An Interpolating Sequent Calculus for Quantifier-Free Presburger Arithmetic*

Angelo Brillout[1], Daniel Kroening[2], Philipp Rümmer[2], and Thomas Wahl[2]

[1] ETH Zurich, Switzerland
[2] Oxford University Computing Laboratory, United Kingdom

Abstract. Craig interpolation has become a versatile tool in formal verification, for instance to generate intermediate assertions for safety analysis of programs. Interpolants are typically determined by annotating the steps of an unsatisfiability proof with *partial interpolants*. In this paper, we consider Craig interpolation for full *quantifier-free Presburger arithmetic* (QFPA), for which currently no efficient interpolation procedures are known. Closing this gap, we introduce an *interpolating sequent calculus* for QFPA and prove it 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 linear integer arithmetic benchmarks. The results indicate the robustness and efficiency of our proof-based interpolation procedure.

1 Introduction

Craig interpolation [3], a principle known to logicians since the 1950s, has recently emerged in formal verification as a practical approximation method. Its applications range from efficient image computations in SAT-based model checking to accelerating convergence of fixpoint calculations for infinite-state systems. 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 I contains only non-logical symbols occurring in both A and C. Interpolants exist for any two first-order formulae A and C such that $A \Rightarrow C$. As is common in formal verification, we also consider interpolation for unsatisfiable conjunctions $A \wedge B$, which corresponds to $C = \neg B$ in the above formulation.

In software verification, interpolation is applied to formulae encoding the transition relation of a model underlying a program. In order to support expressive programming languages, much effort has been invested in the design of algorithms that compute interpolants for formulae of various theories. As a result, efficient interpolation methods are known for propositional logic, linear arithmetic over the reals with uninterpreted functions [10, 1, 16], datastructures like arrays and sets [7], and other theories. As for integer arithmetic, a theory

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

particularly relevant for software, interpolating solvers have so far been reported only for restricted fragments such as difference-bound logic, and logics with linear equalities and constant-divisibility predicates. For these theories, an interpolant can be derived in time polynomial in the size of the input formulae.

In this paper, we push the boundaries of interpolation-based software model checking by presenting an interpolation method for full quantifier-free *Presburger arithmetic* (QFPA), i.e., linear arithmetic over the integers. This theory has been used, besides others, to model the behavior of infinite-state programs and of hardware designs. Presburger arithmetic was shown to be decidable by quantifier elimination [12]. A brute-force interpolation method is to quantify out the variables not common to the input formulae, and then to eliminate those quantifiers. This approach suffers, however, from the triply-exponential complexity of the elimination procedure and tends to be ineffective in many practical cases.

A more promising approach (that has also been used, e.g., in [10, 1, 8, 5]) is to extract interpolants directly from an unsatisfiability proof for $A \wedge B$. To this end, 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.

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 [17] or the Omega rule [13]) 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 [8, 10], 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 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 [15] 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, such as from the QF-LIA category of the SMT library. We compare the efficiency of the prover to the only currently known interpolation method for Presburger arithmetic, which is based on local-variable quantification and subsequent brute-force quantifier elimination (QE). Our experiments not only demonstrate the weaknesses of interpolation using QE, but also indicate the robustness and efficiency of our proof-based interpolation procedure, in terms of both time and interpolant size.

2 Preliminaries

Presburger arithmetic. We assume familiarity with classical first-order logic (e.g., [4]). 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:

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

The symbol t denotes terms of linear arithmetic. For simplicity, we only allow 0 as the right-hand side of equalities and inequalities. The explicit divisibility operator $\alpha \mid t$, which is short for $\exists s.\, \alpha s - t \doteq 0$, is included to permit quantifier-free interpolants for formulae such as $y - 2x \doteq 0 \wedge y - 2z - 1 \doteq 0$, with interpolant $2 \mid y$. We use the abbreviations *true* and *false* for the equalities $0 \doteq 0$ and $1 \doteq 0$, and $\phi \rightarrow \psi$ 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 short-hand notation, we sometimes also quantify over constants (as in $\forall c.\phi$) and assume that the constants are implicitly replaced by fresh variables. For reasons of presentation, we further assume that terms t 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.

The semantics of Presburger arithmetic is defined over the universe $\mathbb{Z}$ of integers in the standard way [4]. Furthermore, we only allow quantifiers that can be handled by Skolemization (only universal/existential quantifiers under an even/odd number of negations).

Gentzen-style sequent calculi. If Γ, Δ are finite sets of formulae and C is a formula, all without free variables, then $\Gamma \vdash \Delta$ is a *sequent*. The sequent is *valid* if the formula $\bigwedge \Gamma \rightarrow \bigvee \Delta$ 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. Proof trees are defined to grow upwards. 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 for QFPA presented in this paper extends the ground fragment of the sequent calculus in [15].

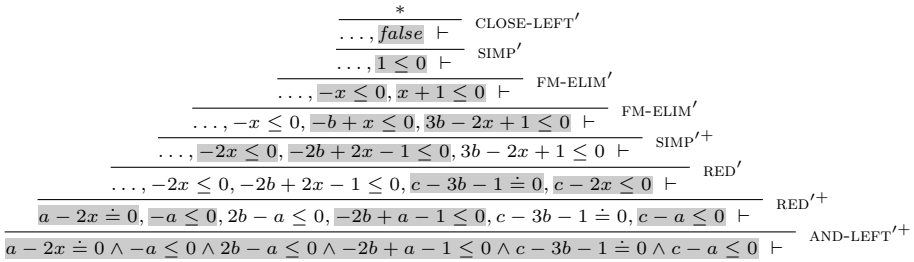

Fig. 1. Unsatisfiability proof for the examples of Sect. 3

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. Fig. 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.

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 equation (1) encoding the program up to this point. Currently, the only known interpolation method for QFPA is to quantify out the 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 the remaining part of (1)) 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:

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

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

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

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 from 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., [11]). 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 Sect. 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 [4], 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 [10]).

More formally, if ϕ is a formula and t, t^A are terms, all without free variables, then $\lfloor \phi \rfloor_L$ 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. Furthermore, if Γ, Δ are sets of labeled formulae and I is an unlabeled formula such that (i) none of the formulae contains free variables, (ii) 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 (iii) Δ only contains formulae $\lfloor \phi \rfloor_L$, $\lfloor \phi \rfloor_R$, $t \doteq 0\,[t^A \doteq 0]$, or $t \doteq 0\,[t^A \not\doteq 0]$, then $\Gamma \vdash \Delta \blacktriangleright I$ is an *interpolating sequent*.

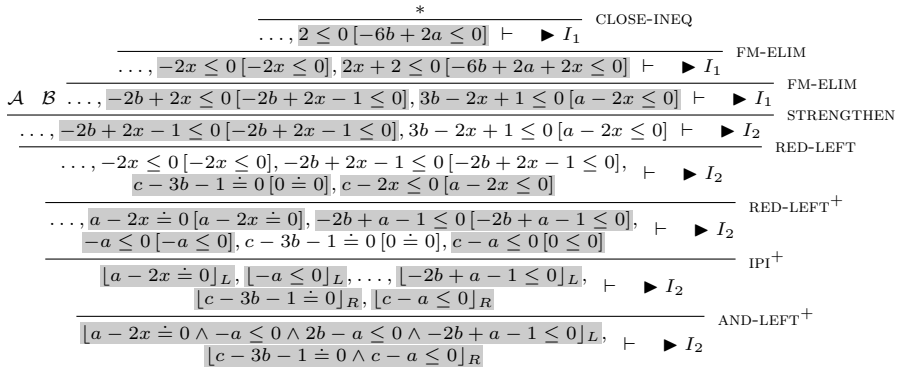

Fig. 2. The interpolating version of Fig. 1. The initial interpolant generated by CLOSE-INEQ is $I_1 = (-6b + 2a \leq 0) \equiv (-3b + a \leq 0)$, which is by STRENGTHEN combined with the interpolants *false* and ϕ from the subproofs $\mathcal{A}$ and $\mathcal{B}$ to form the final interpolant $I_2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

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. A sequent $\Gamma \vdash \Delta \blacktriangleright I$ is *valid* if (i) the (Gentzen-style) 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 Sect. 4.3; intuitively, a labeled formula $\phi\,[\phi^A]$ in an interpolation problem $A \wedge B$ expresses the implications $A \Rightarrow \phi^A$ and $B \wedge \phi^A \Rightarrow \phi$.

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. 2, 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 Sect. 4.3), 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 Sect. 3 as a succinct interpolant and intermediate program assertion.

In the remainder of Sect. 4, we explain the rules of our interpolating calculus given in Fig. 3, 4. 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.

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

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

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

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

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

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

Fig. 3. Initialization and closure rules. In the rules IPI-LEFT-L/R, $\circ \in \{\doteq, \leq\}$ denotes a relation symbol. In the rules CLOSE-*, $\exists_{LA}$ denotes existential quantification $\exists c_1, \ldots, c_n.$, where $c_1, \ldots, c_n$ are the constants 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 apply propositional and Skolemization rules to decompose A and B (the applications of rule AND-LEFT in Fig. 2). Because our propositional rules closely follow standard interpolating calculi (see [9, 4]), we only show two of these rules, namely the top-most two in Fig. 4. When splitting over L-disjunctions in the antecedent (OR-LEFT-L), it is necessary to form the disjunction of the interpolants derived in the subproofs. Analogously, R-disjunctions 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 upper part of Fig. 3 are used to turn L/R-formulae into formulae with PIs, to prepare them for later rewriting (the applications IPI in Fig. 2). 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

$$\frac{\Gamma, \lfloor\phi\rfloor_L \vdash \Delta \blacktriangleright I \quad \Gamma, \lfloor\psi\rfloor_L \vdash \Delta \blacktriangleright J}{\Gamma, \lfloor\phi \vee \psi\rfloor_L \vdash \Delta \blacktriangleright I \vee J}\ \textsc{or-left-l} \qquad \frac{\Gamma, \lfloor\phi\rfloor_D, \lfloor\psi\rfloor_D \vdash \Delta \blacktriangleright I}{\Gamma, \lfloor\phi \wedge \psi\rfloor_D \vdash \Delta \blacktriangleright I}\ \textsc{and-left}$$

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

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

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

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

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

$$\frac{\Gamma, \lfloor (\alpha \mid t+1) \vee \cdots \vee (\alpha \mid t+\alpha-1)\rfloor_D \vdash \Delta \blacktriangleright I}{\Gamma \vdash \lfloor \alpha \mid t\rfloor_D, \Delta \blacktriangleright I}\ \textsc{div-right}$$

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

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

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

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

Fig. 4. Rules for propositional connectives, equalities, divisibility, and inequalities. In AND-LEFT, we assume $D \in \{L, R\}$. In RED-LEFT and MUL-LEFT, $\circ \in \{\doteq, \leq\}$, while $\circ \in \{\doteq, \not\doteq\}$ in RED-RIGHT and MUL-RIGHT. In COL-RED-L and COL-RED-R, c is a constant that does not occur in the conclusion or 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 an arbitrary variable, and $\alpha > 0$. In FM-ELIM, $\alpha > 0$ and $\beta > 0$ are positive integers.

used in condition (iii) of the definition of valid interpolating sequents, but also in the closure rules discussed next. For 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 Sect. 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. 2). Closure rules are given in the lower part of Fig. 3. 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 judgement $\gcd(\alpha_1, \ldots, \alpha_n) \mid t$.

4.2 Rewriting Rules for Equality, Inequality and Divisibility

The arithmetic rewriting rules form a calculus to solve systems of equalities by means of Gaussian elimination and Euclid's algorithm (the middle part of Fig. 4), as well as a calculus for systems of inequalities based on Fourier-Motzkin elimination and cutting planes (the lower part of Fig. 4). Decision procedures for QFPA in terms of the corresponding non-interpolating rules have been introduced in [14, 15] and directly carry over to the interpolating case. We therefore focus on the differences between the normal and the interpolating rules.

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. 2). The RED rules alone do not form a complete calculus for integer equalities and have to be 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 [13], as well as standard procedures to transform sets of equalities (or matrices) to Hermite and Smith normalform [6, 5]. Because COL-RED-L/R only introduce local L/R-constants, it is guaranteed that the new constants do not occur in interpolants.

In contrast to [14, 15], we do not introduce a simplification rule SIMP$'$ for literals, as full simplification is not always possible in the presence of PIs. For instance, the equality $2x \doteq 0\,[a \doteq 0]$ cannot be simplified to the form $x \doteq 0\,[t^A \doteq 0]$ (as it would happen in [14, 15]) because the factor 2 does not occur in the PI. This raises a potential problem, as terms αx cannot be rewritten to 0 with the help of $2x \doteq 0$ if α is odd. As a solution, we introduce the rules MUL-LEFT/RIGHT to multiply terms with positive integers prior to rewriting.

Similar to rewriting with equalities, inequalities can be added to each other with the help of the rule FM-ELIM realizing Fourier-Motzkin variable elimination. The STRENGTHEN rule is introduced to achieve completeness over the integers (Fig. 2 shows applications of FM-ELIM and STRENGTHEN). Compared to the calculi in [14, 15], the use of STRENGTHEN in our interpolating calculus is

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

Table 1. Sequents with partial interpolants and correctness conditions (i) and (ii)

threefold: (i) STRENGTHEN can simulate the OMEGA-ELIM rule in [15], (ii) as shown in Fig. 2, 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 [14, 15]), and (iii) STRENGTHEN can simulate the law of anti-symmetry that is implemented by the rule ANTI-SYMM$'$ in [14, 15]. As STRENGTHEN is the most central rule in our calculus, we provide a detailed discussion in Sect. 5.

4.3 Properties of the Calculus

Soundness. Our interpolating calculus generates correct interpolants: whenever a sequent $\lfloor A \rfloor_L \vdash \lfloor C \rfloor_R \blacktriangleright I$ is derived, the implications $A \Rightarrow I \Rightarrow C$ are valid, and all constants in I occur in both A and C. More generally:

Lemma 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 lemma, we first need to define the semantics of PIs (although the sequent $\Gamma \vdash \Delta \blacktriangleright I$ in the lemma does not contain any PIs, they are likely to be introduced in the course of a proof). We say that a PI is *correct* if the sequents (i) and (ii) given in Table 1 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 $\Gamma_R \cup \Delta_R$. Soundness is then proven in two steps: (i) 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. (ii) 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.

As a technical difficulty, we need to annotate some rules by introducing further auxiliary formulae in the premises to ensure (i) holds. These annotations are only required for the soundness proof; soundness of the rules with auxiliary formulae directly implies soundness of the original rules.

Completeness. Vice versa, 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.

Lemma 2 (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.*

The lemma follows from the completeness of the calculi in [14, 15] by means of proof lifting: given that $\Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_R$ is valid, there is a proof of this fact in the non-interpolating calculus. This proof can be lifted by replacing each rule application with an application of the corresponding interpolating rule.

5 Strengthening and Mixed Cuts

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 [14] 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 [17]), and the general STRENGTHEN′ rule:

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

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 can first observe that two special cases are easy to handle:

$$\frac{\Gamma, t \doteq 0\,[t \doteq 0] \vdash \Delta \blacktriangleright I \quad \Gamma, t+1 \leq 0\,[t+1 \leq 0] \vdash \Delta \blacktriangleright J}{\Gamma, t \leq 0\,[t \leq 0] \vdash \Delta \blacktriangleright I \vee J}\ \textsc{strengthen-l}$$

$$\frac{\Gamma, t \doteq 0\,[0 \doteq 0] \vdash \Delta \blacktriangleright I \quad \Gamma, t+1 \leq 0\,[0 \leq 0] \vdash \Delta \blacktriangleright J}{\Gamma, t \leq 0\,[0 \leq 0] \vdash \Delta \blacktriangleright I \wedge J}\ \textsc{strengthen-r}$$

These cases are called *pure cuts* in [8], 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* [8] 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. 4 and features *three* premises, one more than the non-interpolating rule STRENGTHEN′. To understand the shape of STRENGTHEN, note that we can represent $t \leq 0$ as the sum of the inequalities $t^A \leq 0$ 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 then be simulated by applying STRENGTHEN-L to $t^A \leq 0\,[t^A \leq 0]$, and afterward 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 [17]. 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, due to two reasons: (i) the fact that STRENGTHEN in Fig. 4 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 Sect. 5.1), and (ii) because the rule SIMP′ (which has to be 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.

5.1 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 [13]. 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 tackle this growth, 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:

$$K = \bigvee_{0 \leq j \leq k} \left(I^j \wedge \bigwedge_{j \leq i < k} E_i^j \right) \tag{3}$$

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 $t + i \leq 0$, not in any other formula. We can exploit this by treating the family $(E_i^j)_{0 \leq j \leq i}$ as a *single* premise that is parameterized in the free variable j. This way, a single subproof can generate a parameterized interpolant $E_i(j)$. The parameter j can be instantiated to the values $0 \leq j \leq i$ when constructing K. Parametrized interpolants $I(j)$ can be derived similarly.

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

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

We can observe that $\alpha t + \beta + i \doteq 0$ is unsatisfiable for $0 \leq i < \alpha\lceil\frac{\beta}{\alpha}\rceil - \beta$, 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 $K = I^k \vee \bigvee_{0\leq j<k}(I^j \wedge E^j)$.

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

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

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

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

6 Experimental Evaluation

We implemented[3] the proposed interpolating calculus on top of the PRINCESS theorem prover [15], including all optimizations described in Sect. 5. To this end, we extended PRINCESS to generate proofs. The interpolation procedure then processes the proof and generates an interpolant using the rules presented in this

[3] Implementation and benchmarks: `www.philipp.ruemmer.org/iprincess.shtml`

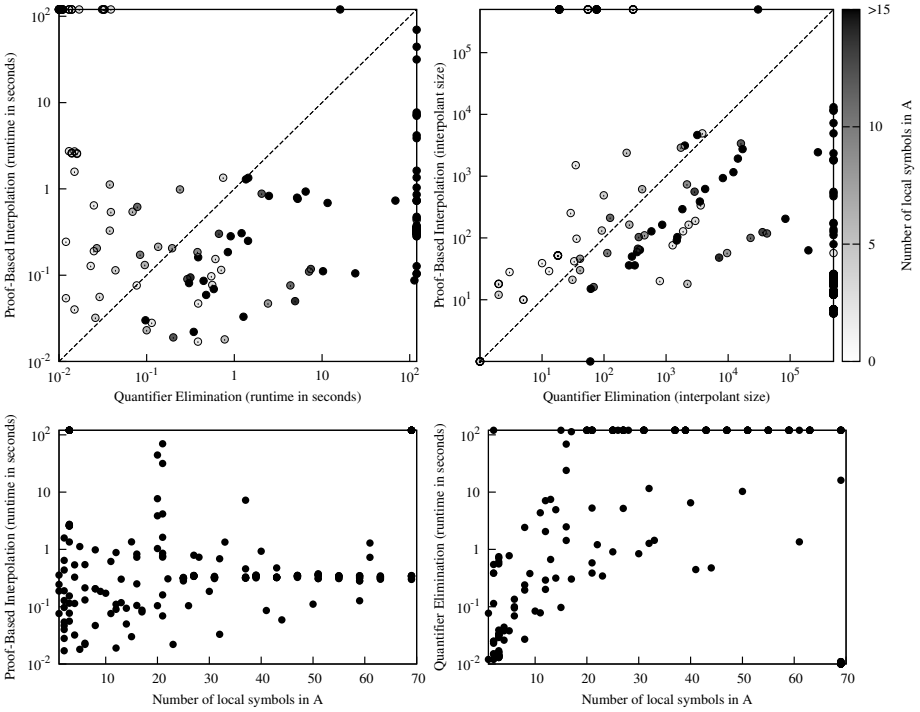

Fig. 5. Benchmarks comparing interpolant extraction with quantifier elimination

paper. The benchmarks for our experiments are derived from the SMT-LIB category QF-LIA. We evaluate them on an Intel Pentium Xeon with 3 GHz and 4 MB cache, running Linux. Because SMT-LIB benchmarks are usually conjunctions at the outermost level, we partitioned them into $A \wedge B$ by choosing the first $\frac{k}{10} \cdot n$ of the benchmark conjuncts as A, the rest as B (where n is the total number of conjuncts, and $k \in \{1, \ldots, 9\}$). Partitionings where A did not contain any local symbols (constants or propositional variables) were ignored.

Since (to the best of our knowledge) no other interpolation procedure for QFPA was available, we compared the performance of the Princess interpolation procedure with interpolation by quantifier elimination (QE), eliminating all local symbols in A. For the latter, we use the implementation of the Omega [13] test available in Princess. The results are shown in Fig. 5.

The upper left diagram compares runtimes of proof-based interpolation (PBI) with QE, with a timeout of 120s. We do not include the time to generate proofs, because in typical applications (like software model checking) many interpolants will be generated from each proof, and because QE does not decide the input formula. Considering only the cases without timeout, proving took on average about 4 times as long as the extraction of all interpolants from one proof. The diagram shows that PBI outperforms QE in 147 out of 205 cases, while QE is faster in 58 cases. QE times out for 103 of the benchmarks, PBI for 29. When

analyzing the cases where QE is faster than PBI, we observed that QE typically performs well when A only contains few local symbols, i.e., when few quantifiers need to be eliminated. We highlight cases where the number of local symbols is less than 15 by gray points in the diagrams; with an increasing number of local symbols, the performance of QE quickly degrades. To quantify this phenomenon, we measured interpolation runtimes classified by the number of local symbols in A: the two lower diagrams in Fig. 5 show that PBI is a lot less dependent on the number of such symbols than QE.

The upper right diagram compares the sizes of the interpolants (the number of operators) generated by the two techniques. In 149 cases, the interpolants obtained using PBI are smaller than those derived by QE, in 122 cases they are at least one order of magnitude smaller.

7 Related Work and Conclusions

Related work. 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 [10]; an interpolating SMT solver for the same logic has been developed by Beyer et al. [1]. Rybalchenko et al. [16] 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 [11], which is decidable by reduction to rational arithmetic. As an extension, Cimatti et al. [2] 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 [5], 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 not supported, however.

Kapur et al. [7] 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. [8] 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).

Conclusions. We have presented the first interpolating sequent calculus for quantifier-free Presburger arithmetic, permitting arbitrary combinations of linear integer equalities, inequalities, and stride predicates. Our calculus is intended to be used with a reasoning engine for sequent calculi, resulting in an interpolating decision procedure for Presburger arithmetic. We have implemented our

calculus rules in Princess and demonstrated experimentally that our method is able to generate much more succinct interpolants than quantifier elimination, which is the only other method for Presburger interpolation we are aware of.

Currently, we are working on the integration of our interpolation procedure into a software model checker based on lazy abstraction [11]. 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.

Acknowledgments. We want to thank Jerome Leroux, Vijay D'Silva, Georg Weissenbacher, and the anonymous referees for discussions and/or comments.

References

1. Beyer, D., Zufferey, D., Majumdar, R.: CSIsat: Interpolation for LA+EUF. In: Gupta, A., Malik, S. (eds.) CAV. LNCS, vol. 5123, pp. 304–308. Springer (2008)
2. Cimatti, A., Griggio, A., Sebastiani, R.: Interpolant generation for UTVPI. In: Schmidt, R.A. (ed.) CADE, LNCS, vol. 5663, pp. 167–182. Springer (2009)
3. Craig, W.: Linear reasoning. a new form of the Herbrand-Gentzen theorem. The Journal of Symbolic Logic 22(3), 250–268 (September 1957)
4. Fitting, M.C.: First-Order Logic and Automated Theorem Proving. Springer, 2nd edn. (1996)
5. Jain, H., Clarke, E., Grumberg, O.: Efficient interpolation for linear diophantine (dis)equations and linear modular equations. In: Gupta, A., Malik, S. (eds.) CAV. LNCS, vol. 5123, pp. 254–267. Springer (2008)
6. 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)
7. Kapur, D., Majumdar, R., Zarba, C.G.: Interpolation for data structures. In: SIGSOFT '06/FSE-14. pp. 105–116. ACM, New York, NY, USA (2006)
8. Lynch, C., Tang, Y.: Interpolants for linear arithmetic in SMT. In: ATVA. LNCS, vol. 5311, pp. 156–170. Springer (2008)
9. Maehara, S.: On the interpolation theorem of Craig. Sugaku 12, 235–237 (1960)
10. McMillan, K.L.: An interpolating theorem prover. Theor. Comput. Sci. 345(1) (2005)
11. McMillan, K.L.: Lazy abstraction with interpolants. In: Ball, T., Jones, R.B. (eds.) CAV. LNCS, vol. 4144, pp. 123–136. Springer (2006)
12. Presburger, M.: Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In: Comptes Rendus du Ier congrès de Mathématiciens des Pays Slaves (1929)
13. Pugh, W.: The Omega test: a fast and practical integer programming algorithm for dependence analysis. Communications of the ACM 8, 102–114 (1992)
14. Rümmer, P.: A sequent calculus for integer arithmetic with counterexample generation. In: VERIFY. CEUR (`http://ceur-ws.org/`), vol. 259 (2007)
15. Rümmer, P.: A constraint sequent calculus for first-order logic with linear integer arithmetic. In: LPAR. LNCS, vol. 5330, pp. 274–289. Springer (2008)
16. Rybalchenko, A., Sofronie-Stokkermans, V.: Constraint solving for interpolation. In: VMCAI (2007)
17. Schrijver, A.: Theory of Linear and Integer Programming. Wiley (1986)