundness of the rules without auxiliary formulae, because removing formulae from the premises could only make fewer sequents provable.

Partial interpolant annotation	Sequent (i)	Sequent (ii)
$\Gamma, t \doteq 0[t^A \doteq 0] \vdash \Delta$	$\Gamma_L \vdash t^A \doteq 0, \Delta_L$	$\Gamma_R \vdash t - t^A \doteq 0, \Delta_R$
$\Gamma, t \leq 0[t^A \leq 0] \vdash \Delta$	$\Gamma_L \vdash t^A \leq 0, \Delta_L$	$\Gamma_R \vdash t - t^A \leq 0, \Delta_R$
$\Gamma \vdash t \doteq 0[t^A \doteq 0], \Delta$	$\Gamma_L, t^A \doteq 0 \vdash \Delta_L$	$\Gamma_R \vdash t - t^A \doteq 0, \Delta_R$
$\Gamma \vdash t \doteq 0[t^A \not\doteq 0], \Delta$	$\Gamma_L \vdash t^A \doteq 0, \Delta_L$	$\Gamma_R, t - t^A \doteq 0 \vdash \Delta_R$

Fig. 6 Sequents with partial interpolants and correctness conditions (i) and (ii)

$$\frac{\Gamma, t \doteq 0\,[t^A \doteq 0], \lfloor t^A \doteq 0 \rfloor^*_L, \lfloor t - t^A \doteq 0 \rfloor^*_R \vdash \Delta \blacktriangleright E \quad \Gamma, t+1 \leq 0\,[t^A \leq 0], \lfloor t - t^A + 1 \leq 0 \rfloor^*_R \vdash \Delta \blacktriangleright I^0 \quad \Gamma, t+1 \leq 0\,[t^A + 1 \leq 0], \lfloor t^A + 1 \leq 0 \rfloor^*_L \vdash \Delta \blacktriangleright I^1}{\Gamma, t \leq 0\,[t^A \leq 0] \vdash \Delta \blacktriangleright I^1 \vee (E \wedge I^0)}\ \textsc{strengthen}^*$$

$$\frac{\Gamma, t+1 \leq 0\,[t^A + 1 \leq 0], \lfloor t^A \leq 0 \rfloor^*_L \vdash \Delta \blacktriangleright I \quad \Gamma, -t+1 \leq 0\,[-t^A + 1 \leq 0], \lfloor -t^A \leq 0 \rfloor^*_L \vdash \Delta \blacktriangleright J}{\Gamma \vdash t \doteq 0\,[t^A \doteq 0], \Delta \blacktriangleright I \vee J}\ \textsc{split-eq}^*$$

$$\frac{\Gamma, t+1 \leq 0\,[t^A \leq 0], \lfloor t - t^A \leq 0 \rfloor^*_R \vdash \Delta \blacktriangleright I \quad \Gamma, -t+1 \leq 0\,[-t^A \leq 0], \lfloor -t + t^A \leq 0 \rfloor^*_R \vdash \Delta \blacktriangleright J}{\Gamma \vdash t \doteq 0\,[t^A \not\doteq 0], \Delta \blacktriangleright I \wedge J}\ \textsc{split-neq}^*$$

Fig. 7 Rules modified for proving soundness (Theorem 1) by adding auxiliary formulae. Auxiliary formulae are written in the form $\lfloor \phi \rfloor^*_L$ and $\lfloor \phi \rfloor^*_R$

5.1.1 Correctness of Partial Interpolants in a Proof

We claim that each rule of the calculus preserves the correctness of PIs. We only show two examples here:

- RED-LEFT, for $\circ$ being $\leq$: suppose all partial interpolants of the conclusion are valid. This means that $\Gamma_L \vdash t^A \doteq 0, \Delta_L$ and $\Gamma_L \vdash s^A \leq 0, \Delta_L$ are valid, from which we can conclude that also $\Gamma_L \vdash s^A + \alpha \cdot t^A \leq 0, \Delta_L$ is valid. Furthermore, $\Gamma_R \vdash t - t^A \doteq 0, \Delta_R$ and $\Gamma_R \vdash s - s^A \leq 0, \Delta_R$ are valid, which implies that $\Gamma_R \vdash s - s^A + \alpha \cdot (t - t^A) \leq 0, \Delta_R$ and therefore also the sequent $\Gamma_R \vdash s + \alpha \cdot t - (s^A + \alpha \cdot t^A) \leq 0, \Delta_R$ are valid. Finally, all constants of the term $s^A + \alpha \cdot t^A$ also occur in s^A or t^A, and all constants of $s + \alpha \cdot t - (s^A + \alpha \cdot t^A)$ also in $s - s^A$ or $t - t^A$, so that also the vocabulary conditions are satisfied.
- STRENGTHEN*: by assumption, the annotations of the conclusion are correct, which implies that the sequents $\Gamma_L \vdash t^A \leq 0, \Delta_L$ and $\Gamma_R \vdash t - t^A \leq 0, \Delta_R$ are valid. The correctness of the new partial interpolants is then directly guaranteed by the starred formulae in the premises.

5.1.2 Correctness of Sequents in a Proof

We only show one example, the rule STRENGTHEN*. Suppose each of the premises of the rule is a valid interpolating sequent, which by definition means that the following (ordinary) sequents are valid:

$$\Gamma_L, t^A \doteq 0 \vdash E, \Delta_L \quad (3) \qquad \Gamma_R, t - t^A + 1 \leq 0, I^0 \vdash \Delta_R \quad (6)$$

$$\Gamma_R, t - t^A \doteq 0, E \vdash \Delta_R \quad (4) \qquad \Gamma_L, t^A + 1 \leq 0 \vdash I^1, \Delta_L \quad (7)$$

$$\Gamma_L \vdash I^0, \Delta_L \quad (5) \qquad \Gamma_R, I^1 \vdash \Delta_R \quad (8)$$

Note that the arithmetic atoms in the sequents (like $t^A \doteq 0$ in (3)) stem from the L^*/R^*-formulae in the premises. Furthermore, because $t^A \leq 0$ is a correct partial interpolant, we know that the following sequents are valid:

$$\Gamma_L \vdash t^A \leq 0, \Delta_L \qquad (9) \qquad\qquad \Gamma_R \vdash t - t^A \leq 0, \Delta_R \qquad (10)$$

The validity of (3)–(10) implies that also the sequents $\Gamma_L \vdash I^1 \vee (E \wedge I^0), \Delta_L$ and $\Gamma_R, I^1 \vee (E \wedge I^0) \vdash \Delta_R$ are valid. For instance, the former sequent can be deduced using a normal Gentzen-style calculus (as introduced in Section 2):

$$\dfrac{\dfrac{\dfrac{}{\Gamma_L, t^A \leq 0, t^A \doteq 0 \vdash I^1, E, \Delta_L}\,(3) \qquad \dfrac{}{\Gamma_L, t^A \leq 0 \vdash t^A \doteq 0, I^1, E, \Delta_L}\,(7)}{\mathcal{A}}\,(*)\ \text{CUT}'}$$

$$\dfrac{\dfrac{\dfrac{\mathcal{A}}{\Gamma_L, t^A \leq 0 \vdash I^1, E, \Delta_L} \qquad \dfrac{}{\Gamma_L \vdash t^A \leq 0, I^1, E, \Delta_L}\,(9)}{\Gamma_L \vdash I^1, E, \Delta_L}\,\text{CUT}' \qquad \dfrac{}{\Gamma_L \vdash I^1, I^0, \Delta_L}\,(5)}{\dfrac{\Gamma_L \vdash I^1, E \wedge I^0, \Delta_L}{\Gamma_L \vdash I^1 \vee (E \wedge I^0), \Delta_L}\,\text{OR-RIGHT}'}\,\text{AND-RIGHT}'$$

In $(*)$, we make use of the fact that $(t^A \leq 0 \wedge t^A \not\doteq 0) \equiv t^A + 1 \leq 0$ over integers.

For the vocabulary condition, note that a constant is an L/R-symbol of the conclusion iff it is an L/R-symbol of each of the premises. This is because $t \leq 0\,[t^A \leq 0]$ is annotated with a correct partial interpolant, which implies that all constants in t are L-constants, and all constants in $t - t^A$ are R-constants. Therefore, the L^*/R^* formulae introduced in the premises do not change vocabularies. Because each of the formulae E, I^0, I^1 only contains common L/R-constants, so does $I^1 \vee (E \wedge I^0)$.

Concluding, this means that the sequent $\Gamma \vdash \Delta \blacktriangleright I^1 \vee (E \wedge I^0)$ is valid, and therefore also $\Gamma, t \leq 0\,[t^A \leq 0] \vdash \Delta \blacktriangleright I^1 \vee (E \wedge I^0)$.

5.2 Chain Interpolation and Completeness

In software model checking, it is common to use a slightly generalized version of the interpolation theorem, guaranteeing not only the existence of single interpolants, but of entire *interpolant chains* (see, e.g., [16]). Given an unsatisfiable conjunction $T_1 \wedge \cdots \wedge T_n$ (say corresponding to an infeasible path in a program), an interpolant chain is a sequence $I_0, I_1, \ldots, I_n$ of formulae such that

(i) $I_0 = \mathit{true}$, $I_n = \mathit{false}$,
(ii) for all $i \in \{1, \ldots, n\}$, the implication $I_{i-1} \wedge T_i \Rightarrow I_i$ holds, and
(iii) for all $i \in \{0, \ldots, n\}$, the non-logical symbols (constants, predicates, etc.) in I_i occur in both $T_1 \wedge \cdots \wedge T_i$ and $T_{i+1} \wedge \cdots \wedge T_n$.

Interpolant chains $I_0, I_1, \ldots, I_n$ can be derived by $n - 1$ applications of the standard interpolation theorem. Since this would entail the construction of $n - 1$ proofs and would thus represent a significant overhead, however, a more attractive and more common approach is to extract multiple interpolants from a single unsatisfiability proof for $T_1 \wedge \cdots \wedge T_n$, by considering different partitions of the conjuncts, i.e., different ways to label the formulae in the proof.

In the context of our interpolating calculus for QFPA, we introduce chain interpolation in a constructive manner using a function S that transforms an interpolating proof of a sequent $\lfloor T_1 \rfloor_L, \ldots, \lfloor T_k \rfloor_L, \lfloor T_{k+1} \rfloor_R, \lfloor T_{k+2} \rfloor_R, \ldots, \lfloor T_n \rfloor_R \vdash \emptyset \blacktriangleright I$ into a proof of the sequent $\lfloor T_1 \rfloor_L, \ldots, \lfloor T_k \rfloor_L, \lfloor T_{k+1} \rfloor_L, \lfloor T_{k+2} \rfloor_R, \ldots, \lfloor T_n \rfloor_R \vdash \emptyset \blacktriangleright I'$. This requires recursively changing the labels of formulae in the proof. As an inductive property of the relabeling function S, we then show that the implication $I \wedge T_{k+1} \Rightarrow I'$ holds. Chains of interpolants can be extracted from a proof by n applications of S. Since S has complexity polynomial in the size of the processed proof (updating labels does not change the topology of a proof), this yields a practical method for generating interpolant chains.[1]

5.2.1 Definition of the Relabeling Function S

For sake of brevity, we give a formal definition of S only for the case of L/R-relabeled formulae, primarily covering propositional and first-order rules. A complete definition of relabeling, which also includes the treatment of arithmetic rules and PIs, can be found in [2]. As a notational convention, given a set Γ of L/R-labeled and a set Γ^m of unlabeled formulae, we will write $\Gamma \swarrow \Gamma^m$ for the unique set of labeled formulae satisfying the following equations:

$$(\Gamma \swarrow \Gamma^m)_L = \Gamma_L \cup (\Gamma_R \cap \Gamma^m), \qquad (\Gamma \swarrow \Gamma^m)_R = \Gamma_R \setminus \Gamma^m.$$

Intuitively, $\swarrow$ changes the label of the formulae Γ^m from R to L, for instance $\{\lfloor \phi \rfloor_L, \lfloor \psi \rfloor_R\} \swarrow \{\psi\} = \{\lfloor \phi \rfloor_L, \lfloor \psi \rfloor_L\}$.

Formally, the relabeling function S has three arguments:

- an interpolating proof $\mathcal{P}$ in the calculus presented in this paper. As a convention, we denote the root of $\mathcal{P}$ by $\Gamma^r \vdash \Delta^r \blacktriangleright I^r$;
- an ordinary sequent $\Gamma^m \vdash \Delta^m$ such that $\Gamma^m \subseteq \Gamma^r_R$ and $\Delta^m \subseteq \Delta^r_R$, specifying the formulae in $\mathcal{P}$ to be relabeled; and
- a pair $\Gamma^a \vdash \Delta^a$ of labeled formulae, specifying formulae to be recursively added to all sequents of $\mathcal{P}$.

The result of an application $S(\mathcal{P}, \Gamma^m \vdash \Delta^m, \Gamma^a \vdash \Delta^a)$ is a proof $\mathcal{P}'$ with root $\Gamma^r \swarrow \Gamma^m, \Gamma^a \vdash \Delta^r \swarrow \Delta^m, \Delta^a \blacktriangleright I'$. The function S is defined by a complete case analysis over the rules that can be applied at the root of $\mathcal{P}$. Due to space constraints, we show this definition only for the OR-LEFT-R, the other rules are handled similarly. For OR-LEFT-R, $\mathcal{P}$ has the shape:

$$\frac{\mathcal{Q}_1 \qquad \mathcal{Q}_2}{\Gamma, \lfloor \phi \vee \psi \rfloor_R \vdash \Delta \blacktriangleright I \wedge J} \text{ OR-LEFT-R}$$

with $\Gamma^r = \Gamma \cup \{\lfloor \phi \vee \psi \rfloor_R\}$, $\Delta^r = \Delta$, and $I^r = I \wedge J$. The relabeled proof $\mathcal{P}' = S(\mathcal{P}, \Gamma^m \vdash \Delta^m, \Gamma^a \vdash \Delta^a)$ is defined by case analysis; we only show the case $\phi \vee \psi \in \Gamma^m$, as $\phi \vee \psi \notin \Gamma^m$ is handled similarly. The assumption $\phi \vee \psi \in \Gamma^m$ implies

[1] However, some of the optimizations discussed in Section 6 can have the side effect of increasing the complexity of S to exponential.

that the formula $\lfloor \phi \vee \psi \rfloor_R$ of the root is relabeled to $\lfloor \phi \vee \psi \rfloor_L$. Due to the equation $(\Gamma \cup \{\lfloor \phi \vee \psi \rfloor_R\}) \swarrow \Gamma^m = (\Gamma \swarrow \Gamma^m) \cup \{\lfloor \phi \vee \psi \rfloor_L\}$, the proof $\mathcal{P}'$ has the shape

$$\frac{\mathcal{Q}'_1 \qquad \mathcal{Q}'_2}{\Gamma \swarrow \Gamma^m, \Gamma^a, \lfloor \phi \vee \psi \rfloor_L \ \vdash\ \Delta \swarrow \Delta^m, \Delta^a \ \blacktriangleright\ I' \vee J'} \text{ OR-LEFT-L}$$

The direct sub-proofs $\mathcal{Q}'_1, \mathcal{Q}'_2$ of $\mathcal{P}'$ are derived from $\mathcal{Q}_1, \mathcal{Q}_2$ by recursive application of S. Concentrating on $\mathcal{Q}_1$ (the case $\mathcal{Q}_2$ is similar), we choose

$$\Gamma_1^m = (\Gamma^m \cap \Gamma_R) \cup \{\phi\}, \quad \Delta_1^m = \Delta^m, \quad \Gamma_1^a = \Gamma^a \cup \big(\{\lfloor \phi \rfloor_R\} \cap (\Gamma \swarrow \Gamma^m)\big), \quad \Delta_1^a = \Delta^a,$$

and thus obtain $\mathcal{Q}'_1 = S(\mathcal{Q}_1, \Gamma_1^m \ \vdash\ \Delta_1^m, \Gamma_1^a \ \vdash\ \Delta_1^a)$. The root of $\mathcal{Q}'_1$ constructed like this is $(\Gamma \cup \{\lfloor \phi \rfloor_R\}) \swarrow \Gamma_1^m, \Gamma_1^a \ \vdash\ \Delta \swarrow \Delta_1^m, \Delta_1^a \ \blacktriangleright\ I'$. By proving that this sequent coincides with the left premise of OR-LEFT-L in $\mathcal{P}'$, we can then show that $\mathcal{P}'$ is indeed a well-formed proof.

5.2.2 The Chain Interpolation Theorem

It can be observed that the relabeling function S modifies the interpolants in a way satisfying the chain interpolation property, as introduced in the beginning of Section 5.2. To derive an interpolant chain $I_0, I_1, \ldots, I_n$ for a conjunction $T_1 \wedge \cdots \wedge T_n$, we start by constructing an interpolating proof $\mathcal{P}_0$ for the sequent

$$\lfloor T_1 \rfloor_R, \lfloor T_2 \rfloor_R, \ldots, \lfloor T_n \rfloor_R \ \vdash\ \emptyset \ \blacktriangleright\ I_0$$

This proof can then be transformed to a proof $\mathcal{P}_1 = S(\mathcal{P}_1, \{T_1\} \ \vdash\ \emptyset, \emptyset \ \vdash\ \emptyset)$ to obtain the next interpolant I_1 with $I_0 \wedge T_1 \Rightarrow I_1$, etc. The relationship between the interpolants $I_0, I_1, \ldots, I_n$ is more generally stated in the following theorem:

Theorem 2 (Chain Interpolation) *Suppose that proof $\mathcal{P}$ has root $\Gamma^r \ \vdash\ \Delta^r \ \blacktriangleright\ I^r$, and proof $S(\mathcal{P}, \Gamma^m \ \vdash\ \Delta^m, \Gamma^a \ \vdash\ \Delta^a)$ has root $\Gamma^r \swarrow \Gamma^m, \Gamma^a \ \vdash\ \Delta^r \swarrow \Delta^m, \Delta^a \ \blacktriangleright\ I'$. Then the (ordinary) sequent $\Gamma^m, I^r \ \vdash\ I', \Delta^m$ is valid.*

The theorem is proven by induction on the size of the proof $\mathcal{P}$, following the same case analysis as in the definition of S. Again, we show the rule OR-LEFT-R as an example and concentrate on the case $\phi \vee \psi \in \Gamma^m$. From the recursive application of S we know that the sequents $\Gamma_1^m, I \ \vdash\ I', \Delta_1^m$ and $\Gamma_2^m, J \ \vdash\ J', \Delta_2^m$ are valid, where $\Gamma_1^m = (\Gamma^m \cap \Gamma_R) \cup \{\phi\}$ and $\Gamma_2^m = (\Gamma^m \cap \Gamma_R) \cup \{\psi\}$. Due to $\phi \vee \psi \in \Gamma^m$, we can conclude the validity of $\Gamma^m, I \wedge J \ \vdash\ I' \vee J', \Delta^m$:

$$\dfrac{\dfrac{\dfrac{\dfrac{*}{\Gamma_1^m, I \ \vdash\ I', \Delta_1^m}}{(\Gamma^m \cap \Gamma_R) \cup \{\phi\}, I \ \vdash\ I', \Delta^m}}{\Gamma^m, \phi, I \ \vdash\ I', \Delta^m} \qquad \dfrac{\dfrac{\dfrac{*}{\Gamma_2^m, J \ \vdash\ J', \Delta_2^m}}{(\Gamma^m \cap \Gamma_R) \cup \{\psi\}, J \ \vdash\ J', \Delta^m}}{\Gamma^m, \psi, J \ \vdash\ J', \Delta^m}}{\dfrac{\dfrac{\Gamma^m, \phi \vee \psi, I, J \ \vdash\ I', J', \Delta^m}{\Gamma^m, I, J \ \vdash\ I', J', \Delta^m}}{\Gamma^m, I \wedge J \ \vdash\ I' \vee J', \Delta^m}}$$

A recent dissertation contains a detailed proof of the chain interpolation theorem [2].

5.2.3 Completeness of the Calculus

As a corollary of chain interpolation, we can also conclude the *completeness* of our calculus: whenever an implication $A \Rightarrow C$ holds, our calculus is able to derive an interpolant. We have to ban quantifiers that cannot be handled by Skolemization:

Theorem 3 (Completeness) *Suppose* Γ, Δ *are sets of labeled formulae* $\lfloor\phi\rfloor_L$ *and* $\lfloor\phi\rfloor_R$ *such that all occurrences of existential quantifiers in* Γ/Δ *are under an even/odd number of negations, and all occurrences of universal quantifiers in* Γ/Δ *are under an odd/even number of negations. If* $\Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_R$ *is valid, then there is a formula* I *such that* $\Gamma \vdash \Delta \blacktriangleright I$ *is provable.*

To justify this theorem, we first observe that the validity of $\Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_R$ implies the existence of a proof $\mathcal{P}_{\text{ni}}$ in the (complete) non-interpolating calculus from [20]. The non-interpolating proof can be lifted to a *trivial* interpolating proof $\mathcal{P}$ of the sequent $\{\lfloor\phi\rfloor_R \mid \phi \in \Gamma_L \cup \Gamma_R\} \vdash \{\lfloor\phi\rfloor_R \mid \phi \in \Delta_L \cup \Delta_R\} \blacktriangleright I'$ for some valid formula I' by simply labeling *all* formulae with R, uniformly replacing the rules from [20] with the corresponding interpolating rules presented in this paper (during this process, translating the SIMP′ rule may require applications of MUL-LEFT/RIGHT). Finally, $\mathcal{P}$ can be transformed to the final proof $\mathcal{Q}$ of the sequent $\Gamma \vdash \Delta \blacktriangleright I$ by applying the relabeling function S.

6 Strengthening and Rounding of Inequalities

Reasoning in linear integer arithmetic generally requires some kind of *cut rule* to deal with the phenomenon of formulae that are satisfiable over the rationals, but unsatisfiable over integers. The non-interpolating calculus in [18] provides two rules for this: the SIMP′ rule to round inequalities $\alpha t + \beta \leq 0$ to $\alpha t + \alpha\lceil\frac{\beta}{\alpha}\rceil \leq 0$ (which resembles Gomory cuts [22]), and the general STRENGTHEN′ rule:

$$\frac{\Gamma, t \doteq 0 \vdash \Delta \quad \Gamma, t+1 \leq 0 \vdash \Delta}{\Gamma, t \leq 0 \vdash \Delta}\ \text{STRENGTHEN}'$$

Because STRENGTHEN′ subsumes rounding via the rule SIMP′, we can ignore the latter rule for the time being and concentrate on STRENGTHEN′.

In order to lift STRENGTHEN′ to the interpolating calculus, we first observe two special cases that are easy to handle:

$$\frac{\Gamma, t \doteq 0\,[t \doteq 0] \vdash \Delta \blacktriangleright I \quad \Gamma, t+1 \leq 0\,[t+1 \leq 0] \vdash \Delta \blacktriangleright J}{\Gamma, t \leq 0\,[t \leq 0] \vdash \Delta \blacktriangleright I \vee J}\ \text{STRENGTHEN-L}$$

$$\frac{\Gamma, t \doteq 0\,[0 \doteq 0] \vdash \Delta \blacktriangleright I \quad \Gamma, t+1 \leq 0\,[0 \leq 0] \vdash \Delta \blacktriangleright J}{\Gamma, t \leq 0\,[0 \leq 0] \vdash \Delta \blacktriangleright I \wedge J}\ \text{STRENGTHEN-R}$$

These cases are called *pure cuts* in [14], because the PIs tell that the inequality $t \leq 0$ has been derived only from L- or only from R-formulae, respectively. Strengthening inequalities of this kind corresponds to splitting a disjunction labeled with L or R.

The general case is known as *mixed cut* [14] and encompasses an application of STRENGTHEN to a formula $t \leq 0\,[t^A \leq 0]$ with $t^A \notin \{0, t\}$. The rule for this general case is given in Fig. 5 and features *three* premises, rather than two as for

the non-interpolating rule STRENGTHEN′. To understand the shape of STRENGTHEN, note that we can represent $t \leq 0$ as the sum of $t^A \leq 0$ and $t - t^A \leq 0$, the first of which is derived from L-formulae, and the second from R-formulae. The effect of STRENGTHEN can now be simulated by applying STRENGTHEN-L to $t^A \leq 0\,[t^A \leq 0]$, and then STRENGTHEN-R to $t - t^A \leq 0\,[0 \leq 0]$; the combined application of the two rules explains the interpolant $I^1 \vee (E \wedge I^0)$ resulting from STRENGTHEN.

Complexity Non-interpolating refutations of unsatisfiable conjunctions of literals have exponential size in the worst case [22]. Similarly, it can be shown that any valid sequent (without quantifiers or propositional connectives) has interpolants of worst-case exponential size that can be derived using a proof of worst-case exponential size (using the rules STRENGTHEN-L/R from above).

In general, however, lifting a non-interpolating to an interpolating proof can increase the size of the proof exponentially, for two reasons: (i) STRENGTHEN in Fig. 5 has three premises, while the non-interpolating rule STRENGTHEN′ has only two, which can make it necessary to repeatedly duplicate subproofs during lifting (this is partly addressed in Section 6.1), and (ii) the rule SIMP′ (which is simulated by STRENGTHEN in the interpolating calculus) often allows very succinct proofs. As a result, there are unsatisfiable conjunctions $A \wedge B$ with non-interpolating proofs of linear size, although all interpolants have exponential size.

6.1 Successive Strengthening

It is quite common that STRENGTHEN is applied repeatedly to a sequence $t \leq 0$, $t + 1 \leq 0$, $t + 2 \leq 0$, … of inequalities, for instance to simulate rounding of an inequality, or the Omega rule [17]. Because each application of STRENGTHEN generates two new inequalities, $2^k - 1$ applications are necessary in order to strengthen an inequality $t \leq 0$ to $t + k \leq 0$, and the resulting interpolant will be of exponential size as well. To curb this explosion, we present an optimized rule that captures k-fold strengthening and requires only a quadratic number of premises. The optimized rule k-STRENGTHEN exploits the fact that many of the goals created by repeated application of STRENGTHEN are redundant:

$$\frac{\begin{array}{l}\left\{\Gamma, t + i \doteq 0\,[t_A + j \doteq 0] \;\vdash\; \Delta \;\blacktriangleright\; E_i^j\right\}_{0 \leq j \leq i < k} \\ \left\{\Gamma, t + k \leq 0\,[t_A + j \leq 0] \;\vdash\; \Delta \;\blacktriangleright\; I^j\right\}_{0 \leq j \leq k}\end{array}}{\Gamma, t \leq 0\,[t_A \leq 0] \;\vdash\; \Delta \;\blacktriangleright\; K}\; k\text{-STRENGTHEN}$$

where the resulting interpolant K is defined by:

$$K = \bigvee_{0 \leq j \leq k} \left(I^j \wedge \bigwedge_{j \leq i < k} E_i^j \right) \tag{11}$$

The size of K grows quadratically, rather than exponentially, in k. Thus, whenever the STRENGTHEN rule is to be applied k times in succession, it is possible and more efficient to use the k-STRENGTHEN rule instead.

The number of premises of k-STRENGTHEN (but not the size of the resulting interpolant) can be reduced further to a linear number: any two premises generating E_i^j and E_i^l differ only in the partial interpolant of $t + i \leq 0$. We can exploit this by

treating the family $(E_i^j)_{0\le j\le i}$ as a *single* premise, parameterized in the free variable j. This way, a single subproof can generate a parameterized interpolant $E_i(j)$. Parameter j can be instantiated to the values $0 \le j \le i$ when constructing K. Parametrized interpolants $I(j)$ are derived similarly.

Interpolation of Rounding Operations An additional optimization is possible when the rule k-STRENGTHEN is used to round an inequality $\alpha t + \beta \le 0$ to $\alpha t + \alpha\lceil\frac{\beta}{\alpha}\rceil \le 0$. Rounding corresponds to k-STRENGTHEN with $k = \alpha\lceil\frac{\beta}{\alpha}\rceil - \beta$:

$$\frac{\left\{\Gamma, \alpha t+\beta+i \doteq 0\,[t_A + j \doteq 0] \;\vdash\; \Delta \;\blacktriangleright\; E_i^j\right\}_{0\le j\le i<k} \quad \left\{\Gamma, \alpha t+\alpha\lceil\tfrac{\beta}{\alpha}\rceil \le 0\,[t_A + j \le 0] \;\vdash\; \Delta \;\blacktriangleright\; I^j\right\}_{0\le j\le k}}{\Gamma, \alpha t+\beta \le 0\,[t_A \le 0] \;\vdash\; \Delta \;\blacktriangleright\; K}\; k\text{-STRENGTHEN}$$

We can observe that $\alpha t + \beta + i \doteq 0$ is unsatisfiable for $0 \le i < \alpha\lceil\frac{\beta}{\alpha}\rceil - \beta$, so that the equality-premises can be closed immediately via CLOSE-EQ-LEFT. Consequently, the interpolants $E_i^j = E^j = (\exists_{LA}\, t^A + j \doteq 0)$ do not depend on i, and the overall interpolant can be simplified to $K = I^k \vee \bigvee_{0\le j<k}(I^j \wedge E^j)$.

If (bounded) quantifiers in interpolants are acceptable, the last formula K can also be encoded compactly without any duplication of sub-formulae:

$$K \;\equiv\; \exists x.\, \big(-x \le 0 \wedge x - k \le 0 \wedge I(x) \wedge (E(x) \vee x - k \doteq 0)\big)$$

A similar observation was made in [9], where it was shown that interpolants can be extracted from cutting-planes proofs in polynomial time if the interpolant language is augmented by an operation for scaling with rational coefficients and the ceiling function $\lceil\cdot\rceil$.

Example We use k-STRENGTHEN to compute an interpolant for the formula $A \wedge B$ with $A = -y + 5x - 1 \le 0 \wedge y - 5x \le 0$ and $B = 5z - y + 1 \le 0 \wedge -5z + y - 2 \le 0$. Note that $A \wedge B$ is satisfiable over rationals, but unsatisfiable over the integers. An interpolating proof of unsatisfiability is as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots\,*}{\{\cdots \blacktriangleright E_i^j\}} \quad \dfrac{*}{\ldots, 1 \le 0\,[j-1 \le 0] \;\vdash\; \blacktriangleright\, j-1 \le 0}\,\text{CLOSE-INEQ}}{\ldots, -5z+5x \le 0\,[-y+5x-1+j \le 0],\; 5z-5x+1 \le 0\,[y-5x \le 0] \;\vdash\; \blacktriangleright\, j-1 \le 0}\,\text{FM-ELIM}}{\ldots, -5z+5x-3 \le 0\,[-y+5x-1 \le 0], 5z-5x+1 \le 0\,[y-5x \le 0] \;\vdash\; \blacktriangleright\, K}\,\text{3-STRENGTHEN}}{\ldots, y-5x \le 0\,[y-5x \le 0], 5z-y+1 \le 0\,[0 \le 0],\; -5z+5x-3 \le 0\,[-y+5x-1 \le 0] \;\vdash\; \blacktriangleright\, K}\,\text{FM-ELIM}}{-y+5x-1 \le 0\,[-y+5x-1 \le 0], y-5x \le 0\,[y-5x \le 0],\; 5z-y+1 \le 0\,[0 \le 0], -5z+y-2 \le 0\,[0 \le 0] \;\vdash\; \blacktriangleright\, K}\,\text{FM-ELIM}}{\lfloor -y+5x-1 \le 0\rfloor_L, \lfloor y-5x \le 0\rfloor_L, \lfloor 5z-y+1 \le 0\rfloor_R, \lfloor -5z+y-2 \le 0\rfloor_R \;\vdash\; \blacktriangleright\, K}\,\text{IPI}^+}{\lfloor -y+5x-1 \le 0 \wedge y-5x \le 0\rfloor_L, \lfloor 5z-y+1 \le 0 \wedge -5z+y-2 \le 0\rfloor_R \;\vdash\; \blacktriangleright\, K}\,\text{AND-LEFT}^+$$

Most importantly, the rule 3-STRENGTHEN is used to round $-5z + 5x - 3 \le 0$ to $-5z + 5x \le 0$, from which a contradiction can be derived via FM-ELIM. The inequality interpolants $I^j = (j - 1 \le 0)$, as well as the equality interpolants $E^j = (\exists x.\, -y + 5x - 1 + j \doteq 0) \equiv (5 \mid (y + 1 - j))$ in the premises of 3-STRENGTHEN are derived as discussed above. The overall interpolant is:

$$K = \underbrace{3-1 \le 0}_{I^k} \vee \bigvee_{0\le j<3} \underbrace{(j-1 \le 0)}_{I^j} \wedge \underbrace{5 \mid (y+1-j))}_{E^j} \quad\equiv\quad 5 \mid (y+1) \vee 5 \mid y$$

6.2 A Comparison with Branch-and-Bound and Gomory Cuts

In order to illustrate the generality of our calculus, we discuss how it can be used to naturally simulate two of the most common cut/splitting rules used in SMT solvers: the branch-and-bound rule and Gomory cuts. Both rules follow the concept of *rational relaxation:* in order to solve an integer linear program, it is checked whether the program is satisfiable over rational numbers. If this check derives a rational solution $\beta : C \to \mathbb{Q}$ that is not integral, nothing can directly be concluded about the program; however, the solution β can be used to refine the integer program by generating additional constraints that exclude β.

6.2.1 Branch-and-Bound

Branch-and-bound generates new constraints by selecting an arbitrary constant $c \in C$ such that $\beta(c) \notin \mathbb{Z}$, considering the two cases $c \leq \lfloor \beta(c) \rfloor$ and $c \geq \lceil \beta(c) \rceil$. Because a single constant c is necessarily a *pure* term (which does not mix local symbols from L- and R-formulae), it is easy to interpolate the case analysis introduced by branch-and-bound; in fact, this does not even require the STRENGTHEN rule. For instance, if c is a constant occurring in L-formulae, branch-and-bound corresponds to an application of the rule OR-LEFT-L:

$$\dfrac{\dfrac{\vdots}{\Gamma, c - \lfloor \beta(c) \rfloor \leq 0 \ \vdash \ \Delta \ \blacktriangleright I} \qquad \dfrac{\vdots}{\Gamma, \lceil \beta(c) \rceil - c \leq 0 \ \vdash \ \Delta \ \blacktriangleright J}}{\Gamma \ \vdash \ \Delta \ \blacktriangleright I \vee J}$$

R-constants can be handled in a similar manner.

6.2.2 Gomory Cuts

It is well-known that the application of the branch-and-bound rule does not necessarily terminate for unsatisfiable integer programs [22], and does not give rise to a decision procedure for QFPA. In solvers, branch-and-bound is therefore often combined with Gomory cuts, which derive additional constraints from non-integral solutions in a more sophisticated manner and indeed achieve completeness. We follow the version of Gomory cuts introduced in [7, Section 4.2.1], which has become the standard solution used in SMT solvers. As we do not consider mixed-integer cuts, the rule described here is somewhat simpler than the one in [7].

Carrying over the approach from [7] to our calculus, and in particular only considering integral coefficients in constraints, a Gomory cut can be applied if reasoning has resulted in a sequent[2]

$$\Gamma, \ \begin{array}{c} \{l_j - x_j \leq 0\}_{j \in J}, \ \{x_j - u_j \leq 0\}_{j \in K}, \\ a_{ii}\, x_i - \sum_{j \in J} a_{ij}\, x_j - \sum_{j \in K} a_{ij}\, x_j \doteq 0 \end{array} \ \vdash \ \Delta \tag{12}$$

in which J, K are disjoint sets of indexes, $i \notin J \cup K$, the variables x_j are bounded by the integers $\{l_j \mid j \in J\} \subset \mathbb{Z}$ and $\{u_j \mid j \in K\} \subset \mathbb{Z}$, the coefficient $a_{ii} \in \mathbb{Z}$ is positive, and all of the coefficients $\{a_{ij} \mid j \in J \cup K\} \subset \mathbb{Z}$ are non-zero. The equation of the

[2]The procedure in [7] normalizes all inequalities to the form $l_j - x_j \leq 0$ or $x_j - u_j \leq 0$ prior to the actual solving process.

sequent corresponds to a line of a Simplex tableau. Furthermore, we require that the bounds l_j, u_j correspond to a non-integral extremal solution:

$$\left(\sum_{j\in J} a_{ij} l_j - \sum_{j\in K} a_{ij} u_j\right) = b_0 + a_{ii}\, b_1, \qquad b_0 \in \{1, \ldots, a_{ii} - 1\}, b_1 \in \mathbb{Z}.$$

As [7] shows, in this situation it is sound to introduce the additional constraint

$$\begin{aligned} b_0 \sum_{j\in J^+} a_{ij}\,(l_j - x_j) \;-\; (a_{ii} - b_0) \sum_{j\in J^-} a_{ij}\,(l_j - x_j) \\ +\; (a_{ii} - b_0) \sum_{j\in K^+} a_{ij}\,(x_j - u_j) \\ -\; b_0 \sum_{j\in K^-} a_{ij}\,(x_j - u_j) \le -b_0\,(a_{ii} - b_0) \end{aligned} \tag{13}$$

where the index sets J^+, J^-, K^+, K^- are defined as follows:

$$J^+ = \{j \in J \mid a_{ij} \ge 0\}, \qquad J^- = \{j \in J \mid a_{ij} < 0\},$$
$$K^+ = \{j \in K \mid a_{ij} \ge 0\}, \qquad K^- = \{j \in K \mid a_{ij} < 0\}.$$

We can observe that the Gomory cut (13) can also be generated using the rules presented in this paper, which directly implies that interpolants can be extracted. This is done by the following sequence of rule applications:

(i) Since the left-hand side of (13) is a positive linear combination of inequalities in (12), the inequality (13) can be introduced using the rule FM-ELIM followed by an application of k-STRENGTHEN with $k = b_0\,(a_{ii} - b_0)$.

(ii) In those premises of k-STRENGTHEN that introduce an equality, we can first distinguish the two cases $a_{ii}\, x_j \ge b_0 + a_{ii}\, b_1$ and $a_{ii}\, x_j < b_0 + a_{ii}\, b_1$ as discussed in Section 6.2.1; the cases can subsequently be rounded to $a_{ii}\, x_j \ge a_{ii}\,(b_1 + 1)$ and $a_{ii}\, x_j \le a_{ii}\, b_1$. The resulting goals can be closed by a sequence of FM-ELIM and RED-LEFT applications, essentially formalizing the justification for (13) that is given in [7] in our sequent calculus.

7 Experimental Evaluation

We implemented the presented interpolating calculus on top of the PRINCESS theorem prover [20],[3] including all optimizations described in Section 6. Interpolation is performed in PRINCESS in two steps: the prover first (internally) generates a non-interpolating proof, which is then processed using the rules presented in this article to extract interpolants. To keep the size of proofs manageable, we also implemented a number of proof reduction heuristics, e.g., eliminating simplification steps that were later found to be unnecessary.

[3] http://www.philipp.ruemmer.org/princess.shtml.

The benchmarks for our experiments are derived from different families of the SMT-LIB category QF-LIA. Because SMT-LIB benchmarks are usually conjunctions at the outermost level, we can partition them to the form $T_1 \wedge T_2 \wedge \cdots \wedge T_{10}$ by choosing the first 10% of the benchmark conjuncts as T_1, the second 10% as T_2, etc. We then compute chains $I_0, I_1, \ldots, I_{10}$ of interpolants from a single proof of unsatisfiability of $T_1 \wedge T_2 \wedge \cdots \wedge T_{10}$. Since I_0 and I_{10} are trivial, this yields 9 interpolation problems for each SMT-LIB benchmark. We choose this setup since model checkers often require the computation of similar chains of interpolants.

We compare our procedure with the following tools:

- SMTInterpol,[4] which is an interpolation engine for quantifier-free linear integer arithmetic (among others) and thus targets a similar theory as Princess. We are not aware of a publication that describes the algorithms behind SMTInterpol.
- the interpolating version of the OpenSMT [4] solver that was developed in our previous work [13, together with Jérôme Leroux].
- CSIsat [1] (also see [21]), an interpolation procedure for rational arithmetic and uninterpreted functions that reduces interpolation problems to a set of linear constraints, which are solved using LP techniques. A comparison with CSIsat is interesting as it is not based on proof construction; the fact that proofs can get large is often perceived to be a handicap of proof-based methods.
- the Omega quantifier elimination procedure [17], which is used to generate interpolants by eliminating all symbols local to the left conjunct A in an interpolation problem. We use the implementation of Omega available in Princess.

Our experimental results are shown in Table 1 and Fig. 8:

- the number of unsatisfiable/satisfiable problems tested, and the number of SAT/UNSAT results that the tools were able to derive; in the remaining cases, either a timeout or a memory-out occurred. These data are not provided for QE, as it does not decide satisfiability of interpolation problems.
- the total number of quantifier-free interpolants that could be computed. For OpenSMT, SMTInterpol, and CSIsat, which compute interpolants on the fly while solving a problem, this is always 9× the number of UNSAT results. Princess first constructs a proof for a problem, and afterwards extracts interpolants, which means that sometimes not all 9 interpolants can be computed for a benchmark, due to the potentially high complexity of rewriting a non-interpolating to an interpolating proof.
- the average time (in seconds) required to solve each benchmark, including the time to compute the 9 interpolants. For QE, this is simply the average time to compute 9 interpolants.
- the average size of generated interpolants, in terms of the number of equations, inequalities, and occurrences of propositional variables in the interpolant. The sizes of generated interpolants are also compared in the scatter plots in Fig. 8.

Comparison with OpenSMT and SMTInterpol The experiments show that Princess is overall competitive with OpenSMT and SMTInterpol. On benchmarks with a focus on arithmetic (*Multiplier*, *Bitadder*, *Rings*), Princess can uniformly solve the

[4] http://swt.informatik.uni-freiburg.de/research/tools/smtinterpol.

Table 1 Results of applying the compared tools to SMT-LIB benchmarks (times in seconds)

	Multiplier 16 unsat 1 sat	Bitadder 17 unsat	Mathsat 100 unsat	Rings 294 unsat	Convert 38 unsat 109 sat 172 unkn.
Princess	**8**/1/41 154/**1623**	**7**/0/**63** 298/76953	44/13/396 106/7007	**133**/0/**183** 249/5984	**39**/**82**/**334** 87.8/**1**
OpenSMT	5/1/**45** 48.9/2357	**7**/0/**63** 103/**23362**	**74**/**15**/**666** 53.0/**2020**	9/0/81 **59.9**/**4611**	37/0/333 **0.08**/**1**
SMTInterpol	5/1/**45** **24.4**/48827	5/0/45 **8.58**/41077	65/13/585 **45.7**/126705	0/0/– –/–	37/0/333 13.6/2
CSIsat	4/1/36 106/2640	1/0/9 0.56/188	25/12/225 70.8/12683	(1)	(2)
Omega QE	–/–/125 109/15392	–/–/129 97.8/93181	–/–/612 169/101088	–/–/1474 227/55307	–/–/297 15.0/2659
	#unsat / #sat / #interpolants / average time (s) / average int. size				

Bold numbers highlight the respective best results. Experiments were done on an Intel Xeon X5667 4-core machine with 3.07GHz, heap-space limited to 12 GB, running Linux, with a timeout of 900 s. In (1), no interpolants could be computed, since the benchmarks were found to be rationally satisfiable by CSIsat. In (2), CSIsat could not handle large literal constants occurring in the benchmarks

largest number of problems, although usually with a somewhat longer solving time, and not always being able to extract interpolants from proofs. Comparing the method from [13] with the procedure introduced in this paper, our calculus allows us to work with virtually arbitrary cut rules (in the implementation in Princess, this is the Omega rule), at the cost of worst-case exponential complexity of interpolant extraction (see Section 6), while [13] uses a branch-and-cut rule that can be interpolated with polynomial complexity, but might lead to exponentially larger proofs. *Rings* represents an example where our method is often able to construct a proof but times out when extracting interpolants, while the method from [13] already times out when constructing a proof.

In the *Mathsat* family of benchmarks, which test propositional reasoning capability more than arithmetic performance, Princess can solve fewer problems than the other tools, but can easily generate interpolants once a proof is found. An explanation for the performance on *Mathsat* is that Princess does not learn lemmas during proof search, in contrast to SMT solvers like OpenSMT and SMTInterpol. On *Convert*, which contains bitvector problems encoded in integer arithmetic, all tools show a very similar performance; Princess is able to prove 82 of the benchmarks to be SAT but usually needs a long time to construct a model, which explains the larger average solving time compared to the other tools. Looking only at UNSAT problems, the average solving time needed by Princess is around 1*s*.

Considering the size of generated interpolants (Fig. 8), Princess shows roughly the same performance as SMTInterpol, and tends to generate somewhat larger interpolants than OpenSMT. However, OpenSMT computes and outputs interpolants in DAG representation, while the interpolants produced by the other tools are already fully expanded to trees, so that the figures are not fully comparable.

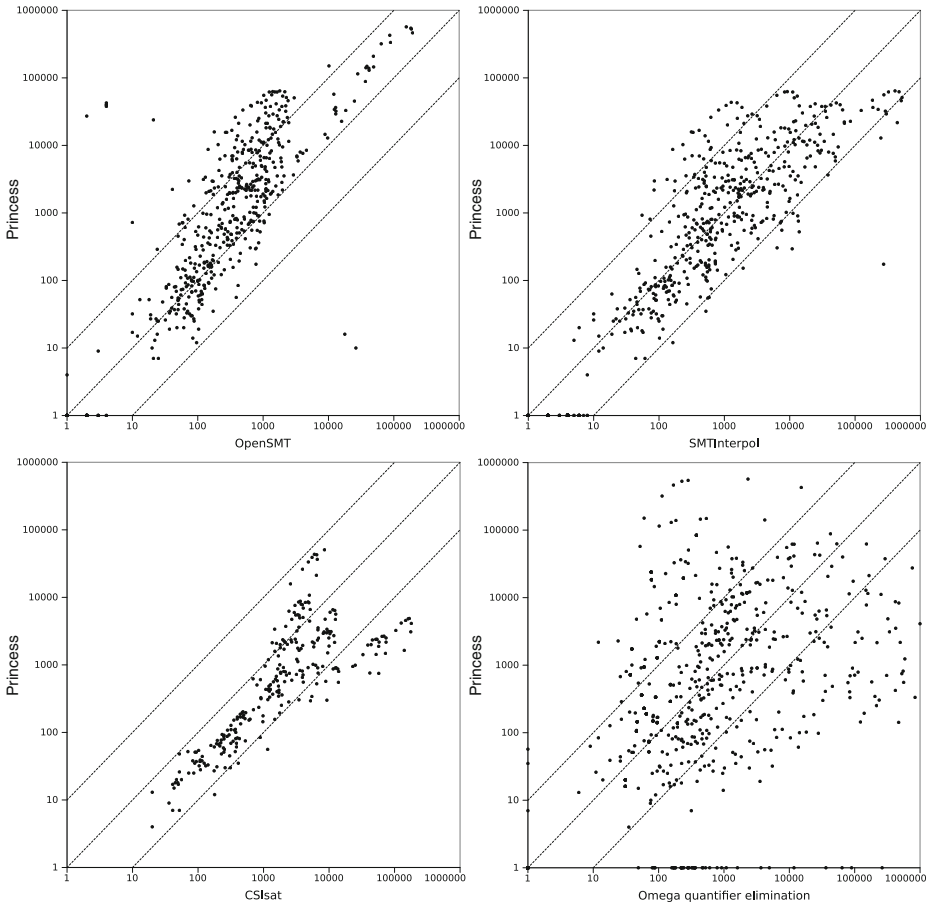

Fig. 8 Comparison of the size of interpolants generated by PRINCESS, OPENSMT, SMTINTERPOL, CSISAT, and quantifier elimination using the OMEGA procedure

Comparison with CSISAT The rational interpolation tool CSISAT could be applied to the benchmarks in *Multiplier*, *Bitadder*, *Mathsat*. Since CSISAT produced a "wrong" answer (SAT instead of UNSAT) only for a single benchmark in *Multiplier*, it can be concluded that most of the reasoning in those benchmark groups is not integer specific. In contrast, CSISAT reported SAT (over rationals) for almost all benchmarks in *Rings*, which are specifically designed to test integer reasoning. We also observed that CSISAT has difficulties handling large literals such as 2^{64}, which may arise, e.g., in overflow checks, and which occur in the *Convert* benchmarks.

The experiments show that CSISAT could solve fewer problems than PRINCESS, OPENSMT, and SMTINTERPOL, while requiring a similar amount of time and producing larger interpolants. We suspect that constraint-based interpolation tends to generate larger interpolants than proof-based methods on our benchmarks. Since it was shown that the performance of CSISAT is comparable or better than that of other standard interpolation procedures for rational arithmetic [1], it seems that the overhead of reasoning in integer arithmetic (for problems that can be interpolated even in rational arithmetic) is negligible.

Comparison with Quantifier Elimination QE is able to generate a large number of interpolants in the families *Multiplier*, *Bitadder*, *Mathsat*, and *Rings*, albeit the generation is slow and the interpolants are large, on average. We observed that QE typically performs well (and produces small interpolants) when the left conjunct A only contains few local symbols, i.e., when few quantifiers need to be eliminated. With an increasing number of local symbols, the performance of QE quickly degrades. In contrast, with proof-based approaches the number of A-local symbols is less relevant for performance or the size of interpolants. A more detailed discussion of quantifier elimination-based interpolation can be found in [3].

8 Related Work and Concluding Discussions

Interpolation for propositional logic, linear rational arithmetic, and uninterpreted functions is a well-explored field. In particular, McMillan presents an interpolating theorem prover for rational arithmetic and uninterpreted functions [15]; an interpolating SMT solver for the same logic has been developed by Beyer et al. [1] (see Section 7 for a comparison). Rybalchenko et al. [21] introduce an interpolation procedure for this logic that works without constructing proofs.

Interpolation has also been investigated in several fragments of integer arithmetic. McMillan considers the logic of difference-bound constraints [16], which is decidable by reduction to rational arithmetic. As an extension, Cimatti et al. [5] present an interpolation procedure for the $\mathcal{UTVPI}$ fragment of linear integer arithmetic. Both fragments allow efficient reasoning and interpolation, but are not sufficient to express many typical program constructs, such as integer division. In [10], separate interpolation procedures for two theories are presented, namely (i) QFPA restricted to conjunctions of integer linear (dis)equalities and (ii) QFPA restricted to conjunctions of stride constraints. The combination of both fragments with integer linear inequalities is, however, not supported.

The first complete interpolation methods (other than by quantifier elimination) for quantifier-free Presburger arithmetic are [3], of which the present paper is an extended version, and [13]. The latter proposes an approach based on the Simplex method to solve a decision problem over the rationals, combined with a branch-and-cut rule that can be interpolated efficiently. Recently, a new method to extract interpolants in the presence of ordinary Gomory cuts was presented [9]. Avoiding the otherwise high complexity of interpolating mixed cuts, this method relies on an extended language for interpolants; also see Section 6.1 for a discussion. The SMT solver SMTInterpol decides and interpolates problems in linear integer arithmetic, apparently using an architecture similar to the one in [15]. To the best of our knowledge, the precise design and calculus of SMTInterpol has not been documented in publications yet.

Kapur et al. [12] prove that full QFPA is closed under interpolation (as an instance of a more general result about recursively enumerable theories), but their proof does not directly give rise to an efficient interpolation procedure. Lynch et al. [14] define an interpolation procedure for linear rational arithmetic, and extend it to integer arithmetic by means of Gomory cuts. No interpolating rule is provided for mixed cuts, however, which means that sometimes formulae are generated that are not true interpolants because they violate the vocabulary condition (i.e., contain symbols that are not common to A and B).

In conclusion, we have presented a sound and complete interpolating sequent calculus for quantifier-free Presburger arithmetic. We have shown that the resulting interpolation approach is flexible and can be combined with a variety of decision procedures for full quantifier-free Presburger arithmetic, including the ones common in modern SMT solvers. The interpolation procedure also guarantees the chain interpolation property that is important for model checking applications. We have implemented the interpolation procedure on top of the Princess theorem prover and demonstrated experimentally that the procedure is competitive with available interpolation tools.

We are integrating our interpolation procedure into a software model checker based on lazy abstraction [16]. The model checker uses interpolation to refine the abstraction and avoids the expensive image computation required by predicate abstraction. When using our QFPA interpolation procedure, we expect to be able to verify software with more complex numerical features than other model checkers.

Acknowledgements We thank Jérôme Leroux, Vijay D'Silva, Georg Weissenbacher, and the anonymous referees for their comments.

References

1. Beyer, D., Zufferey, D., Majumdar, R.: CSIsat: interpolation for LA+EUF. In: CAV. LNCS, vol. 5123, pp. 304–308. Springer (2008)
2. Brillout, A.: Approximating and interpolating theories of arithmetic for software verification. Ph.D. thesis, ETH Zürich (2011)
3. Brillout, A., Kroening, D., Rümmer, P., Wahl, T.: An interpolating sequent calculus for quantifier-free Presburger arithmetic. In: Proceedings, International Joint Conference on Automated Reasoning (IJCAR). LNCS, vol. 6173, pp. 384–399. Springer (2010)
4. Bruttomesso, R., Pek, E., Sharygina, N., Tsitovich, A.: The OpenSMT solver. In: TACAS, LNCS, pp. 150–153. Springer (2010)
5. Cimatti, A., Griggio, A., Sebastiani, R.: Interpolant generation for UTVPI. In: Schmidt, R.A. (ed.) CADE, LNCS, vol. 5663, pp. 167–182. Springer (2009)
6. Craig, W.: Linear reasoning. A new form of the Herbrand–Gentzen theorem. J. Symb. Log. **22**(3), 250–268 (1957)
7. Dutertre, B., de Moura, L.: Integrating Simplex with DPLL(T). Tech. Rep. SRI-CSL-06-01, SRI International (2006)
8. Fitting, M.C.: First-Order Logic and Automated Theorem Proving, 2nd edn. Springer (1996)
9. Griggio, A., Le, T.T.H., Sebastiani, R.: Efficient interpolant generation in satisfiability modulo linear integer arithmetic. In: TACAS, LNCS, vol. 6605, pp. 143–157. Springer (2011)
10. Jain, H., Clarke, E., Grumberg, O.: Efficient interpolation for linear diophantine (dis)equations and linear modular equations. In: CAV, LNCS, pp. 254–267. Springer (2008)
11. Kannan, R., Bachem, A.: Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix. SIAM J. Comput. **8**(4), 499–507 (1979)
12. Kapur, D., Majumdar, R., Zarba, C.G.: Interpolation for data structures. In: SIGSOFT '06/FSE-14, pp. 105–116. ACM (2006)
13. Kroening, D., Leroux, J., Rümmer, P.: Interpolating quantifier-free Presburger arithmetic. In: Proceedings, LPAR. LNCS, vol. 6397, pp. 489–503. Springer (2010)
14. Lynch, C., Tang, Y.: Interpolants for linear arithmetic in SMT. In: ATVA. LNCS, pp. 156–170. Springer (2008)
15. McMillan, K.L.: An interpolating theorem prover. Theor. Comp. Sci. **345**(1), 101–121 (2005)
16. McMillan, K.L.: Lazy abstraction with interpolants. In: Ball, T., Jones, R.B. (eds.) Computer Aided Verification (CAV). LNCS, vol. 4144, pp. 123–136. Springer (2006)
17. Pugh, W.: The Omega test: a fast and practical integer programming algorithm for dependence analysis. Commun. ACM **8**, 102–114 (1992)

18. Rümmer, P.: A sequent calculus for integer arithmetic with counterexample generation. In: Verification Workshop (VERIFY). CEUR Workshop Proceedings, vol. 259 (2007)
19. Rümmer, P.: Calculi for program incorrectness and arithmetic. Ph.D. thesis, University of Gothenburg (2008)
20. Rümmer, P.: A constraint sequent calculus for first-order logic with linear integer arithmetic. In: Proceedings, LPAR. LNCS, vol. 5330, pp. 274–289. Springer (2008)
21. Rybalchenko, A., Sofronie-Stokkermans, V.: Constraint solving for interpolation. In: Proceedings, VMCAI. LNCS, vol. 4349, pp. 346–362. Springer (2007)
22. Schrijver, A.: Theory of Linear and Integer Programming. Wiley (1986)