Beyond Quantifier-Free Interpolation in Extensions of 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 emerged as an effective means of generating candidate program invariants. We present interpolation procedures for the theories of Presburger arithmetic combined with (i) uninterpreted predicates (QPA+UP), (ii) uninterpreted functions (QPA+UF) and (iii) extensional arrays (QPA+AR). We prove that none of these combinations can be effectively interpolated without the use of quantifiers, even if the input formulae are quantifier-free. We go on to identify fragments of QPA+UP and QPA+UF with restricted forms of *guarded quantification* that are closed under interpolation. Formulae in these fragments can easily be mapped to quantifier-free expressions with integer division. For QPA+AR, we formulate a sound interpolation procedure that potentially produces interpolants with unrestricted quantifiers.

1 Introduction

Given two first-order logic formulae A and C such that A implies C, written $A \Rightarrow C$, *Craig interpolation* determines a formula I such that the implications $A \Rightarrow I$ and $I \Rightarrow C$ hold, and I contains only non-logical symbols occurring in both A and C [1]. Interpolation has emerged as a practical approximation method in computing and has found many uses in formal verification, ranging from efficient image computations in SAT-based model checking, to computing candidate invariants in automated program analysis.

In software verification, interpolation is applied to formulae encoding the transition relation of a model underlying the program. In order to support a wide variety of programming language constructs, much effort has been invested in the design of algorithms that compute interpolants for formulae of various first-order theories. For example, interpolating integer arithmetic solvers have been reported for fragments such as difference-bound logic, linear equalities, and constant-divisibility predicates.

The goal of this paper is an interpolation procedure that is instrumental in analysing programs manipulating integer variables. We therefore consider the first-order theory of *quantified Presburger arithmetic* (quantified linear integer arithmetic), denoted QPA. Combined with *uninterpreted predicates* (UP) and

* This research is supported by the EPSRC project EP/G026254/1, by the EU FP7 STREP MOGENTES, and by the EU ARTEMIS CESAR project.

uninterpreted functions (UF), this allows us to encode the theory of *extensional arrays* (AR), using uninterpreted function symbols for read and write operations. Our interpolation procedure extracts an interpolant directly from a proof of $A \Rightarrow C$. Starting from a sound and complete proof system based on a sequent calculus, the proof rules are extended by labelled formulae and annotations that reduce, at the root of a closed proof, to interpolants. In earlier work, we presented a similar procedure for quantifier-free Presburger arithmetic [2].

In program verification, an interpolating theorem prover often interacts tightly with various decision procedures. It is therefore advantageous for the interpolants computed by the prover to be expressible in simple logic fragments. Unfortunately, interpolation procedures for expressive first-order fragments, such as integer arithmetic with uninterpreted predicates, often generate interpolants with *quantifiers*, which makes subsequent calls to decision procedures involving these interpolants expensive. This is not by accident. In fact, in this paper we first show that interpolation of QPA+UP in general requires the use of quantifiers, *even if the input formulae are themselves free of quantifiers.*

In order to solve this problem, we study fragments of QPA+UP that are *closed under interpolation*: fragments such that interpolants for input formulae can again be expressed in the theory. By the result above, such fragments must allow at least a limited form of quantification. Our second contribution is to show that the theory PAID+UP of Presburger arithmetic with uninterpreted predicates and a restricted form of *guarded quantifiers* indeed has the closure property. A similar fragment, PAID+UF, can be identified for the combination of Presburger arithmetic with uninterpreted functions. Moreover, by allowing *integer divisibility* (ID) predicates, the guarded quantifiers can be rewritten into quantifier-free form, facilitating further processing of the interpolants.

In summary, we present in this paper an interpolating calculus for the first-order theory of Presburger arithmetic and uninterpreted predicates, QPA+UP. We show that, for some quantifier-free input formulae, quantifiers in interpolants cannot be avoided, and suggest a restriction of QPA+UP that is closed under interpolation, yet permits quantifier-free interpolants conveniently expressible in standard logics. We extend these results to Presburger theories with uninterpreted functions and, specifically, to quantified array theory, resulting in the first sound interpolating decision procedure for Presburger arithmetic and arrays.

2 Background

2.1 Presburger Arithmetic with Predicates and Functions

Presburger arithmetic. We assume familiarity with classical first-order logic (e.g., [3]). Let x range over an infinite set X of variables, c over an infinite set C of constants, p over a set P of uninterpreted predicates with fixed arity, f over a set F of uninterpreted functions with fixed arity, and α over the set $\mathbb{Z}$ of integers. (Note the distinction between constant *symbols*, such as c, and integer *literals*, such as 42.) The syntax of terms and formulae considered in this paper is defined by the following grammar:

$$\phi \ ::= \ t \doteq 0 \mid t \leq 0 \mid \alpha \mid t \mid p(t,\ldots,t) \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \forall x.\phi \mid \exists x.\phi$$
$$t \ ::= \ \alpha \mid c \mid x \mid \alpha t + \cdots + \alpha t \mid f(t,\ldots,t)$$

The symbol t denotes terms of linear arithmetic. Divisibility atoms $\alpha \mid t$ are equivalent to formulae $\exists s.\ \alpha s - t \doteq 0$, but are required for quantifier-free interpolation. Simultaneous substitution of a vector of terms $\bar{t} = (t_1,\ldots,t_n)$ for variables $\bar{x} = (x_1,\ldots,x_n)$ in ϕ is denoted by $[\bar{x}/\bar{t}]\phi$; we assume that variable capture is avoided by renaming bound variables as necessary. For simplicity, we sometimes write $s \doteq t$ as a shorthand of $s - t \doteq 0$, and $\forall c.\phi$ as a shorthand of $\forall x.[c/x]\phi$ if c is a constant. The abbreviation *true* (*false*) stands for the equality $0 \doteq 0$ ($1 \doteq 0$), and the formula $\phi \rightarrow \psi$ abbreviates $\neg\phi \vee \psi$. Semantic notions such as structures, models, satisfiability, and validity are defined as is common over the universe $\mathbb{Z}$ of integers (e.g., [3]).

Full *quantified Presburger arithmetic* (QPA) consists of the formulae that do not contain uninterpreted predicates or functions; *(quantifier-free) Presburger arithmetic* (PA) is the quantifier-free fragment of QPA. The logic QPA+UP (QPA+UF) extends QPA to formulae with uninterpreted predicates (functions), according to the above grammar.

2.2 An Interpolating Sequent Calculus

Interpolating sequents. To extract interpolants from unsatisfiability proofs of $A \wedge B$, formulae are labelled either with the letter L ("left") to indicate that they are derived from A or with R ("right") for formulae derived from B (as in [2]). More formally, if ϕ is a formula without free variables, then $\lfloor\phi\rfloor_L$ and $\lfloor\phi\rfloor_R$ are L/R-labelled formulae. If Γ, Δ are finite sets of labelled formulae and I is an unlabelled formula without free variables, then $\Gamma \vdash \Delta \blacktriangleright I$ is an *interpolating sequent*. Similarly, if Γ, Δ are sets of unlabelled formulae without free variables, then $\Gamma \vdash \Delta$ is an (ordinary) *sequent*. An ordinary sequent is *valid* if the formula $\bigwedge\Gamma \rightarrow \bigvee\Delta$ is valid.

The semantics of interpolating sequents is defined using the projections $\Gamma_L =_{\text{def}} \{\phi \mid \lfloor\phi\rfloor_L \in \Gamma\}$ and $\Gamma_R =_{\text{def}} \{\phi \mid \lfloor\phi\rfloor_R \in \Gamma\}$, which extract the L/R-parts of a set Γ of labelled formulae. A sequent $\Gamma \vdash \Delta \blacktriangleright I$ is *valid* if (i) the sequent $\Gamma_L \vdash \Delta_L, I$ is valid, (ii) the sequent $\Gamma_R, I \vdash \Delta_R$ is valid, and (iii) the constants and uninterpreted predicate/functions in I occur in both $\Gamma_L \cup \Delta_L$ and $\Gamma_R \cup \Delta_R$. 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 $\lfloor A\rfloor_L, \lfloor B\rfloor_R \vdash \emptyset \blacktriangleright I$ captures the concept of interpolants for unsatisfiable conjunctions $A \wedge B$ common in formal verification.

Interpolating sequent calculi. An *interpolating rule* is a binary relation between a finite set of interpolating sequents, called the premises, and a sequent called the conclusion:

$$\frac{\Gamma_1 \vdash \Delta_1 \blacktriangleright I_1 \quad \cdots \quad \Gamma_n \vdash \Delta_n \blacktriangleright I_n}{\Gamma \vdash \Delta \blacktriangleright I}$$

An interpolating rule is *sound* if, for all instances whose premises $\Gamma_1 \vdash \Delta_1 \blacktriangleright I_1$, $\ldots$, $\Gamma_n \vdash \Delta_n \blacktriangleright I_n$ are valid, the conclusion $\Gamma \vdash \Delta \blacktriangleright I$ is valid, too. Fig. 1

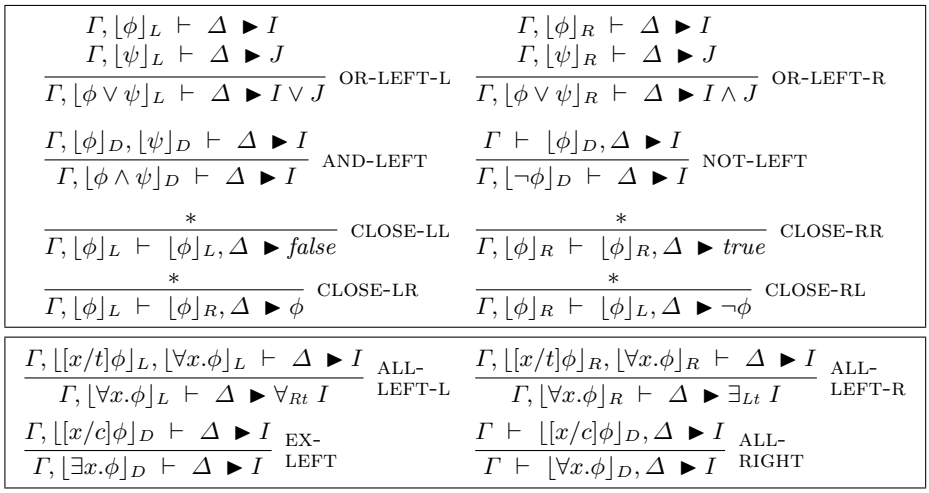


Fig. 1. The upper box presents a selection of interpolating rules for propositional logic, while the lower box shows the interpolating rules to handle quantifiers. Parameter D stands for either L or R. The quantifier $\forall_{Rt}$ denotes universal quantification over all constants occurring in t but not in $\Gamma_L \cup \Delta_L$; likewise, $\exists_{Lt}$ denotes existential quantification over all constants occurring in t but not in $\Gamma_R \cup \Delta_R$. In the rules EX-LEFT and ALL-RIGHT, c is a constant that does not occur in the conclusion.

presents a selection of interpolating rules (used throughout the paper) for predicate logic. An exhaustive list of rules is given in [2].

Interpolating proofs are trees growing upwards, in which each node is labelled with an interpolating 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.

To construct a proof for an interpolation problem, we build a proof tree starting from the root $\Gamma \vdash \Delta \blacktriangleright I$ with unknown interpolant I, i.e., I acts as a place holder. For example, to solve an interpolation problem $A \wedge B$, we start with the sequent $\lfloor A\rfloor_L, \lfloor B\rfloor_R \vdash \emptyset \blacktriangleright I$. Rules are then applied successively to decompose and simplify the sequent. Once all branches are closed, i.e., a proof is found, an interpolant can be extracted from the proof. Starting from the leaves, intermediate interpolants are computed and propagated back to the root leading to an interpolant I. An example of this procedure is given in the next section.

3 Interpolation for Uninterpreted Predicates

3.1 Presburger Arithmetic and Uninterpreted Predicates

We begin by studying the interpolation problem for Presburger arithmetic extended with uninterpreted predicates (QPA+UP), which forms a simple yet expressive base logic in which functions and arrays can be elegantly encoded. The

case of predicates is instructive, since essentially the same phenomena occur under interpolation as with uninterpreted functions.

Example 1. We illustrate the construction of an interpolating proof by deriving an interpolant for $A \Rightarrow C$, with $A = (\neg p(c) \vee p(d)) \wedge p(c)$ and $C = p(d)$. A complete interpolating proof of this implication looks as follows:

$$\dfrac{\dfrac{\dfrac{*}{\lfloor p(c)\rfloor_L \vdash \lfloor p(d)\rfloor_R, \lfloor p(c)\rfloor_L \blacktriangleright \mathit{false}}\,\textsc{close-ll}}{\lfloor \neg p(c)\rfloor_L, \lfloor p(c)\rfloor_L \vdash \lfloor p(d)\rfloor_R \blacktriangleright \mathit{false}}\,\textsc{not-left} \qquad \dfrac{*}{\lfloor p(d)\rfloor_L, \lfloor p(c)\rfloor_L \vdash \lfloor p(d)\rfloor_R \blacktriangleright p(d)}\,\textsc{close-lr}}{\lfloor \neg p(c) \vee p(d)\rfloor_L, \lfloor p(c)\rfloor_L \vdash \lfloor p(d)\rfloor_R \blacktriangleright \mathit{false} \vee p(d)}\,\textsc{or-left-l}}{\lfloor (\neg p(c) \vee p(d)) \wedge p(c)\rfloor_L \vdash \lfloor p(d)\rfloor_R \blacktriangleright \mathit{false} \vee p(d)}\,\textsc{and-left}$$

The shaded regions indicate the parts of the formula being matched against the rules in Fig. 1. The sequent $\lfloor (p(c) \vee p(d)) \wedge p(c)\rfloor_L \vdash \lfloor p(d)\rfloor_R \blacktriangleright I$ is the root of the proof, where $I = \mathit{false} \vee p(d)$ has been filled in once the proof was closed. The AND-LEFT rule propagates the L-label to the subformulae of the antecedent of the first sequent. By applying OR-LEFT-L to the disjunction $p(c) \vee p(d)$, the proof splits into two branches. The right branch can immediately be closed using CLOSE-LR. The left branch requires an application of NOT-LEFT before it can be closed with CLOSE-LL. We compute an interpolant by propagating (intermediate) interpolants from the leaves back to root of the proof. As specified by CLOSE-LR, the interpolant of the right branch is $p(d)$. On the left branch, the CLOSE-LL rule yields the interpolant *false*, which is carried through by NOT-LEFT. The rule OR-LEFT-L takes the interpolants of its two subproofs and generates $\mathit{false} \vee p(d)$. This is the final interpolant, since the last rule AND-LEFT propagates interpolants without applying modifications. □

In this example, the arguments of occurrences of uninterpreted predicates literally matched up, which need not be the case. The rules presented so far are insufficient to prove more complex theorems, such as $p(c) \wedge c \doteq d \rightarrow p(d)$, in which arithmetic and predicate calculus interact. To fully integrate uninterpreted predicates, we use an explicit *predicate consistency* axiom

$$PC_p = \quad \forall \bar{x}, \bar{y}.\ \big((p(\bar{x}) \ \wedge\ \bar{x} - \bar{y} \doteq 0) \ \rightarrow\ p(\bar{y})\big) \tag{1}$$

which can be viewed as an L- or R-labelled formula that is implicitly present in every sequent. The label L/R is chosen depending on whether p occurs in $\Gamma_L \cup \Delta_L$, in $\Gamma_R \cup \Delta_R$, or in both.

To make use of (1) in a proof, we need additional proof rules to instantiate quantifiers, which are given in the bottom part of Fig. 1. Formula (1) can be instantiated with techniques similar to the e-matching in SMT solvers [4]: it suffices to generate a ground instance of (1) by applying ALL-LEFT-L/R whenever literals $p(\bar{s})$ and $p(\bar{t})$ occur in the antecedent and succedent [5]:

$$\dfrac{\Gamma, \lfloor p(\bar{s})\rfloor_D, \lfloor (p(\bar{s}) \ \wedge\ \bar{s} - \bar{t} \doteq 0) \ \rightarrow\ p(\bar{t})\rfloor_L \vdash \lfloor p(\bar{t})\rfloor_E, \Delta \blacktriangleright I}{\Gamma, \lfloor p(\bar{s})\rfloor_D \vdash \lfloor p(\bar{t})\rfloor_E, \Delta \blacktriangleright \forall_{R\bar{s}\bar{t}}\, I}\,\textsc{all-left-l}^{+}$$

where $D, E \in \{L, R\}$ are arbitrary labels, and $\forall_{R\bar{s}\bar{t}}$ denotes universal quantification over all constants occurring in the terms $\bar{s}, \bar{t}$ but not in the set of left formulae $\big(\Gamma, \lfloor p(\bar{s})\rfloor_D\big)_L \cup \big(\Delta, \lfloor p(\bar{t})\rfloor_E\big)_L$ (like in Fig. 1). Similarly, instances of (1) labelled with R can be generated using ALL-LEFT-R. To improve efficiency, refinements can be formulated that drastically reduce the number of generated instances [6].

Correctness. The calculus consisting of the rules in Fig. 1, the arithmetic rules of [2], and axiom (1) generates correct interpolants. That is, 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 the constants and predicates in I occur in both A and C. More precisely:

Lemma 2 (Soundness). *If an interpolating QPA+UP sequent $\Gamma \vdash \Delta \blacktriangleright I$ is provable in the calculus, then it is valid.*

In particular, the sequent $\Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_R$ is valid in this case. As shown in [2], Lem. 2 holds for the calculus consisting of the arithmetic and propositional rules. It is easy to see that the additional rules presented in this paper are sound, too.

Concerning completeness, we observe that the logic of quantified Presburger arithmetic with predicates is Π^1_1-complete, which means that no complete calculi exist [7]. On the next pages, we therefore discuss how to restrict the quantification allowed in formulae to achieve completeness, while retaining the ability to extract interpolants from proofs.

3.2 Quantifiers in QPA+UP Interpolants

We first consider the quantifier-free fragment PA+UP. With the help of results in [5, 2], it is easy to see that our calculus is sound and complete for PA+UP, and can in fact be turned into a decision procedure. There is a caveat, however: although formulae in PA+UP are quantifier-free, generated interpolants may still contain quantifiers and thus lie outside of PA+UP. The source of quantifiers are the rules ALL-LEFT-L/R in Fig. 1, which can be used to instantiate L/R-labelled quantified formulae with terms containing alien symbols. Such symbols have to be eliminated from resulting interpolants through quantifiers. The following example illustrates this situation.

Example 3. Fig. 2 shows the derivation of an interpolant for the unsatisfiable conjunction $\big(2c - y \doteq 0 \wedge p(c)\big) \wedge \big(2d - y \doteq 0 \wedge \neg p(d)\big)$. After propositional reductions, we instantiate PC_p with the predicate arguments c and d, due to the occurrences of the literals $p(c)$ and $p(d)$ in the sequent. The proof can then be closed using propositional rules, complementary literals, and arithmetic reasoning [2]. The final interpolant is the formula $I = \forall x.\ (y - 2x \not\doteq 0 \vee p(x))$, in which a quantifier has been introduced via ALL-LEFT-L to eliminate the constant d. □

In fact, as we formally prove in [8], quantifier-free interpolants for the inconsistent PA+UP formulae $2c - y \doteq 0 \wedge p(c)$ and $2d - y \doteq 0 \wedge \neg p(d)$ do not exist. Abstracting from this example, we obtain:

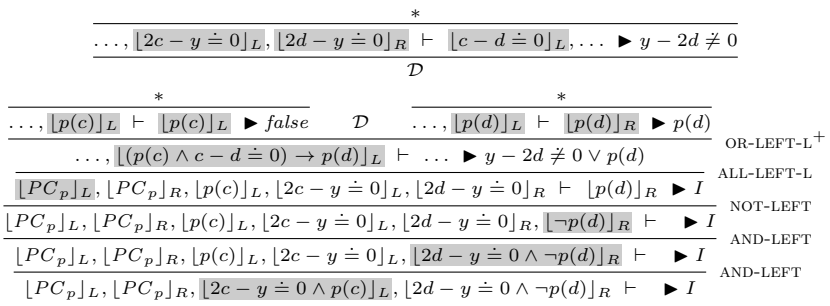


Fig. 2. Example proof involving uninterpreted predicates.

Theorem 4. *PA+UP is not closed under interpolation.*

Intuitively, Theorem 4 holds because the logic PA does not provide an integer division operator. Divisibility predicates $\alpha \mid t$ are insufficient in the presence of uninterpreted predicates, because they cannot be used within terms: no quantifier-free formula can express the statement $\forall x.\ (y - 2x \not\doteq 0 \vee p(x))$, which is equivalent to $2 \mid y \rightarrow p(\frac{y}{2})$.

Adding integer division is sufficient to close PA+UP under interpolation. More formally, we define the logic PAID ("PA with Integer Divisibility"), extending PA by *guarded* quantified expressions

$$\forall x.\ (\alpha x + t \not\doteq 0 \vee \phi), \qquad \exists x.\ (\alpha x + t \doteq 0 \wedge \phi) \tag{2}$$

where $x \in X$ ranges over variables, $\alpha \in \mathbb{N} \setminus \{0\}$ over non-zero integers, t over terms not containing x, and ϕ over PAID formulae (possibly containing x as a free variable). The logic PAID+UP is obtained by adding uninterpreted predicates to PAID. Note that the interpolant I computed in Example 3 is in PAID+UP.

It is easy to extend our interpolating calculus to a sound and complete calculus for PAID+UP; the only necessary additional rules are

$$\frac{\Gamma, \lfloor (\alpha \nmid t) \vee \exists x.\ (\alpha x + t \doteq 0 \wedge \phi) \rfloor_D \vdash \Delta \blacktriangleright I}{\Gamma, \lfloor \forall x.\ (\alpha x + t \not\doteq 0 \vee \phi) \rfloor_D \vdash \Delta \blacktriangleright I}\ \text{ALL-LEFT-GRD}$$

$$\frac{\Gamma \vdash \lfloor (\alpha \mid t) \wedge \forall x.\ (\alpha x + t \not\doteq 0 \vee \phi) \rfloor_D, \Delta \blacktriangleright I}{\Gamma \vdash \lfloor \exists x.\ (\alpha x + t \doteq 0 \wedge \phi) \rfloor_D, \Delta \blacktriangleright I}\ \text{EX-RIGHT-GRD}$$

with the side conditions that $\alpha \neq 0$, and that x does not occur in t.

Theorem 5 (Completeness). *Suppose Γ, Δ are sets of labelled PAID+UP formulae. If the sequent $\Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_R$ is valid, then there is a formula I such that (i) the sequent $\Gamma \vdash \Delta \blacktriangleright I$ is provable in the calculus of Sect. 3.1, enriched with the rules* ALL-LEFT-GRD *and* EX-RIGHT-GRD*, and (ii) I is a PAID+UP formula up to normalisation of guards to obtain expressions of the form* (2)*.*

Guard normalisation is necessary in general, because interpolants generated by proofs can take the shape $\forall \bar{x}.\, (t_1 \not\doteq 0 \vee \cdots \vee t_k \not\doteq 0 \vee \phi)$, grouping together multiple quantifiers and guards. We show in [8] that such formulae can effectively be transformed to the form (2). To prove the theorem, we first argue that sequent proofs of a certain restricted form are guaranteed to result in PAID+UP interpolants, up to normalisation of guards:

Lemma 6. *Suppose that every instantiation of the axiom* (1) *in a proof* $\mathcal{P}$ *of the PAID+UP sequent* $\Gamma \vdash \Delta \blacktriangleright I$ *has the form*

$$\dfrac{\dfrac{\dfrac{}{\ldots, \lfloor p(\bar{s})\rfloor_D \vdash \lfloor p(\bar{s})\rfloor_F, \ldots \blacktriangleright J_1}\,{}^{*} \quad \mathcal{Q} \quad \dfrac{\dfrac{}{\ldots, \lfloor p(\bar{t})\rfloor_F \vdash \lfloor p(\bar{t})\rfloor_E, \ldots \blacktriangleright J_3}\,{}^{*}}{\ldots, \lfloor (p(\bar{s}) \wedge \bar{s} - \bar{t} \doteq 0) \rightarrow p(\bar{t})\rfloor_F \vdash \ldots \blacktriangleright J_4}\ \text{OR-LEFT}^{+}}{\ldots, \lfloor p(\bar{s})\rfloor_D \vdash \lfloor p(\bar{t})\rfloor_E, \ldots \blacktriangleright J_5}\ \text{ALL-LEFT}^{+}$$

$$\dfrac{\dfrac{}{\ldots, \lfloor p(\bar{s})\rfloor_D \vdash \lfloor \bar{s} - \bar{t} \doteq 0\rfloor_F, \lfloor p(\bar{t})\rfloor_E, \ldots \blacktriangleright J_2}}{\mathcal{Q}}$$

where (i) $D, E \in \{L, R\}$ *and* $F \in \{D, E\}$ *are arbitrary labels, (ii) the proof* $\mathcal{Q}$ *only uses the rules* RED-RIGHT, MUL-RIGHT, IPI-RIGHT, AND-RIGHT-L, *and* CLOSE-EQ-RIGHT *applied to an equality derived from* $\bar{s} - \bar{t} \doteq 0$ *(see [2] for definitions of the rules), and (iii)* ALL-LEFT *and* EX-RIGHT *are not applied in any other places in* $\mathcal{P}$*. Then* I *is a PAID+UP formula up to normalisation of guards.*

A proof of this lemma is contained in [8]. Intuitively, the conditions in the lemma enable the application of (1) to atoms $p(\bar{s})$ and $p(\bar{t})$ only if the equations present in a sequent entail that the arguments $\bar{s}$ and $\bar{t}$ match up. There are various ways of relaxing this restriction: most importantly, the applications of axiom (1) only has to be constrained when unifying literals $\lfloor p(\bar{s})\rfloor_D$ and $\lfloor p(\bar{t})\rfloor_E$ with *distinct* labels $D \neq E$. Applications of the axiom to literals with the same label are uncritical, because they never introduce quantifiers in interpolants. In fact, practical experience with our theorem prover PRINCESS shows that generated interpolants are often naturally in the PAID+UP fragment, even when not imposing any restrictions on the proof generation process.

The second ingredient in proving the completeness theorem Thm. 5 is to show that the calculus with the restrictions imposed in Lem. 6 is still complete. We describe a proof procedure abiding by these restrictions in [8]. As a corollary of the completeness, we obtain:

Corollary 7. *PAID+UP is closed under interpolation.*

Despite this closure property, some proofs may result in interpolants outside PAID+UP, by applying "wrong" rules in the sub-proof $\mathcal{Q}$ of Lem. 6:

Example 8. Starting from PAID+UP input formulae, the following proof generates the interpolant $\forall c.\, p(c)$, which is not equivalent to any PAID+UP formula:

$$\dfrac{\dfrac{}{\lfloor p(0)\rfloor_L \vdash \lfloor p(0)\rfloor_L \blacktriangleright \mathit{false}}\,{}^{*} \quad \dfrac{}{\lfloor q\rfloor_L \vdash \lfloor c \doteq 0\rfloor_L, \lfloor q\rfloor_L \blacktriangleright \mathit{false}}\,{}^{*} \quad \dfrac{}{\lfloor p(c)\rfloor_L \vdash \lfloor p(c)\rfloor_R \blacktriangleright p(c)}\,{}^{*}}{\dfrac{\ldots, \lfloor p(0)\rfloor_L, \lfloor q\rfloor_L, \lfloor (p(0) \wedge c \doteq 0) \rightarrow p(c)\rfloor_L \vdash \lfloor c\rfloor_R, \lfloor q\rfloor_L \blacktriangleright p(c)}{\lfloor PC_p\rfloor_L, \lfloor PC_p\rfloor_R, \lfloor p(0)\rfloor_L, \lfloor q\rfloor_L \vdash \lfloor p(c)\rfloor_R, \lfloor q\rfloor_L \blacktriangleright \forall c.\, p(c)}\ \text{ALL-LEFT-L}}$$

The first step in the proof is to instantiate axiom (1), in an attempt to unify the formula $\lfloor p(0) \rfloor_L$ and $\lfloor p(c) \rfloor_R$; this instantiation later introduces the unguarded quantifier $\forall c$ in the interpolant. The proof violates the conditions in Lem. 6, because the middle sub-proof is closed using the atoms $\lfloor q \rfloor_L$ instead of the equation $\lfloor c \doteq 0 \rfloor_L$. A correct PAID+UP interpolant for this example is *false*. □

PAID and integer division. Despite the presence of guarded quantifiers, PAID is close to simple quantifier-free assertion languages found in programming languages like Java or C, making PAID expressions convenient to pass on to decision procedures. Specifically, the following equivalences hold:

$$\forall x.\ (\alpha x + t \not\doteq 0 \vee \phi) \ \equiv\ (\alpha \nmid t) \vee [x/(t \div \alpha)]\phi, \qquad (\alpha \mid t) \ \equiv\ \alpha(t \div \alpha) \doteq t$$

where $\div$ denotes integer division. Vice versa, an expression $c \doteq t \div \alpha$ can be encoded in PAID using axioms like $\alpha c \leq t \ \wedge\ (t < \alpha c + \alpha \vee t < \alpha c - \alpha)$.

4 Interpolation for Uninterpreted Functions

4.1 A Relational Encoding of Uninterpreted Functions

For practical verification and interpolation problems, uninterpreted functions are more common and often more important than uninterpreted predicates. In the context of interpolation, functions share many properties with predicates; in particular, the quantifier-free fragment PA+UF is again not closed under interpolation, in analogy to Theorem 4.

Similar to the previous section, the interpolation property can be restored by adding means of integer division. To this end, we define the logic PAID+UF like PAID, but allowing arbitrary occurrences of uninterpreted functions in terms. For reasoning and interpolation purposes, we represent functions via an encoding into uninterpreted predicates. The resulting calculus strongly resembles the congruence closure approach used in SMT solvers (e.g., [4]). To formalise the encoding, we introduce a further logic, PAID+UF$_p$. Recall that P and F denote the vocabularies of uninterpreted predicates and functions. We assume that a fresh $(n+1)$-ary uninterpreted predicate $f_p \in P$ exists for every n-ary uninterpreted function $f \in F$. The logic PAID+UF$_p$ is then derived from PAID by incorporating occurrences of predicates f_p of the following form:

$$\exists x.\ \big(f_p(t_1, \ldots, t_n, x) \wedge \phi\big) \tag{3}$$

where $x \in X$ ranges over variables, $t_1, \ldots, t_n$ over terms that do not contain x, and ϕ over PAID+UF$_p$ formulae (possibly containing x). In order to avoid universal quantifiers, we do not allow expressions (3) underneath negations.

Formulae in PAID+UF can uniformly be mapped to PAID+UF$_p$ by rewriting:

$$\phi[f(t_1, \ldots, t_n)] \quad \rightsquigarrow \quad \exists x.\ (f_p(t_1, \ldots, t_n, x) \wedge \phi[x]) \tag{4}$$

provided that the terms $t_1, \ldots, t_n$ do not contain variables bound in ϕ. To stay within PAID+UF$_p$, application of the rule underneath negations has to

be avoided, which can be done by transformation to negation normal form. We write ϕ_{Rel} for the function-free PAID+UF$_p$ formula derived from a PAID+UF formula ϕ by exhaustive application of (4). Vice versa, ϕ can be obtained from ϕ_{Rel} by applying (4) in the opposite direction. Assuming *functional consistency*, the formulae ϕ and ϕ_{Rel} are satisfiability-equivalent:

Lemma 9. *Let FC_f denote the functional consistency axiom:*[3]

$$FC_f = \quad \forall \bar{x}_1, \bar{x}_2, y_1, y_2.\ \big((f_p(\bar{x}_1, y_1) \wedge f_p(\bar{x}_2, y_2) \wedge \bar{x}_1 \doteq \bar{x}_2) \ \rightarrow\ y_1 \doteq y_2\big) \quad (5)$$

A PAID+UF formula ϕ is satisfiable exactly if $\phi_{Rel} \wedge \bigwedge_{f \in F} FC_f$ is satisfiable.

By the lemma, it is sufficient to construct a proof of $\neg(\phi_{Rel} \wedge \bigwedge_{f \in F} FC_f)$ in order to show that ϕ is unsatisfiable.[4] The axioms FC_f can be handled by ground instantiation, just like the predicate consistency axiom (1): whenever atoms $f_p(\bar{s}_1, t_1)$ and $f_p(\bar{s}_2, t_2)$ occur in the antecedent of a sequent, an instance of FC_f can be generated using the rules ALL-LEFT-L/R and the substitution $[\bar{x}_1/\bar{s}_1, \bar{x}_2/\bar{s}_2, y_1/t_1, y_2/t_2]$. This form of instantiation is sufficient, because predicates f_p only occur in positive positions in ϕ_{Rel}, and therefore only turn up in antecedents. As before, the number of required instances can be kept under control by formulating suitable refinements [6].

4.2 Interpolation for PAID+UF

PAID+UF conjunctions $A \wedge B$ can be interpolated by constructing a proof of

$$\lfloor A_{Rel} \rfloor_L, \lfloor B_{Rel} \rfloor_R, \{\lfloor FC_f \rfloor_L\}_{f \in F_A}, \{\lfloor FC_f \rfloor_R\}_{f \in F_B} \ \vdash\ \emptyset\ \blacktriangleright I \qquad (6)$$

where F_A/F_B are the uninterpreted functions occurring in A/B. Due to the soundness of the calculus, the existence of a proof guarantees that I is an interpolant. Vice versa, a completeness result corresponding to Thm. 5 also holds for PAID+UF$_p$. Because PAID+UF$_p$ interpolants can be translated back to PAID+UF by virtue of (4), we also have a closure result:

Theorem 10. *The logic PAID+UF is closed under interpolation.*

Example 11. We consider the PAID+UF interpolation problem $A \wedge B$ with

$$A = \ b \doteq f(2) \wedge f(a+1) \doteq c \wedge d \doteq 1, \qquad B = \ a \doteq 1 \wedge f(b) \doteq f(c) + d\ .$$

The corresponding PAID+UF$_p$ formulae are:

$$A_{Rel} = \ \exists x_1.\ \big(f_p(2, x_1) \wedge \exists x_2.\ \big(f_p(a+1, x_2) \wedge b \doteq x_1 \wedge x_2 \doteq c \wedge d \doteq 1\big)\big)$$
$$B_{Rel} = \ \exists y_1.\ \big(f_p(b, y_1) \wedge \exists y_2.\ \big(f_p(c, y_2) \wedge a \doteq 1 \wedge y_1 \doteq y_2 + d\big)\big)\ .$$

[3] Axiom (5) can also be formulated as $\forall \bar{x}_1, y_1, y_2.\ (f_p(x, y_1) \wedge f_p(\bar{x}, y_2) \rightarrow y_1 \doteq y_2)$, assuming the predicate consistency axiom (1). We chose (5) to avoid having to consider the auxiliary axiom (1) at this point, which simplifies presentation.

[4] Note that this formulation fails to work if arbitrary quantifiers are allowed in ϕ; this case would require axioms for totality of functions as well.

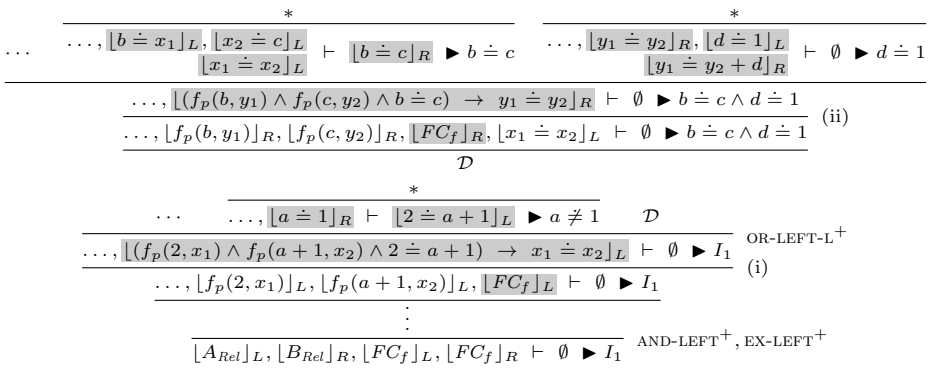
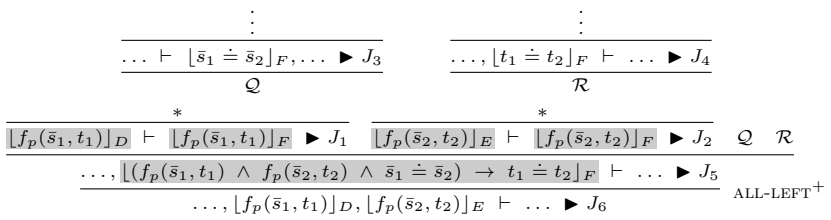


Fig. 3. Interpolating proof of Example 11. Parts of the proof concerned with arithmetic reasoning or application of the CLOSE-* rules are not shown.

The unsatisfiability of $A_{Rel} \wedge B_{Rel}$ is proven in Fig. 3, requiring two applications of FC_f: (i) for the pair $f(2), f(a+1)$, and (ii) for $f(b), f(c)$. The resulting interpolant is $I_1 = \ a \neq 1 \vee (b \doteq c \wedge d \doteq 1)$ and contains a disjunction due to splitting over an L-formula (i), and a conjunction due to (ii). □

As in Lem. 6, a sufficient condition for PAID+UF$_p$ interpolants can be given by restricting applications of the functional consistency axiom:

Lemma 12. *Suppose that every instantiation of an axiom FC_f in a proof $\mathcal{P}$ of* (6) *has the form*



where (i) $D, E \in \{L, R\}$ and $F \in \{D, E\}$ are arbitrary labels, (ii) $R \in \{D, E\}$ implies $F = R$, (iii) the proof $\mathcal{Q}$ only uses the rules RED-RIGHT, MUL-RIGHT, IPI-RIGHT, AND-RIGHT-L, *and* CLOSE-EQ-RIGHT *applied to an equality derived from $\bar{s}_1 \doteq \bar{s}_2$ (see [2]), (iv)* ALL-LEFT *and* EX-RIGHT *are not applied in any other places in $\mathcal{P}$. Then I is a PAID+UF$_p$ formula up to normalisation of guards.*

Proofs of this shape closely correspond to the reasoning of congruence closure procedures (e.g., [4]): two terms/nodes $f(\bar{s}_1)$ and $f(\bar{s}_2)$ are collapsed only once the equations $\bar{s}_1 \doteq \bar{s}_2$ have been derived. Congruence closure can therefore be used to efficiently generate proofs satisfying the conditions of the lemma (abstracting from the additional reasoning necessary to handle the integers).

As in Sect. 3.2, it is also possible to relax the conditions of the lemma; in particular, there is no need to restrict FC_f applications with $D = E$. The

resulting interpolation procedure is very flexible, in the sense that many different interpolants can be generated from essentially the same proof. Reordering FC_f applications, for instance, changes the propositional structure of interpolants:

Example 13. In Example 11, the interpolant $I_1 = \ a \not\doteq 1 \vee (b \doteq c \wedge d \doteq 1)$ is derived using two FC_f applications (i) and (ii). Reordering the applications, so as to perform (ii) before (i), yields the interpolant $I_2 = \ (a \not\doteq 1 \vee b \doteq c) \wedge d \doteq 1$. □

4.3 Interpolation for the Theory of Extensional Arrays

The first-order theory of arrays [9] is typically encoded using uninterpreted function symbols *select* and *store* by means of the following axioms:

$$\forall x, y, z.\ \mathit{select}(\mathit{store}(x, y, z), y) \doteq z \tag{7}$$

$$\forall x, y_1, y_2, z.\ \big(y_1 \doteq y_2 \vee \mathit{select}(\mathit{store}(x, y_1, z), y_2) \doteq \mathit{select}(x, y_2)\big) \tag{8}$$

Intuitively, $\mathit{select}(x, y)$ retrieves the element of array x stored at position y, while $\mathit{store}(x, y, z)$ denotes the array that is identical to x, except that position y stores value z. The *extensional* theory of arrays additionally supports equalities between arrays and is encoded using the following axiom:

$$\forall x_1, x_2.\ (x_1 \doteq x_2 \leftrightarrow (\forall y.\ \mathit{select}(x_1, y) \doteq \mathit{select}(x_2, y))) \tag{9}$$

The quantifier-free theory of arrays is again not closed under interpolation, even without arithmetic, as was already noted in [10, 11]. A classical example is given by the following inconsistent formulae:

$$\begin{aligned} A = &\ \ M' \doteq \mathit{store}(M, a, d) \\ B = &\ \ b \not\doteq c \ \wedge\ \mathit{select}(M', b) \not\doteq \mathit{select}(M, b) \ \wedge\ \mathit{select}(M', c) \not\doteq \mathit{select}(M, c)\,, \end{aligned}$$

which only permit quantified interpolants, of the form

$$\forall y_1, y_2.\ \big(y_1 \doteq y_2 \ \vee\ \mathit{select}(M, y_1) \doteq \mathit{select}(M', y_1) \ \vee\ \mathit{select}(M, y_2) \doteq \mathit{select}(M', y_2)\big).$$

Naturally, combining array theory with quantifier-free Presburger arithmetic only exacerbates the problem. As we have shown in previous sections, extending PA+UP by guarded integer divisibility predicates results in a theory that is closed under interpolation. We can extend this solution to the theory of arrays, but still only obtain closure under interpolation for small fragments of the logic (like for formulae that do not contain the *store* symbol). The resulting interpolation procedure is similar in flavour to the procedures in [12, 13] and works by explicit instantiation of the array axioms. As in Sect. 3, axioms are handled lazily using the rules ALL-LEFT-L/R, which introduce quantifiers in interpolants as needed.

Array interpolation via relational encoding. To reduce array expressions to expressions involving uninterpreted predicates, we use the same relational encoding as in Sect. 4. We first lift the axioms (7), (8), and (9) to the relational encoding:

$$AR_1 = \forall x_1, x_2, y, z_1, z_2.\ \big(store_p(x_1, y, z_1, x_2) \wedge select_p(x_2, y, z_2)\ \rightarrow\ z_1 \doteq z_2\big)$$

$$AR_2 = \forall x_1, x_2, y_1, y_2, z, z_1, z_2.\ \begin{pmatrix} store_p(x_1, y_1, z, x_2) \\ \wedge\ select_p(x_1, y_2, z_1) \\ \wedge\ select_p(x_2, y_2, z_2) \end{pmatrix} \rightarrow\ y_1 \doteq y_2 \vee z_1 \doteq z_2$$

$$AR_3 = \forall x_1, x_2.\ \begin{pmatrix} \forall y, z_1, z_2.\ (select_p(x_1, y, z_1) \wedge select_p(x_2, y, z_2) \rightarrow z_1 \doteq z_2) \\ \rightarrow x_1 \doteq x_2 \end{pmatrix}$$

As in the previous sections, these axioms can be used in proofs by ground instantiation based on literals that occur in antecedents of sequents; in the case of AR_3, it is also necessary to perform instantiation based on equations occurring in the succedent. This yields an interpolating (though incomplete) calculus for the full logic QPA+AR, and an interpolating decision procedure for the combined theory PAID+AR of Presburger arithmetic with integer division and arrays. Interpolants expressed via the relational encodings of the functions *select* and *store* can be translated into interpolants over array expressions via re-substitution rules.

Array properties. The *array property fragment*, introduced by Bradley et al. [14], comprises Presburger arithmetic and the theory of extensional arrays parameterised by suitable element theories. In array property formulae, integer variables may be quantified universally, provided that the matrix of the resulting quantified formula is *guarded* by a Boolean combination of equalities and non-strict inequalities. Using such formulae, one can express properties like equality and sortedness of arrays, as they commonly occur in formulae extracted from programs. Despite its expressiveness, satisfiability for this fragment was shown to be decidable by providing an effective decision procedure [14].

Although Bradley et al. did not consider interpolation for the theory of array properties, we observe that the decision procedure given in [14] can easily be made interpolating using the calculus for QPA+AR provided in this paper. The decision procedure proceeds by reducing, in a sequence of 5 steps, array property formulae to formulae in the combined theory of Presburger arithmetic with uninterpreted functions and the element theories. These 5 steps essentially correspond to instantiation of the array axioms and of quantified parts of the input formulae, which can be implemented using the interpolating rules provided in Fig. 1. The final step is a call to an interpolating decision procedure for Presburger arithmetic and uninterpreted functions combined with suitable element theories; we have presented such a procedure in this paper.

We remark that the array property fragment is not subsumed by the restriction of QPA+AR to Presburger arithmetic and array theory with guarded quantification as allowed in PAID+UF.

5 Related Work and Conclusion

Related work. For work on interpolation in pure quantifier-free Presburger arithmetic, see [2]. Yorsh et al. [15] present a combination method to generate interpolants using interpolation procedures for individual theories. To be applicable, the method requires individual theories to be *equality interpolating*; this is neither the case for Presburger arithmetic nor for arrays. To the best of our knowledge, it is unknown whether quantifier-free Presburger arithmetic with the integer division operator $\div$ is equality interpolating.

Interpolation procedures for uninterpreted functions are given by McMillan [10] and Fuchs et al. [16]. The former approach uses an interpolating calculus with rules for transitivity, congruence, etc.; the latter is based on congruence closure algorithms. Our calculus in Sect. 4 has similarities with [16], but is more flexible concerning the order in which congruence rules are applied. A more systematic comparison is planned as future work, including estimating the cost of interpolating uninterpreted functions via a reduction to predicates, rather than via some direct procedure. The papers [10, 16] do not consider the combination with full Presburger arithmetic.

Kapur et al. [11] present an interpolation method for arrays that works by reduction to the theory of uninterpreted functions. To some degree, the interpolation procedure of Sect. 4.3 can be considered as a lazy version of the procedure in [11], performing the reduction to uninterpreted functions only on demand.

In [12], Jhala et al. define a *split prover* that computes quantifier-free interpolants in a fragment of the theory of arrays, among others. The main objective of [12] is to derive interpolants in restricted languages, which makes it possible to guarantee convergence and a certain form of completeness in model checking. While our procedure is more general in that the full combined theory of PA with arrays can be handled, we consider it as important future work to integrate techniques to restrict interpolant languages into our procedure.

McMillan provides a complete procedure to generate (potentially) quantified interpolants for the full theory of arrays [13] by means of explicit array axioms. Our interpolation method resembles McMillan's in that explicit array axioms are given to a theorem prover, but our procedure is also complete in combination with Presburger arithmetic.

Bradley et al. introduce the concept of constrained universal quantification in array theory [14], which essentially allows a single universal array index quantifier, possibly restricted to an index subrange, e.g. all indices in some range $[l, u]$. Unlike full quantified array theory, satisfiability is decidable in Bradley's fragment; interpolation is not considered in this work. We have discussed the relationship of this fragment to QPA+AR in Section 4.3.

Conclusion. We have presented interpolating calculi for the theories of Presburger arithmetic combined with uninterpreted predicates (QPA+UP), uninterpreted functions (QPA+UF), and extensional arrays (QPA+AR). We have demonstrated that these extensions require the use of quantifiers in interpolants.

Adding notions of *guarded quantification*, we therefore identified fragments of the full first-order theories that are closed under interpolation, yet are expressible in assertion languages present in standard programming languages.

As future work, we plan to extend our results to interpolating SMT solvers, particularly aiming at procedures that can be used in model checkers based on the *lazy abstraction with interpolants* paradigm. On the theoretical side, we will study the relationship between the logics discussed in this paper, and architectures for combining interpolating procedures, e.g., [15]. We also plan to investigate, possibly along the lines of [17], how our interpolation procedure for uninterpreted functions relates to existing methods [10, 16], and how it affects the strength of computed interpolants. Finally, we plan to investigate a combination of our calculus with the Split-Prover approach in [12].

References

1. Craig, W.: Linear reasoning. A new form of the Herbrand-Gentzen theorem. The Journal of Symbolic Logic **22** (1957) 250–268
2. Brillout, A., Kroening, D., Rümmer, P., Wahl, T.: An interpolating sequent calculus for quantifier-free Presburger arithmetic. In: IJCAR. LNCS, Springer (2010)
3. Fitting, M.C.: First-Order Logic and Automated Theorem Proving. 2nd edn. Springer (1996)
4. Detlefs, D., Nelson, G., Saxe, J.B.: Simplify: A theorem prover for program checking. Journal of the ACM **52** (2005) 365–473
5. Rümmer, P.: A constraint sequent calculus for first-order logic with linear integer arithmetic. In: LPAR. (2008)
6. Rümmer, P.: Calculi for Program Incorrectness and Arithmetic. PhD thesis, University of Gothenburg (2008)
7. Halpern, J.Y.: Presburger arithmetic with unary predicates is Π_1^1 complete. Journal of Symbolic Logic **56** (1991)
8. Brillout, A., Kroening, D., Rümmer, P., Wahl, T.: Beyond quantifier-free interpolation in extensions of Presburger arithmetic (extended Technical Report). Technical report, CoRR abs/1011.1036 (2010)
9. McCarthy, J.: Towards a mathematical science of computation. In: Information Processing 1962: Proceedings IFIP Congress 62, Amsterdam, North Holland (1963)
10. McMillan, K.L.: An interpolating theorem prover. Theor. Comput. Sci. **345** (2005)
11. Kapur, D., Majumdar, R., Zarba, C.G.: Interpolation for data structures. In: SIGSOFT '06/FSE-14, New York, NY, USA, ACM (2006) 105–116
12. Jhala, R., McMillan, K.L.: A practical and complete approach to predicate refinement. In: TACAS. LNCS, Springer (2006)
13. McMillan, K.L.: Quantified invariant generation using an interpolating saturation prover. In: TACAS. (2008)
14. Bradley, A.R., Manna, Z., Sipma, H.B.: What's decidable about arrays? In: VMCAI. (2006) 427–442
15. Yorsh, G., Musuvathi, M.: A combination method for generating interpolants. In: CADE. (2005)
16. Fuchs, A., Goel, A., Grundy, J., Krstić, S., Tinelli, C.: Ground interpolation for the theory of equality. In: TACAS. LNCS, Springer (2009)
17. D'Silva, V., Purandare, M., Weissenbacher, G., Kroening, D.: Interpolant strength. In: VMCAI. LNCS, Springer (2010)