Word-level Quantifier Elimination and Interpolation for Modular Arithmetic Constraints

Bishoksan Kafle, Graeme Gange, Peter Schachte, Harald Søndergaard and Peter J. Stuckey

Department of Computing and Information Systems,
The University of Melbourne, Victoria 3010, Australia
{bishoksank,gkgange,schachte,harald,pstuckey}@unimelb.edu.au

Abstract

Automated software verification methods frequently rely on two key operations: *quantifier elimination* to eliminate irrelevant program states, and *interpolant generation* to generalize failed executions. These operations are well understood for linear arithmetic over rationals and integers. However, most verification problems need to reason about fixed precision integers and modular (two's complement) arithmetic. Directly applying algorithms for integer arithmetic to the resulting constraint systems is unsound; thus the problem must be transformed into either pure integer constraints or bit-vectors, which in either case obscures the structure of the problem. In this paper, we describe procedures for *quantifier elimination* and *interpolant generation* over modular linear arithmetic. The proposed algorithms deal gracefully with wrapping effects, while exploiting as much of the linear structure as possible. Preliminary evaluation of the proposed algorithms shows promising results.

Keywords and phrases Modular arithmetic, Quantifier Elimination, Interpolation, Verification, Benders decomposition.

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

There has been a great deal of success in applying constraint-based reasoning for automated program verification and (unbounded) model checking (e.g. [28, 30, 23]). These approaches model both the program (or transition system) and verification conditions as systems of constraints, and often rely on two critical operations: *quantifier elimination* (QE) [22, 23], and *interpolant generation* (IG) [28]. In this context, QE allows us to eliminate irrelevant intermediate program states from consideration whereas IG allows us to generalise failed executions. However, in most analysis and verification tasks we are dealing with modular arithmetic rather than classical integer arithmetic, and the generated constraints express an underlying logic of fixed-width integers and machine arithmetic based on a two's complement representation. Modular Linear Integer Arithmetic (MLIA) is an important variant of linear integer arithmetic (LIA) where the integers are interpreted modulo m, for some $m \in \mathbb{N}$. QE and IG are well studied for the theories of reals, rationals and integers, but MLIA has received much less attention.

Given a quantified formula F in some theory, *quantifier elimination* produces a quantifier-free formula G equivalent to F. Given two formulas A and B where $A \wedge B$ is unsatisfiable, *interpolant generation* produces a formula I only involving variables shared by A and B such that $A \rightarrow I$ and $I \wedge B$ is unsatisfiable.

▶ **Example 1.** Consider the conjunctive formula

$$F = [\exists x \ (x \geq y \wedge x + 1 = y)]_m$$

(We use $[F]_m$ to indicate that formula F is interpreted in $\mathbb{Z}_m$, the ring of integers modulo m.) It tests an increment operation for overflow. Its only model is $\{x = m-1, y = 0\}$, irrespective of the bit-width of x and y.

Clearly, F is equivalent to the quantifier-free formula $y = 0$. Nevertheless, for bit-blasting solvers, examples like this are increasingly challenging as the bit-width grows. Similarly, the direct LIA encoding produces a constraint system that is not easily amenable to QE. □

We can also use QE to do IG. However, this generates the strongest possible interpolant, and often for invariant generation, we would like to get an inductive invariant which is usually weaker than the strongest one. QE is also often much more expensive than IG.

▶ **Example 2.** Consider the formulas A and B over unsigned integers.

$$A \equiv [y = x + 9 \wedge x \leq y \wedge y \leq z]_{16}$$
$$B \equiv [z = y + 9]_{16}$$

This formula $A \wedge B$ is satisfiable over mathematical integers with $\{x = 0, y = 9, z = 18\}$ but unsatisfiable over 4-bit arithmetic. Therefore, there exist interpolants between A and B; the formula $9 \leq y \wedge y \leq z$ is one of them. We show later how we derive this using our approach. □

The-state-the-art QE/IG approaches for MLIA constraints use two different encodings and appropriate QE/IG procedures for the encodings. The first and the most common approach encodes MLIA constraints as bit-vector (BV) constraints and uses QE/IG methods for BV constraints, which often involves bit-blasting. This bit-blasting followed by bit-level QE/IG often scales poorly in practice. The second approach translates MLIA constraints into equivalent constraints in linear integer arithmetic and uses a QE/IG procedure for linear integer arithmetic. However the resulting LIA constraints are often hard to solve with existing LIA solvers.

In this paper, we develop an optimistic approach to QE and IG for MLIA which also uses an LIA encoding. Rather than applying QE/IG to the resulting LIA system, it alternates between logic-based Benders decomposition (to eliminate introduced variables during translation) and QE/IG for linear integer arithmetic on the residual constraints. The key contributions of the paper are:

1. We present novel (bit-precise and *bit-blasting* free) algorithms for QE and IG of MLIA constraints (Section 3). These algorithms are compositional and are achieved by using Benders decomposition approach.
2. We present encouraging experimental results obtained by our prototype implementation, compared with the state-of-the-art solvers Z3 and MathSAT (Section 4).

2 Preliminaries

We consider a logic of quantified MLIA constraints consisting of connectives $\exists, \vee, \wedge, \neg$, and primitive constraints $\sum_i c_i x_i \bowtie \sum_j k_j y_j$, with $\bowtie \in \{\leq, <\}$, where variables and expressions are interpreted modulo a constant m.

The $\neq$ is encoded as a disjunction of strict inequalities. A universally quantified formula $\forall x\ (F)$ can be expressed as $\neg(\exists x\ (\neg F))$. We abbreviate $\exists x_1(\exists x_2 \ldots (\exists x_n\ (F)))$ to $\exists x_1, x_2, \ldots, x_n\ (F)$ (or sometimes simply to $\exists X\ F$). For an expression e, let $vars(e)$ denote the set of variables appearing in e. We extend this to formulas in the obvious way. We will sometimes treat a conjunctive formula $\bigwedge_{c \in S} c$ as a set S of constraints. A valuation

$\mu : V \mapsto \mathbb{Z}$ is a mapping from variables to values. We extend μ to expressions and formulas in the natural manner.

Given two formulas A and B such that $A \wedge B \models \bot$, a *Craig interpolant* [11] (or just interpolant) is a formula I over $vars(A) \cap vars(B)$ such that $A \rightarrow I$, and $I \wedge B \models \bot$. It is frequently used to compute a generalization of the cause of unsatisfiability. In any theory equipped with projection, the formula $\exists(vars(A) \setminus vars(B))\ (A)$ is an interpolant. However, the projection operation is frequently quite expensive, and the resulting formula is frequently too specific to be a useful generalization. More efficient interpolation algorithms have been developed for a range of theories, including SAT [33], SMT for bit-vectors [15] and linear arithmetic over $\mathbb{Q}$ [27] and $\mathbb{Z}$ [**?**]. These procedures follow a common pattern: run a decision procedure on $A \wedge B$ to prove unsatisfiability, then reconstruct a new formula by inspecting the structure of the refutation.

Benders decomposition

Informally, Benders decomposition [2] is an approach to solving mathematical programming problems, frequently applied where a problem consists of independent subproblems connected via a small set of variables.

Consider the mixed integer programming (MIP) problem

$$P \equiv \min f(x) \text{ subject to } A(x) \geq b \wedge B(x, y) \geq c \wedge x \in D_x \wedge y \in D_y.$$

P is an optimisation problem, minimising $f(x)$ over variables x, y, consisting of constraints A over x, and additional constraints B over x, y. Rather than solving P directly, we may instead fix x to some optimal value $\tilde{x}$, then check whether there exists some $\tilde{y}$ satisfying $B(\tilde{x}, \tilde{y}) \geq c$. If $\tilde{y}$ exists, $(\tilde{x}, \tilde{y})$ is an optimal solution of P.

We thus construct a *relaxed master problem*

$$P^\sharp \equiv \min f(x) \text{ subject to } A(x) \geq b \wedge B^\sharp(x) \geq c \wedge x \in D_x.$$

The *projection constraint* $B^\sharp(x) \geq c$ is a relaxation of $\exists y.\ B(x, y) \geq c$; this may be any constraint (or set of constraints) satisfying $B^\sharp(x) \geq \mathbf{sup}_{y \in D_y} B(x, y)$. Being a relaxation, $B^\sharp(x)$ may allow invalid solutions. Thus we introduce a satisfiability subproblem $Q(\tilde{x}) \equiv B(\tilde{x}, y) \geq c \wedge y \in D_y$ which checks feasibility. Solving $P^\sharp$ yields a candidate optimum $\tilde{x}$. If $Q(\tilde{x})$ is satisfiable, we have found an optimum. If not, we extract from Q a new constraint $a'x \geq b'$, excluding $\tilde{x}$, which we add to $P^\sharp$ (in effect strengthening $B^\sharp$). This procedure is repeated until either an assignment is found, or $P^\sharp$ is proven unsatisfiable.

The classical form of Benders decomposition requires the subproblems to be linear programs (i.e., over $\mathbb{Q}$). However, in *logic-based Benders decomposition* [19], the subproblem may be an arbitrary decision problem, and the feasibility cut is extracted from the unsatisfiability proof of Q.

From MLIA to equivalent LIA constraints

For an integer $k \in \mathbb{Z}$, let $[k]_m$ denote its value modulo m (the remainder after division by m). This is the unique value satisfying: $0 \leq [k]_m < m \wedge \exists q \in \mathbb{Z}\ (k = m \cdot q + [k]_m)$. The quotient q encodes the number of times k "wraps around" in a number circle $[0, m)$. This can be equivalently written as the following, where $q' = -q$: $0 \leq [k]_m < m \wedge \exists q' \in \mathbb{Z}\ ([k]_m = m \cdot q' + k)$. We extend $[\ ]_m$ onto integer-valued expressions such that all subexpressions are computed in $\mathbb{Z}_m$: $[E_1 + E_2]_m = [[E_1]_m + [E_2]_m]_m = (E_1 + E_2) \bmod m$, and similarly for

subtraction and multiplication by a constant. Next, we describe a linear encoding over $\mathbb{Z}$ of the semantics of constraints over $\mathbb{Z}_m$. By introducing fresh variables for the quotients, we can encode a modular linear constraint $[E_1 \ \lhd \ E_2]_m$ (where $\lhd \in \{<, =, \leq\}$) as a conjunction of linear constraints over $\mathbb{Z}$.

We define a mapping $\Gamma(E_1 \ \lhd \ E_2)$ as:

$$\Gamma(E_1 \ \lhd \ E_2) = \exists q_1, q_2 \ . \begin{pmatrix} & 0 \leq E_1 + m \cdot q_1, E_2 + m \cdot q_2 < m \\ \wedge & E_1 + m \cdot q_1 \lhd E_2 + m \cdot q_2 \\ \wedge & 0 \leq vars(E_1 \ \lhd \ E_2) < m \end{pmatrix}.$$

The first conjunct computes the interpretation of E_i under $\mathbb{Z}_m$, the second constraint enforces the constraint of interest and the last constraint places bounds on the variables. Because $[[E_1]_m + [E_2]_m]_m = [E_1 + E_2]_m$, and similarly for all operations in a MLIA expression, it is sufficient to introduce one quotient variable q_i in each of E_1 and E_2; quotient variables are not needed for subexpressions. We extend Γ to Boolean combinations of constraints in the natural manner.

Encoding simplification: Though the translation looks straightforward, the quotient variables have unbounded domains and large coefficients (the modulo integer m). This can have a dramatic impact on performance, as it substantially weakens the (real) linear relaxation underlying the LIA decision procedure. However, we can frequently infer reasonably tight bounds on feasible quotient values. For some expression E, let l_E and u_E be the minimum and maximum feasible values of E under $\mathbb{Z}$ (assuming *variables* are restricted to $[0, m)$). As we have $[E]_m = E + m \cdot q_E$, we may impose the constraint $-\left\lfloor \frac{u_E}{m} \right\rfloor \leq q_E \leq -\left\lfloor \frac{l_E}{m} \right\rfloor$ without changing satisfiability. This bound for an expression can be extended to a constraint $E_1 \lhd E_2$ as follows.

▶ **Definition 3** (Quotient bound)**.** Given a MLIA constraint $E_1 \lhd E_2$, we have $E_1 + m \cdot q_{E_1} \lhd E_2 + m \cdot q_{E_2}$ as the corresponding LIA constraint. The following conjunction of constraints is called a *quotient bound* for $E_1 \lhd E_2$.

$$-\left\lfloor \frac{u_{E_1}}{m} \right\rfloor \leq q_{E_1} \leq -\left\lfloor \frac{l_{E_1}}{m} \right\rfloor \qquad \wedge \qquad -\left\lfloor \frac{u_{E_2}}{m} \right\rfloor \leq q_{E_2} \leq -\left\lfloor \frac{l_{E_2}}{m} \right\rfloor.$$

We extend Γ to Γ^{qb}, which returns a pair of the encoded constraints and the corresponding quotient bounds.

▶ **Example 4.** Consider the constraint $[x + 1 = y]_{16}$ from Example 1. Computing bounds for the LHS and RHS, we find that y (unsurprisingly) cannot overflow, and $x + 1$ overflows at most once. Hence $\Gamma^{qb}([x + 1 = y]_{16})$ yields:

$$\langle x + 1 + 16 \cdot q_x = y \wedge 0 \leq x + 1 + 16 \cdot q_x < 16 \wedge 0 \leq x < 16 \wedge 0 \leq y < 16, -1 \leq q_x \leq 0 \rangle.$$

Similarly, $\Gamma^{qb}([x \geq y]_{16})$ yields $\langle x \geq y \wedge 0 \leq x < 16 \wedge 0 \leq y < 16, \mathit{true} \rangle$. □

Further tightening of the bounds is possible by propagating inequalities produced by the encoding rather than considering expressions in isolation and deriving bounds from them. We illustrate this with an example.

▶ **Example 5.** Consider the constraint $[x + y + 2 = 1]_{16}$ and its corresponding LIA encoding:

$$x + y + 2 + 16 \cdot q_l = 1 \wedge 0 \leq x + y + 2 + 16 \cdot q_l < 16 \wedge 0 \leq x < 16 \wedge 0 \leq y < 16.$$

From the bounds on the variables x and y, we can deduce that $0 \leq x + y \leq 30$. Thus from $x + y + 2 + 16 \cdot q_l = 1$, we get $q_l \in [-31/16, -1/16]$. Therefore, we obtain quotient bounds $q_l = -1$ instead of $-2 \leq q_l \leq 0$ as we would have derived from Definition 3. □

3 QE and IG algorithms for MLIA

Though the LIA encoding maintains the word-level structure, it introduces new challenges to QE/IG since it adds intermediate (quotient) variables with a very large coefficient m; these in turn need to be quantified out. We adopt a logic-based Benders decomposition approach [19], which eliminates these quotient variables by enumeration; isolating the quotient selection problem from the rest of the QE/IG problem.

The master problem Q assigns values μ to the quotient variables, and the subproblem $\mu(H_r)$ is subjected to QE or IG if it is satisfiable, otherwise we extract a Benders cut from the unsatisfiable core of $\mu(H_r)$ and add it to Q. This continues until Q becomes unsatisfiable.

Benders cuts

Let $C_b = \{E_i + \tilde{q_i}m \leq F_i + \tilde{r_i}m \mid i = 1 \ldots n\}$ be the unsatisfiable core of $\mu(H_r)$, where $\tilde{q_i}$ and $\tilde{r_i}$ are the current assignments to quotients q_i, r_i. To restore feasibility, at least one constraint in C_b must be relaxed—so some q_i must decrease, or some r_i must increase. A valid cut, then, would be $c = \bigvee_i q_i < \tilde{q_i} \vee r_i > \tilde{r_i}$. However, this is somewhat weak: the constraint is only relaxed if the *difference* between q_i and r_i increases. Instead, then, we add the more general cut $\bigvee_i r_i - q_i > \tilde{r_i} - \tilde{q_i}$. For $c \equiv E_i + \tilde{q_i}m \lhd F_i + \tilde{r_i}m$ define $\sigma(c)$ as follows:

$$\sigma(c) = \begin{cases} r_i - q_i > \tilde{r_i} - \tilde{q_i} \text{ if } \lhd \in \{\leq, <\} \\ r_i - q_i \neq \tilde{r_i} - \tilde{q_i} \text{ if } \lhd \in \{=\} \end{cases} \tag{1}$$

Then for unsat core $C_b = \{E_i + \tilde{q_i}m \lhd F_i + \tilde{r_i}m \mid i = 1 \ldots n\}$, the cut is given by

$$\bigvee_i \sigma(E_i + \tilde{q_i}m \lhd F_i + \tilde{r_i}m).$$

Solving algorithms

We present algorithms that leverage an LIA solver to solve a conjunction of MLIA constraints; the extension to arbitrary Boolean combinations of constraints using SMT approaches is standard [12, 7] and omitted for the sake of space. Our algorithms assume the availability of the following three routines as black boxes.

- LIA_Decide(H): LIA_Decide is a decision procedure for LIA constraints. We assume that the solver's output is in the form $\langle \mathit{Result}, \mathit{Witness} \rangle$, which can be either $\langle \mathsf{SAT}, \mathit{Model} \rangle$ or $\langle \mathsf{UNSAT}, \mathit{Unsat\_Core} \rangle$.
- LIA_QE(X, H): LIA_QE is a QE procedure for LIA constraints, which produces a quantifier-free formula equivalent to $\exists X.H$. There are several QE algorithms for LIA such as Cooper's method [10], the Omega test [34] and automaton theoretic approaches [14].
- LIA_Interpolant(A, B): LIA_Interpolant is an IG procedure for LIA constraints. Given two mutually unsatisfiable LIA formulas A and B, it generates an interpolant I. Numerous IG algorithms for LIA exist, for example those of McMillan [29] and Brillout et al. [**?**].

QE algorithm for a conjunction of MLIA constraints

Putting all the pieces together, we now define a QE algorithm for a conjunction of MLIA constraints shown in Figure 1. The algorithm translates MLIA constraints H into LIA constraints H_r, and eliminates quantifiers from H_r.

	ITERATION 1	ITERATION 2	ITERATION 3
	$Q_1 \equiv [-1 \leq q_x \leq 0]$, $\mu_1 \equiv \{q_x = 0\}$	$Q_2 \equiv Q_1 \wedge [q_x \neq 0]$, $\mu_2 \equiv \{q_x = -1\}$	$Q_3 \equiv Q_2 \wedge [q_x \neq -1]$
$\mu(H_r)$	$c_1 : x \geq y \wedge 0 \leq x \leq 15 \wedge 0 \leq y \leq 15 \wedge$ $c_2 : x + 1 + 16 \cdot 0 = y \wedge \wedge 0 \leq x + 1 + 16 \cdot 0 \leq 15 \wedge$ $0 \leq x \leq 15 \wedge 0 \leq y \leq 15$	$c_1 : x \geq y \wedge 0 \leq x \leq 15 \wedge 0 \leq y \leq 15 \wedge$ $c_2 : x - 15 = y \wedge 0 \leq x - 15 \leq 15 \wedge$ $0 \leq x \leq 15 \wedge 0 \leq y \leq 15$	
R	$C \equiv \{c_1, c_2\}$, cut $[q_x \neq 0]$	$\neg\mu_2 = \{q_x \neq -1\}$	Q_3 UNSAT
H_{qf}	$H_{qf1} \equiv \mathit{false}$	$H_{qf2} \equiv H_{qf1} \vee y = 0$	$H_{qf3} \equiv H_{qf2} \equiv y = 0$

Figure 2 Steps performed during during QE from $[x \geq y \wedge x + 1 = y]_{16}$ using Algorithm 1.

In essence the algorithm eliminates the quotient variables in H_r by implicitly enumerating different quotient values using a Benders decomposition approach, and it eliminates the original existential variables using QE methods for LIA. The Benders approach decomposes H_r into a number of subproblems, say $H_r^1, H_r^2, \ldots, H_r^n$ for $n > 0$ by implicitly enumerating the quotients values. The decomposition has the following property:

$$H_r^1 \vee \ldots \vee H_r^n \equiv H_r.$$

MLIA_QE_BENDERS(X, H)
  $H_{qf} := \mathit{false}$
  $\langle H_r, Q\rangle := \Gamma^{qb}(H)$
  while$(\exists\mu$ solution of $Q)$ % Q is SAT
    $\langle S, R\rangle :=$ LIA_DECIDE$(\mu(H_r))$
    if$(S = \mathsf{SAT})$
      $H_{qf} := H_{qf} \vee$ LIA_QE$(\exists X\ (\mu(H_r)))$
      $Q := Q \wedge \neg\mu$
    else % R is the unsat core in this case
      $Q := Q \wedge \bigvee\{\sigma(c) \mid c \in R\}$
  return H_{qf}

Figure 1 QE from conjunction of MLIA constraints

Thus from the distributivity of the existential quantifier over the disjunction we have:

$$\exists X\ (H_r^1) \vee \ldots \vee \exists X\ (H_r^n) \equiv \exists X\ H_r.$$

Then, we apply QE on each of these disjuncts and compose the results; thus producing a quantifier free formula for the original problem. Thus, in our approach we leverage the Benders decomposition approach to obtain a disjunctive normal form needed to handle a Boolean combination of constraints for QE.

In more detail, whenever the master problem Q (Quotient selection problem) has a solution μ, we construct a Benders sub-problem $\mu(H_r)$ which is a conjunction of LIA constraints. If it turns out to be satisfiable, then QE is applied on this sub-problem, and the result is combined disjunctively with the previous QE results. In addition, the master problem Q is updated by adding the negation of currently used quotient values given by μ, eliminating such combinations in the next iteration. Otherwise the sub-problem is unsatisfiable with an unsatisfiable core R. In this case, the master problem Q is updated by adding the cut derived from R, eliminating the current selection of quotient values and possibly others. The algorithm returns when Q is unsatisfiable and the result is a (disjunction of) quantifier-free formula. Note that the resulting formula is expressed in LIA. However, this is not a limitation because the modern SMT solvers can handle a combination of different theories.

▶ **Example 6.** Recall $[x \geq y \wedge x + 1 = y]_{16}$ from Example 1. The LIA encoding produces:

$$x \geq y \wedge x + 1 + 16 \cdot q_x = y \ \wedge 0 \leq x + 1 + 16 \cdot q_x < 16 \wedge 0 \leq x < 16 \wedge 0 \leq y < 16.$$

The progress of the algorithm in Figure 1 is outlined in Figure 2. From the quotient bounds $-1 \leq q_x \leq 0$, we construct the master problem Q_1, and solve to obtain $\mu_1 = \{q_x = 0\}$.

```
MLIA_IG_Benders(A, B)
  ⟨A_r, Q_a⟩ := Γ^qb(A); ⟨B_r, Q_b⟩ := Γ^qb(B); I := false
  while(∃μ_a solution of Q_a) % Hence Q_a is SAT
    A_b := μ_a(A_r)
    ⟨S_a, R_a⟩ := LIA_Decide(A_b)
    if(S_a = UNSAT) % R_a is the unsat core in this case
      % eliminate μ_a
      Q_a := Q_a ∧ ⋁{σ(c) | c ∈ R_a}
    else
      I_b := true; Q_ba := Q_b
      while(∃μ_b solution of Q_ba) % Hence Q_ba is SAT
        B_b := μ_b(B_r)
        ⟨S_b, R_b⟩ := LIA_Decide(B_b)
        if(S_b = UNSAT)
          % R_b is the unsat core
          C_{R_b} := ⋁{σ(c) | c ∈ R_b}
          % eliminate μ_b
          Q_ba := Q_ba ∧ C_{R_b}; Q_b := Q_b ∧ C_{R_b}
        else % (A_b, B_b SAT, but not A_b ∧ B_b)
          I_b := I_b ∧ LIA_Interpolant(A_b, B_b)
          Q_ba := Q_ba ∧ ¬μ_b % Not same μ_b
    I := I ∨ I_b
    Q_a := Q_a ∧ ¬μ
  return I
```

Figure 3 Interpolants from conjunction of MLIA constraints A and B

We find $\mu_1(H_r)$ is unsatisfiable, with an unsatisfiable core of $\{x+1+16\cdot 0_{q_x} = y, x \geq y\}$ (having replaced the occurrence of q_x). From this, we derive the feasibility cut $\{q_x \neq 0\}$ and derive a new master problem. Solving Q_2, we obtain $\mu_2 = \{q_x = -1\}$. We find $\mu_2(H_r)$ is satisfiable, so we apply QE for LIA on $\exists x\ (\mu_2(H_2))$ obtaining $y = 0$ which is combined with previously obtained quantified free formula (which is *false* in this case). Now we add $q_x \neq -1$ to Q_2 obtaining a new master problem Q_3. Since Q_3 is unsatisfiable, the algorithm terminates and returns the quantifier-free formula $y = 0$. □

▶ **Theorem 7** (Soundness). *Given a conjunction of quantifier-free MLIA constraints F, and variable set X to eliminate, that is, a quantified formula $\exists X.F$,* MLIA_QE_Benders*(X,F) returns a quantifier-free formula semantically equivalent to $\exists X.F$.* ◀

IG Algorithm for a conjunction of MLIA constraints

The essence of the IG algorithm is similar to QE. We break A into disjunction $A_1 \vee \cdots \vee A_n$, given by different solutions to the quotient bounds for A, and similarly break up B. We then compute interpolants I_{ij} for each pair A_i, B_j and return $I = \vee_{i\in 1..n} \wedge_{j\in 1..m} I_{ij}$, which is an interpolant for the original constraints. We lazily construct the disjunctions of A and B by finding solutions μ_a and μ_b to the quotient bounds for each.

The algorithm shown in Figure 3 works as follows. Given the LIA encodings A_r and B_r for A and B, as well as their quotient bounds Q_a and Q_b. We find a solution μ_a for Q_a and

use this to specialize A_r. If this is unsatisfiable we generate a cut which is added to Q_a. If it is satisfiable, we set Q_{ba} to be Q_b. We find a solution μ_b and use this to specialize B_r. If this is unsatisfiable, we generate a cut and add this to Q_{ba} and Q_b (since it is independent of μ_a). Otherwise we have generated A_b and B_b, which are separately satisfiable in LIA but not together. We use an LIA interpolant generator to generate an interpolant which is added to I_b. We then add $\neg\mu_b$ to Q_{ba}, which guarantees we cannot generate the same solution μ_b. When Q_{ba} is unsatisfiable we have generated an interpolant for the particular solution μ_a of the variables of Q_a. We enforce that Q_a can no longer return this solution, and continue. Finally the outer while terminates with I an interpolant for A and B.

▶ **Example 8.** Recall Example 2. The LIA encoding produces:

$$\begin{array}{lcl} A_r & \equiv & y = x + 9 + 16q_a \wedge x \leq y \wedge y \leq z \wedge 0 \leq x + 9 + 16q_a \leq 15 \\ & & \wedge\, 0 \leq x \leq 15 \wedge 0 \leq y \leq 15 \wedge 0 \leq z \leq 15 \\ Q_a & \equiv & -1 \leq q_a \leq 0 \\ B_r & \equiv & z = y + 9 + 16q_b \wedge 0 \leq y + 9 + 16q_b \leq 15 \wedge 0 \leq y \leq 15 \wedge 0 \leq z \leq 15 \\ Q_b & \equiv & -1 \leq q_b \leq 0 \end{array}$$

A first solution of Q_a is $\mu_a \equiv \{q_a = 0\}$. Now $\mu_a(A_r)$ is satisfiable. We construct $Q_{ba} \equiv Q_b$. A first solution $\mu_b \equiv \{q_b = 0\}$. Now $\mu_b(B_r)$ is satisfiable.

An LIA interpolant for A_b and B_b returns $9 \leq y \wedge y \leq z$. I_b becomes this. We add $q_b \neq 0$ to Q_{ba}. We next find solution $\mu_b \equiv \{q_b = -1\}$. Now B_b is satisfiable. We generate an LIA interpolant of A_b and B_b, which is $y \leq z$, and conjoin this with I_b, which is unchanged. We add $q_b \neq -1$ to Q_{ba}. Now Q_{ba} is unsatisfiable and the inner loop terminates. We set I to be $9 \leq y \wedge y \leq z$. We add $q_a \neq 0$ to Q_a.

Now Q_a has a solution $\mu_a \equiv \{q_a = -1\}$. But $\mu_a(A_r)$ is unsatisfiable. We generate a cut $q_a > -1$ which is added to Q_a. Now Q_a is unsatisfiable and we return $I \equiv y \geq 9 \wedge y \leq z$. □

▶ **Theorem 9** (Soundness). *Given two conjunctions of MLIA constraints A and B such that $A \wedge B \models \bot$,* MLIA_IG_BENDERS*(A,B) returns a formula I where $A \rightarrow I$, $I \wedge B \models \bot$ and $vars(I) \subseteq vars(A) \cap vars(B)$.* ◀

4 Implementation and Experiments

Implementation

SOMOLIA-QE and SOMOLIA-IG denote the implementations of QE and IG algorithms respectively, where SOMOLIA stands for **So**lver for **Mo**dular **LIA**. They are written in Java and use Z3 [12] for input reading, QE and IG for LIA, and Gurobi [17] for checking the satisfiability of LIA formulas. More specifically, the procedure LIA_DECIDE is provided by Gurobi whereas the procedures LIA_QE and LIA_INTERPOLANT are provided by Z3 for LIA constraints. Our implementation supports input in SMT-LIB2 format [1] expressed over bit-vector logic (`BV`).

Benchmarks

We collected a set of 7,883 (4,526 conjunctive and 3,357 disjunctive) QE problems. They arose while verifying programs from the SV-COMP'16 repository [3] using RAHFT [24]. RAHFT is an abstraction-refinement tool, which takes as input a set of constrained Horn clauses of the form

$$p(X) \leftarrow \phi \wedge p_1(X_1) \wedge p_2(X_2) \wedge \ldots \wedge p_n(X_n), n \geq 0$$

Solver/Bench		QE Software Verification				QE Software Verification			
		C-4	C-8	C-16	C-32	D-4	D-8	D-16	D-32
Z3	#solved	**4526**	4401	3965	3962	1865	865	856	856
	Time (s)	**0.71**	1.15	0.68	0.60	81.13	1.66	1.68	1.69
SoMoLIA-QE	#solved	4523	**4521**	**4518**	**4517**	**3357**	**3357**	**3357**	**3357**
	Time (s)	1.30	**1.74**	**1.89**	**1.89**	**0.81**	**0.83**	**0.83**	**0.83**
		IG Software Verification				**IG Pigeonhole**			
		C-4	C-8	C-16	C-32	C-4	C-8	C-16	C-32
SoMoLIA-IG	#solved	529	529	529	529	**472**	**472**	**472**	**472**
	Time (s)	0.75	0.78	0.82	0.84	**3.64**	**3.73**	**3.75**	**3.76**
SoMoLIA-QE	#solved	529	529	529	529	297	297	297	297
	Time (s)	2.07	2.09	2.11	2.11	3.15	3.18	3.18	3.20
MathSAT5	#solved	**529**	**529**	**529**	**529**	378	356	334	320
	Time (s)	**0.01**	**0.01**	**0.01**	**0.02**	25.28	23.99	19.72	19.30

Table 1 Experiments on a set of QE and IG benchmarks arising from software verification

where p, p_i are predicate symbols, ϕ is a formula in some constraint theory, and X, X_i are vectors of variables. First, it applies fixed-point algorithms to derive an assertion for each predicate. The quantifier elimination problem comes when we have already derived assertions for each p_i, from which we need to derive an assertion for the predicate p. The assertion for $p(X)$ is computed as

$$\exists Z\ (\phi \wedge \phi_1(X_1) \wedge \phi_2(X_2) \wedge \ldots \wedge \phi_n(X_n))$$

where $\phi_i(X_i)$ is an assertion for $p_i(X_i)$ and $Z = X \setminus \bigcup_i X_i$. The disjunctive benchmarks are not arbitrary disjunctions; but a disjunction of small number of conjunctive simple QE instances arose during different iterations of the fixed-point computation. Note also that the disjunctive benchmarks are obtained during abstraction of the original programs and contain relatively simpler constraints than the conjunctive benchmarks; the latter are obtained from the refined programs. When Rahft fails to prove safety, it generates an abstract counterexample. If it is unsatisfiable, then interpolation is used for its generalisation. Let $\phi(t)$ be a list of constraint (conjunction) corresponding to the counterexample trace t, and l be the number of atomic constraints of $\phi(t)$. Then we break it into part A and part B as follows: for each $i \in [1 \ldots l-1]$, A contains the constraints up to length i whereas B contains the rest. The 529 interpolation benchmarks are derived this way. Furthermore, we extracted 472 interpolation problems from the category of *Pigeonhole* from SMT-COMP'16 [8]. The *Pigeonhole* benchmarks are appealing for three reasons: (1) they are arguably the most studied problems (see [35] and its references) where resolution proof complexity is exponential [18, 31], (2) they are good examples for exploiting symmetries [13] and (3) they have been applied for encoding cardinality constraints [20] among others. Note that all these benchmarks are expressed over LIA and are translated to constraints over `BV` of sizes 4, 8, 16 and 32.

Experimental setting

SoMoLIA-QE and SoMoLIA-IG are evaluated against the above benchmarks and compared with the QE and IG procedure for bit-vectors, implemented in Z3 [12] and MathSAT

[7] respectively. To the best of our knowledge, Z3 does not have an IG procedure for bit-vectors and MathSAT does not have a QE procedure. The results with the direct LIA encoding are omitted since they were disappointing, at least using the QE procedure for LIA implemented in Z3. The experiments were carried out on a MacBook Pro with a 2.7 GHz Intel Core i5 processor and 16 GB memory running OS X 10.11.6. The timeout for each experiment is set to 5 minutes. The goal of the experiments is to find out how our approach compares with the state-of-the-art approaches for QE/IG for MLIA constraints and to explore the scalability of our approach with respect to the bit-width.

The results are summarized in Table 1. The column headers starting with QE and IG indicate the results for QE and IG respectively. The text following them, that is, *software verification* and *Pigeonhole*, represent the source of the benchmarks. The sub-columns `C-`n or `D-`n ($n \in \{4, 8, 16, 32\}$) indicate conjunctive or disjunctive bit-vector constraints of size n. The rows in the table compare the results of QE/IG (the number of solved instances and the average QE time in seconds over the solved instances) for the different solvers. We compare Z3 with SoMoLIA-QE for QE, whereas we compare SoMoLIA-IG , SoMoLIA-QE (the IG problem is posed as a QE problem) and MathSAT for IG. The best result in each set of instances of size-N is bold-faced, where the best is defined to be the most solved instances, with ties resolved by the lowest runtime.

Discussion of the results

The results show that the performance (number of instances solved as well the time taken to solve them) of SoMoLIA-QE and SoMoLIA-IG is largely independent of bit-width, indicating their potential to scale to large size bit-vector problems. This is due to the bit-width independent encoding of the MLIA constraints and their efficient solving through Benders decomposition. The efficiency of these methods depend on the efficiency of the procedures LIA_DECIDE, LIA_QE and LIA_INTERPOLANT; all manipulating the LIA constraints. The number of calls to these procedures is controlled by the number of models of Q at the entry of the *while* loop. The enumeration of the model is the price we pay in our approach. The small number of timeouts are due to a large number of models of Q. Furthermore, a call to LIA_QE or LIA_INTERPOLANT is conditioned upon the result of LIA_DECIDE being satisfiable. The number of instances solved slightly decreases (for *conjunctive* QE) and the solving time increases as the bit-width increases, for the following reasons. Firstly, a problem can be unsatisfiable over n-bit but satisfiable over m-bit ($m > n$), witness Example 2. This invokes calls to expensive procedure like LIA_QE. Secondly, the bounds on the variables change according to the bit-width which may change the search space of the above procedures, affecting their efficiency.

Z3's performance consistently decreases as the bit-width increases. For conjunctive benchmarks, it decreases smoothly, whereas for disjunctive cases, there is a sharp decrease from 4 to 8 bits and is maintained through 32 bits. This difference in results between Z3 and SoMoLIA-QE is attributed to the bit-width dependent encoding (Z3) and bit-width independent encoding (SoMoLIA-QE) and the underlying methods to solve them. In our case, the word-level structure is maintained which offers word-level propagation and tends to be more efficient than the bit-width dependent *bit-blasting*, which destroys the word-level structure preventing it from propagating word-level information. Surprisingly, SoMoLIA-QE performed better than Z3 on disjunctive benchmarks though Z3 handles disjunction more efficiently. This is because these instances have a small number of disjunction and also a small number of models of Q. In addition, if LIA_DECIDE is efficient then we will be comparing Z3's performance on QE for LIA (our case) and BV (Z3's case). The latter is

inefficient since it is bit-width dependent.

For IG problems, the results show that MathSAT is two orders of magnitude faster than both SoMoLIA-IG and SoMoLIA-QE on software verification problems. Unlike SoMoLIA-IG, the IG algorithm in MathSAT is tuned for software verification problems and uses a layered approach (abstracting BV operations with uninterpreted function symbols, equality substitution, LIA encoding, etc.) to computing interpolants within an SMT solver, which could have allowed early pruning of unsatisfiable combinations of A's and B's [15]. As expected, SoMoLIA-IG performs better than SoMoLIA-QE since the QE algorithm is more expensive than the IG algorithm.

On *Pigeonhole* instances, SoMoLIA-IG performs better than MathSAT. We speculate that the reason is the following. Unlike in the *software verification* problems, each variable in the *pigeonhole* category has a unit coefficient, and this gives rise to only a small number of models of the quotient selection problem Q. Our reliance on model enumeration is therefore less of a problem than is the case with other benchmarks. The solver we use for decidability, Gurobi, handles the *pigeonhole* benchmarks particularly well. Cook et al. [9] have shown the advantage of using cutting planes, compared to resolution, for refuting *pigeonhole* formulas, and Gurobi is indeed cutting-plane based. Hence the performance ends up being dominated by the calls to LIA_INTERPOLANT, computing the interpolant for LIA, rather than calls to LIA_DECIDE. MathSAT, on the other hand, computes interpolants for bit-vectors. IG for LIA (used by Z3) is in principle more efficient than IG for BV (used by MathSAT). We suspect that this is the main source of our performance gain. Since the QE algorithms are much more expensive than IG, the results for SoMoLIA-QE are as expected.

At the moment, our tools do not use any word-level pre-processing. Such pre-processing could improve the results. Our approach scales poorly on those problems which give rise to a large number of models of Q.

5 Related Work

The widespread use of QE and IG in program analysis and verification has led to considerable work on efficient QE and IG procedures, especially for LIA. There are several well-known QE methods for LIA, for example, Cooper's method [10], The Omega test [34], and an automaton theoretic method [14]. Cooper's method is based on Presburger's decision procedure [32]. The Omega test [34] is a version of the well-known Fourier-Motzkin procedure for integer constraints. The method used by Z3 [4] is a combination of [10, 34]. Numerous IG algorithms for LIA exist, for example those of McMillan [26, 29], Griggio et al. [16] and Brillout et al. [**?**].

The QE/IG problem for MLIA, though much less studied, has gained considerable interest due to the ubiquity of two's complement semantics. The dominant approach to QE and IG for MLIA constraints uses bit-vector encoding, word-level simplifications, and *bit-blasting*, followed by bit-level QE [37] and IG [15]. The bit-blasted formula destroys the word-level information and is often too hard to solve. An alternative approach translates MLIA constraints into equivalent LIA constraints using an approach similar to ours, Bozzano et al. [5] or Brinkmann et al. [6], and applies QE/IG algorithms for LIA. In contrast, our approach does not apply QE/IG directly on the resulting LIA constraints, which are hard to solve efficiently; instead, it alternates between a Benders decomposition approach (to simplify the problem) and QE/IG algorithms for the simplified problem. Kafle *et al.* [25] has used the similar approach using Benders decomposition for deciding satisfiability of MLIA constraints.

Recently John and Chakraborty [22] proposed a layered approach to QE from MLIA

constraints, which invokes cheaper and incomplete layers such as word-level rewriting, equality substitution and simplification of MLIA constraints first, and then adapts a variant of Fourier-Motzkin to eliminate quantifiers from MLIA constraints, completely avoiding bit-blasting. They also present different approaches to handle a Boolean combination, one of which is based on SMT approach similar to ours. Their benchmarks, unlike ours, are derived from bounded model checking of RTL designs. They claim that most of their problems are solved as a result of the pre-processing, which is missing in our approach at the moment and we plan to integrate it in the future. Griggio [15] uses a similar layered approach to IG within an SMT solver. One layer involves translation to LIA, and IG for LIA. If the result is not an MLIA formula, he uses *bit-blasting* and bit-level IG as a last resort. The layers retain word-level information as much as possible. However, in our case we are not too concerned about having the resulting formula in MLIA, but an efficient procedure whose result is still useful in practice. Jain et al. [21] describe a word-level IG algorithm based on integer programming techniques for modular Diophantine (dis)equations, a variant of MLIA constraint with only equalities and disequalities.

6 Concluding remarks

We have described efficient, bit-precise and bit-blasting free QE and IG algorithms for MLIA constraints. Our algorithms use an LIA encoding of the constraints and harness a new approach using Benders decomposition with QE or IG for LIA. The overall result demonstrates that our approach is independent of the bit-width and has a potential to scale to large bit-width problems. More importantly, unlike bit-vector solvers, our approach uniformly handles (large) moduli, not necessarily powers of 2, including large primes. The QE algorithm outperforms the existing methods, while the IG algorithm loses out, on examples arising from software verification. However, the IG algorithm outperforms MathSAT on *Pigeonhole* benchmarks.

There are several interesting directions for future work. If the results of QE/IG is intended to be used in other applications, its translation to MLIA constraints of appropriate size would be essential. Therefore, we plan to explore such translations. Another avenue is optimising the algorithm. This includes, among others, simplification and pre-processing of MLIA formulas as done by the current bit-vector solvers, generalising the model or finding a stronger cut so as to reduce the number of iterations and finally developing techniques for handling Boolean combinations of constraints more efficiently.

Acknowledgments

We are grateful for support from the Australian Research Council. The work has been supported by Discovery Project grant DP140102194, and Graeme Gange is supported through Discovery Early Career Researcher Award DE160100568.

References

1 Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB standard: Version 2.5. Technical report, Department of Computer Science, University of Iowa, 2015. www.SMT-LIB.org.

2 J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962. doi:10.1007/BF01386316.

3 Dirk Beyer. Reliable and reproducible competition results with BenchExec and witnesses (Report on SV-COMP 2016). In M. Chechik and J.-F. Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'16)*, volume 9636 of *LNCS*, pages 887–904. Springer, 2016. doi:10.1007/978-3-662-49674-9_55.

4 Nikolaj Bjørner. Linear quantifier elimination as an abstract decision procedure. In J. Giesl and R. Hähnle, editors, *Automated Reasoning*, volume 6173 of *LNCS*, pages 316–330. Springer, 2010.

5 Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Ziyad Hanna, Zurab Khasidashvili, Amit Palti, and Roberto Sebastiani. Encoding RTL constructs for MathSAT: A preliminary report. *Electronic Notes in Theoretical Computer Science*, 144(2):3–14, 2006. URL: http://dx.doi.org/10.1016/j.entcs.2005.12.001.

6 Raik Brinkmann and Rolf Drechsler. Rtl-datapath verification using integer linear programming. In *Proceedings of the ASPDAC 2002 / VLSI Design 2002, CD-ROM, 7-11 January 2002, Bangalore, India*, pages 741–746. IEEE Computer Society, 2002. URL: http://dx.doi.org/10.1109/ASPDAC.2002.995022, doi:10.1109/ASPDAC.2002.995022.

7 Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. The MathSAT5 SMT solver. In N. Piterman and S. A. Smolka, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13)*, volume 7795 of *LNCS*, pages 93–107. Springer, 2013.

8 Sylvain Conchon, David Déharbe, Matthias Heizmann, and Tjark Weber. SMT-COMP, 2016. http://smtcomp.sourceforge.net/2016/.

9 William J. Cook, Collette R. Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, 1987. doi:10.1016/0166-218X(87)90039-4.

10 D. C. Cooper. Theorem proving in arithmetic without multiplication. In *Machine Intelligence*, volume 7, pages 91–99. 1972.

11 William Craig. Linear reasoning: A new form of the Herbrand-Gentzen theorem. *Journal of Symbolic Logic*, 22(3):250–268, 1957.

12 Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08)*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

13 David Déharbe, Pascal Fontaine, Stephan Merz, and Bruno Woltzenlogel Paleo. Exploiting symmetry in SMT problems. In N. Bjørner and V. Sofronie-Stokkermans, editors, *Automated Deduction (CADE-23)*, volume 6803 of *LNCS*, pages 222–236. Springer, 2011.

14 Vijay Ganesh, Sergey Berezin, and David L. Dill. Deciding Presburger arithmetic by model checking and comparisons with other methods. In M. Aagaard and J. W. O'Leary, editors, *Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design (FMCAD'02)*, volume 2517 of *LNCS*, pages 171–186. Springer, 2002. doi:10.1007/3-540-36126-X_11.

15 Alberto Griggio. Effective word-level interpolation for software verification. In *Proceedings of the International Conference on Formal Methods in Computer-Aided Design (FMCAD'11)*, pages 28–36. FMCAD Inc., 2011.

16 Alberto Griggio, Thi Thieu Hoa Le, and Roberto Sebastiani. Efficient interpolant generation in satisfiability modulo linear integer arithmetic. *Logical Methods in Computer Science*, 8(3), 2012.

17 Gurobi Optimization, Inc. Gurobi optimizer reference manual, 2016. http://www.gurobi.com.

18 Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985. doi:10.1016/0304-3975(85)90144-6.

19 J. N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96(1):33–60, 2003. doi:10.1007/s10107-003-0375-9.

20 Saïd Jabbour, Lakhdar Sais, and Yakoub Salhi. A pigeon-hole based encoding of cardinality constraints. In *Proceedings of the International Symposium on Artificial Intelligence and Mathematics (ISAIM'14)*, 2014. URL: https://www.cs.uic.edu/pub/Isaim2014/WebPreferences/ISAIM2014_Jabbour_etal.pdf.

21 Himanshu Jain, Edmund M. Clarke, and Orna Grumberg. Efficient Craig interpolation for linear diophantine (dis)equations and linear modular equations. In A. Gupta and S. Malik, editors, *Computer Aided Verification (CAV'08)*, volume 5123 of *LNCS*, pages 254–267, 2008.

22 Ajith K. John and Supratik Chakraborty. A layered algorithm for quantifier elimination from linear modular constraints. *Formal Methods in System Design*, 49(3):272–323, 2016.

23 Bishoksan Kafle and John P. Gallagher. Horn clause verification with convex polyhedral abstraction and tree automata-based refinement. *Computer Languages, Systems & Structures*, 47:2–18, 2017. URL: https://doi.org/10.1016/j.cl.2015.11.001, doi:10.1016/j.cl.2015.11.001.

24 Bishoksan Kafle, John P. Gallagher, and José F. Morales. Rahft: A tool for verifying Horn clauses using abstract interpretation and finite tree automata. In S. Chaudhuri and A. Farzan, editors, *Computer Aided Verification (CAV'16)*, volume 9779 of *LNCS*, pages 261–268. Springer, 2016. doi:10.1007/978-3-319-41528-4_14.

25 Bishoksan Kafle, Graeme Gange, Peter Schachte, Harald Søndergaard, and Peter J. Stuckey. A Benders decomposition approach to deciding modular linear integer arithmetic. In S. Gaspers and T. Walsh, editors, *Theory and Applications of Satisfiability Testing (SAT'17)*, volume 10491 of *LNCS*. Springer, 2017. To appear.

26 Kenneth L. McMillan. Interpolation and SAT-based model checking. In W. A. Hunt Jr. and F. Somenzi, editors, *Computer Aided Verification (CAV'03)*, volume 2725 of *LNCS*, pages 1–13. Springer, 2003. doi:10.1007/978-3-540-45069-6_1.

27 Kenneth L. McMillan. An interpolating theorem prover. *Theoretical Computer Science*, 345(1):101–121, 2005.

28 Kenneth L. McMillan. Lazy abstraction with interpolants. In T. Ball and R. B. Jones, editors, *Computer Aided Verification (CAV'06)*, volume 4144 of *LNCS*, pages 123–136. Springer, 2006.

29 Kenneth L. McMillan. Interpolants from Z3 proofs. In *Proceedings of the International Conference on Formal Methods in Computer-Aided Design (FMCAD'11)*, pages 19–27. FMCAD Inc., 2011.

30 Kenneth L. McMillan and Andrey Rybalchenko. Solving constrained Horn clauses using interpolation. Technical report, Microsoft Research, 2013.

31 Jakob Nordström. On the interplay between proof complexity and SAT solving. *SIGLOG News*, 2(3):19–44, 2015. doi:10.1145/2815493.2815497.

32 Mojżesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du 1ere Congrès de Mathématiciens des Pays Slaves*, pages 92–101. 1929.

33 Pavel Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *Journal of Symbolic Logic*, 62(3):981–998, 1997.

34 William Pugh. A practical algorithm for exact array dependence analysis. *Communications of the ACM*, 35(8):102–114, 1992. doi:10.1145/135226.135233.

35 Alexander A. Razborov. Proof complexity of pigeonhole principles. In W. Kuich, G. Rozenberg, and A. Salomaa, editors, *Developments in Language Theory (DLT'01)*, volume 2295 of *LNCS*, pages 100–116. Springer, 2001. doi:10.1007/3-540-46011-X_8.

36 Andrey Rybalchenko and Viorica Sofronie-Stokkermans. Constraint solving for interpolation. *Journal of Symbolic Computation*, 45(11):1212–1233, 2010.
37 Fabio Somenzi. Efficient manipulation of decision diagrams. *STTT*, 3(2):171–181, 2001. URL: `https://doi.org/10.1007/s100090100042`, `doi:10.1007/s100090100042`.

7 Appendix

In this section, we provide proofs to the theorems introduced in the paper and present an extension of the QE algorithm to a Boolean combination. The interpolation algorithm can be extended similarly.

The proofs of the theorems rely on Propositions 1 and 3 from Kafle *et al.* [25]:

[25, Proposition 1] *Let H be a formula and $\Gamma^{qb}(H)$ be its LIA encoding. Then H and $\Gamma^{qb}(H)$ are logically equivalent.*

[25, Proposition 3] *Given a system of MLIA constraints H, let H_b be an infeasible sub-problem of H for the quotient μ and $C = \{E_i + \tilde{q}_i m \lhd F_i + \tilde{r}_i m \mid i = 1 \dots n\}$ be any unsat core of H_b, and $C_b = \bigvee\{\sigma(c) \mid c \in C\}$. Then (1) C_b excludes μ and (2) C_b does not exclude any model of H.*

Theorem 7 *Given a conjunction of quantifier-free MLIA constraints H, and set of variables X to eliminate (i.e; $\exists X.H$),* MLIA_QE_Benders*(X,H) returns a quantifier-free formula semantically equivalent to $\exists X.H$.*

Proof. The algorithm translates MLIA formula H into semantically equivalent LIA formula $H_r \wedge Q$. Namely, by Proposition 1, $H \equiv \Gamma^{qb}(H) \equiv H_r \wedge Q$.

If Q has no model, then H_r is unsatisfiable (so is H), so the formula *false* is the quantifier-free formula equivalent to H.

Otherwise Q has a non-zero but finite set of models, as all variables in Q are bounded. Let $\mu_1, \mu_2, \dots, \mu_n$ $(n \geq 1)$ be the different models of Q and $\mu_1(H_r), \mu_2(H_r), \dots, \mu_n(H_r)$ be sub-problems of H_r such that

$$H_r \wedge Q \equiv \mu_1(H_r) \vee \mu_2(H_r) \vee \dots \vee \mu_n(H_r).$$

The soundness of this decomposition is guaranteed by the soundness of Proposition 3, since it does not exclude any model μ' of Q such that $\mu'(H_r)$ is feasible.

Now since existential quantification distributes over disjunction, we have

$$\exists X.(H_r \wedge Q) \equiv \exists X.\mu_1(H_r) \vee \exists X.\mu_2(H_r) \vee \dots \vee \exists X.\mu_n(H_r).$$

The quantifiers from the RHS can be eliminated using the sound procedure LIA_QE which produces quantifier-free equivalents to each $\exists X.\mu_i(H_r)$ $(i = 1 \dots n)$. Thus, the disjunction of such formulas is equivalent to $\exists X.(H_r \wedge Q)$ and to $\exists X.H$.

◀

Theorem 9 *Given two conjunctions of MLIA constraints A and B such that $A \wedge B \models \bot$,* MLIA_IG_Benders*(A,B) returns a formula I where $A \to I$, $I \wedge B \models \bot$ and $vars(I) \subseteq vars(A) \cap vars(B)$.*

Proof. The algorithm translates MLIA formulas A and B into semantically equivalent LIA formulas $A_r \wedge Q_a$ and $B_r \wedge Q_b$ respectively.

Then it breaks A_r and B_r into sub-problems based on the models of Q_a and Q_b, respectively.

We omit the trivial cases when either Q_a or Q_b has no model.

Let $\mu_1, \mu_2, \ldots, \mu_n$ $(n \geq 1)$ be the different models of Q_a and $\mu_1(A_r), \mu_2(A_r), \ldots, \mu_n(A_r)$ be sub-problems of A_r. Similarly, let $\sigma_1, \sigma_2, \ldots, \sigma_n$ $(n \geq 1)$ be the different models of Q_b and $\sigma_1(B_r), \sigma_2(B_r), \ldots, \sigma_m(B_r)$ be sub-problems of B_r. Then we have, as above:

$$\begin{aligned} A_r \wedge Q_a &\equiv \mu_1(A_r) \vee \mu_2(A_r) \vee \ldots \vee \mu_n(A_r) \\ B_r \wedge Q_b &\equiv \sigma_1(B_r) \vee \sigma_2(B_r) \vee \ldots \vee \sigma_m(B_r) \end{aligned}$$

As before, Proposition 3 guarantees that no model of Q_a or Q_b which leads to feasible solutions of the corresponding subproblems are left out.

Then the algorithm produces an interpolant $I \equiv \bigvee_{i=1,\ldots,n} \bigwedge_{j=1,\ldots,m} I_{i_j}$, where I_{i_j} is an interpolant between $\mu_i(A_r)$ and $\sigma_j(B_r)$. The correctness of I_{i_j} follows from the soundness of the procedure LIA_INTERPOLANT.

It remains to show that I is an interpolant of A and B.

1. $A \implies I$**:** For any $i \in [1 \ldots n]$, $\mu_i(A_r)$ implies the formula $\bigwedge_{j=1,\ldots,m} I_{i_j}$. This is because $\mu_i(A_r)$ implies each of I_{i_j} for $j = 1 \ldots m$ since I_{i_j} is an interpolant between $\mu_i(A_r)$ and $\sigma_j(B_r)$. Therefore, the disjuction $\mu_1(A_r) \vee \mu_2(A_r) \vee \cdots \vee \mu_n(A_r)$ implies the disjunction $\bigvee_{i=1,\ldots,n} \bigwedge_{j=1,\ldots,m} I_{i_j}$ and thus $A \implies I$.
2. $I \wedge B \implies \bot$**:** Since each $\sigma_j(B_r)$ is inconsistent with I_{i_j} for any $i \in [1 \ldots n]$ (I_{i_j} is an interpolant between $\mu_i(A_r)$ and $\sigma_j(B_r)$), the formula $\bigwedge_{j=1,\ldots,m} I_{i_j}$ is inconsistent with the formula $\sigma_1(B_r) \vee \sigma_2(B_r) \vee \cdots \vee \sigma_m(B_r)$ and thus with B. From this it follows that $I \wedge B \implies \bot$ since each disjunct in I is inconsistent with B.
3. $vars(I) \subseteq vars(A) \cap vars(B)$**:** $vars(I_{i_j}) \subseteq vars(\mu_i(A_r)) \cap vars(\sigma_j(B_r))$ since I_{i_j} is an interpolant between $\mu_i(A_r)$ and $\sigma_j(B_r)$. Reasoning similarly outwards in I, we have $vars(\wedge_{j=1,\ldots,m} I_{i_j}) \subseteq \mu_i(A_r) \cap \bigcup_j vars(\sigma_j(B_r))$. This is because, for a given i, the conjunction of interpolants can only contain variables common to $\mu_i(A_r)$ and $\sigma_j(B_r)$ for some j. Continuing this way, we have $vars(I) \subseteq vars(A) \cap vars(B)$.

◀

Extending the algorithms to a Boolean combination

The Algorithm in Figure 1 can easily be extended to a Boolean combination of MLIA in an eager DPLL(T) style—we introduce a fresh Boolean variable representing the activation of each atomic MLIA constraint, and add the resulting *Boolean skeleton* to the quotient selection problem Q. A model of Q then determines both the choice of quotients *and* the set of constraints to be added to the Benders subproblem. When extracting cuts from the subproblem, we must then also account for these activation literals (i.e., $q_i > q_j$ becomes $act_{ij} \implies q_i > q_j$).

▶ **Example 10.** Consider a set of constraints $(c_1 \vee (c_2 \wedge c_3))$. We introduce fresh variables a_1, a_2, a_3 (plus intermediate t_{23}), adding the *Boolean skeleton* $(a_1 \vee t_{23}) \wedge (t_{23} \rightarrow a_2) \wedge (t_{23} \rightarrow a_3)$ to Q. Given $\mu_Q = \{\neg a_1, t_{23}, a_2, a_3\}$, we will only add c_2 and c_3 (also called *active constraints*) to the Benders subproblem. Similarly, if the resulting unsat core consists only of c_3, the cut will contain the additional disjunct $\neg a_3$. □

Below, we present an extension of the algorithm in Figure 1 to handle a Boolean combination of MLIA constraints. Unlike the previous version, the master problem (Q) now also contains the *Boolean skeleton* of the problem. An assumption (activation literals true under the model) is obtained from the model of Q from which a conjunctive active MLIA problem is derived. The algorithm behaves like before on this conjunction except when adding a negation of the current solution or a cut to the master problem. These are added only when the assumption (ψ) or the conjunction of activation literals of the unsatisfiable core R holds.

MLIA_QE_Benders(X, P)
  $H_{qf} := \mathit{false}$; $Q :=$ Bool_Skeleton(P)
  while($\exists \mu$ solution of Q) % Q is SAT
    $\psi :=$ Assumption(P, μ_Q) % conjunction of active P literals as given by μ
    $H :=$ Active_Constraints(P, ψ) % conjunction of active P constraints under ψ
    $\langle H_r, Q' \rangle := \Gamma^{qb}(H)$; $Q := Q \wedge Q'$
    $\langle S, R \rangle :=$ LIA_Decide$(\mu(H_r))$
    if$(S = \mathsf{SAT})$
      $H_{qf} := H_{qf} \vee$ LIA_QE$(\exists X\ (\mu(H_r)))$
      $Q := Q \wedge (\psi \implies \neg\mu)$
    else % R is the unsat core in this case
      $Q := Q \wedge (ac(R) \implies \bigvee\{\sigma(c) \mid c \in R\})$
  return H_{qf}

Figure 4 QE from a Boolean combination of MLIA constraints