A Benders Decomposition Approach to Deciding Modular Linear Integer Arithmetic

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

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

Abstract. Verification tasks frequently require deciding systems of linear constraints over modular (machine) arithmetic. Existing approaches for reasoning over modular arithmetic use bit-vector solvers, or else approximate machine integers with mathematical integers and use arithmetic solvers. Neither is ideal; the first is sound but inefficient, and the second is efficient but unsound. We describe a linear encoding which correctly describes modular arithmetic semantics, yielding an optimistic but sound approach. Our method abstracts the problem with linear arithmetic, but progressively refines the abstraction when modular semantics is violated. This preserves soundness while exploiting the mostly integer nature of the constraint problem. We present a prototype implementation, which gives encouraging experimental results.

1 Introduction

Linear integer arithmetic (LIA) and its decision procedures have been studied for a long time. Here we consider the important variant in which the integers involved are in fact integers modulo m for some m. We refer to this variant as modular LIA, MLIA in short. Decision procedures for MLIA are needed for sound reasoning about "machine arithmetic" and related problems in software verification and analysis. In this paper we address the problem of deciding a Boolean combination of MLIA constraints, that is, whether a system of MLIA constraints is satisfiable, and if so, how.

Existing approaches for deciding MLIA either use bit-vector solvers, or else approximate fixed-size integers ($\mathbb{Z}_m$, the integers modulo m) with unbounded mathematical integers ($\mathbb{Z}$) and use LIA solvers. The theory of bit-vectors allows modeling the precise semantics of two's complement arithmetic and is a natural candidate for modelling MLIA constraints. However this approach is inefficient, especially for large bit-vector problems that are primarily arithmetic in nature, and for problems involving long bit-vectors [28, 19, 37]. Most bit-vector solvers are based on some form of SAT encoding which tends to obscure word-level information and consume excessive memory. On the other hand, the LIA approach is efficient but unsound because approximating fixed-size integers with mathematical integers ignores the "wrap around" nature of fixed-width integer

arithmetic. As a result, the (un)satisfiability of a problem over $\mathbb{Z}$ does not imply its (un)satisfiability over $\mathbb{Z}_m$, as we now show. To keep examples simple we shall usually assume unsigned 4-bit arithmetic (so that $m = 16$). (Our method handles signed or unsigned arithmetic equally well.) Constraints will often be given in the form $id : c$, so that constraint c can be referred to through the identifier id. We indicate a formula F interpreted in $\mathbb{Z}_m$ with the notation $[F]_m$.

Example 1. Consider $F_1 = c_1 : x \geq y \wedge c_2 : x + 1 = y$. Clearly $[F_1]_{16}$ allows solution $x = 15(1111)$ and $y = 0(0000)$, but F_1 is unsatisfiable in $\mathbb{Z}$ since c_1 is in conflict with c_2. □

Example 1 is adapted from a benchmark in the `pspace` subset [18] of `SMT-LIB QF_BV` [11], translated to MLIA form. It is typical of software verification problems; it tests the overflow of integer addition. Its model consists of x with only 1-bits and y with only 0-bits regardless of the size of bit-vectors x and y. But it has proven to be a challenging example for sufficiently long bit-vectors for *bit-blasting* solvers, and the problem becomes increasingly intractable as the size of bit-vectors increases [37]. On the other hand, LIA solvers may be efficient but they produce an unsound result for Example 1 since F_1 is unsatisfiable over $\mathbb{Z}$ and satisfiable over $\mathbb{Z}_m$.

A formula may just as well be unsatisfiable over $\mathbb{Z}_m$ and satisfiable over $\mathbb{Z}$.

Example 2. Consider $F_2 = c_1 \colon y = x + 9 \wedge c_2 \colon z = y + 9 \wedge c_3 \colon x \leq y \wedge c_4 \colon y \leq z$. Here F_2 is satisfiable in $\mathbb{Z}$ with solutions like $x = 0$, $y = 9$, $z = 18$. But $[F_2]_{16}$ is unsatisfiable, as it requires $(x + 18) \equiv_{16} z$, so there must be a wrap around between x and z. And so at least one of the two inequalities fails not hold. □

Hence we can trust neither "satisfiable" nor "unsatisfiable" verdicts from a LIA solver. In practice, neither conventional approach is satisfactory.

In this paper, we develop an optimistic but sound approach—abstracting the problem with linear arithmetic, but progressively refining the abstraction when modular semantics is violated. By transforming MLIA constraints to LIA constraints that correctly describe the modular arithmetic semantics, we can reuse existing approaches for LIA problems while maintaining soundness. In contrast to bit blasting, the method is independent of the number of bits used to represent the variables and it uniformly handles (large) moduli, not necessarily powers of 2, including large primes. It is rather easy to understand and applies beyond linear arithmetic. Moreover, assuming two's complement arithmetic, signed and unsigned integers can be treated uniformly.

In summary we make the following contributions:

1. We give a semantics preserving transformation of MLIA constraints to LIA constraints (Section 3).
2. We design novel algorithms for deciding MLIA constraints which use this transformation effectively (Section 4).
3. We present experimental results obtained with a prototype implementation (Section 5). As they are comparable to those of the best state-of-the-art solvers, we consider this a significant proof-of-concept.

2 Preliminaries

In this paper, we consider a logic of quantifier-free MLIA constraints defined as follows, where F ranges over formulas, C over atomic constraints, E over fixed-size linear arithmetic expressions, v over fixed-size integer variables and a over fixed-size integer constants:

$$F = C \mid \neg C \mid F \vee F \mid F \wedge F$$
$$C = E < E \mid E \leq E \mid E = E \qquad E = a \mid v \mid a \cdot E \mid E - E \mid E + E$$

The other linear constraints $\{>, \geq, \neq\}$ are rewritten as negated elements of C. For $e \in E$, let $vars(e)$ denote the set of variables appearing in e. We extend this to elements of F and C in the obvious way.

For an integer $k \in \mathbb{Z}$, let $[k]_m$ denote its value modulo m (the remainder on division by m). This is the unique value satisfying:

$$0 \leq [k]_m < m \wedge \exists q \in \mathbb{Z} \,.\, k = m \cdot q + [k]_m \tag{1}$$

The quotient q encodes the number of times k "wraps around" in a number circle before landing in the range $[0, m)$. Equation 1 can be equivalently written as Equation 2, where $q' = -q$.

$$0 \leq [k]_m < m \wedge \exists q' \in \mathbb{Z} \,.\, [k]_m = m \cdot q' + k \tag{2}$$

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. Observe that $[\,]_m$ is preserved under translation by multiples of m. Thus for $e \in E$, $[e]_{2^b}$ reflects e computed with b bit fixed precision machine arithmetic. We also extend $[\,]_m$ to functions and maps pointwise.

Note that by our definition of $\mathbb{Z}_m$, negative E values are mapped to positive $[E]_m$, so it is necessary to distinguish signed two-complement comparisons $(<_s, \leq_s)$ from unsigned ones $(<_u, \leq_u)$. For $\lhd \in \{<, \leq, =\}$, the interpretation of $E_1 \lhd_u E_2$ under $\mathbb{Z}_m$ is $[E_1 \lhd_u E_2]_m \equiv [E_1]_m \lhd [E_2]_m$; for the signed case, $[E_1 \lhd_s E_2]_m \equiv \left[E_1 + \frac{m}{2}\right]_m \lhd \left[E_2 + \frac{m}{2}\right]_m$. In the following, we shall assume unsigned comparisons, but signed comparisons can be handled this way. Observe that we must be careful when manipulating modular linear constraints: rewriting $[A \leq B]_m$ as $[A - B \leq 0]_m$ *does not* preserve equivalence (for either signed or unsigned comparison).

An *assignment* μ to F is a mapping from $vars(F)$ onto $\mathbb{Z}_m$. We write $\mu(E)$ to denote the value of E under μ. μ is a *model* of F if F evaluates to TRUE under μ and we denote it by $\mu \models_{\mathbb{Z}_m} F$. Similarly if μ is a $\mathbb{Z}$-model of F, we simply write $\mu \models F$. Also observe that if μ is a model of F, and for any expression E appearing in an inequality $0 \leq \mu(E) < m$, then $[\mu]_m$ is a model of $[F]_m$ (as then $[\mu(E)]_m = \mu(E)$).

$$
\begin{array}{l|l|l}
P = \mathbf{min}_{x,y}\ f(x) & P^\sharp = \mathbf{min}_x\ f(x) & Q(\tilde{x}) = \mathsf{min}_y\ 0 \\
\quad \mathbf{s.t.}\ A(x) \geq b \wedge & \quad \mathbf{s.t.}\ A(x) \geq b \wedge & \quad \mathbf{s.t.}\ B(\tilde{x}, y) \geq c \wedge \\
\quad\quad B(x,y) \geq c \wedge & \quad\quad B^\sharp(x) \geq c \wedge & \quad\quad y \in D_y \\
\quad\quad \ldots & \quad\quad x \in D_x & \\
\quad\quad x \in D_x,\ y \in D_y & & \\
\quad \text{(a)} & \quad \text{(b)} & \quad \text{(c)}
\end{array}
$$

Fig. 1. (a) A decomposable MIP, (b) initial relaxed master, (c) Benders subproblem

Benders decomposition

Benders decomposition [4] is an approach for solving large integer linear programming problems, frequently applied where a problem consists of independent subproblems, connected via a small set of variables.

Consider the mixed integer programming problem (MIP) shown in Figure 1(a). P is an optimisation problem, minimising $f(x)$ over variables x, y, consisting of constraints A over x, and additional constraints over x, y. Rather than solving P directly, we may instead fix x to some optimal value $\tilde{x}$, then check whether there exists a consistent extension $\tilde{y}$ satisfying $B(\tilde{x}, \tilde{y})$.

We thus construct a *relaxed master problem* $P^\sharp$, shown in Figure 1(b), which is a relaxation of the projection $\exists y.\ P$. 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 $B^\sharp(x) \geq \mathbf{sup}_{y \in D_y} B_i(x,y)$. As $B^\sharp(x)$ may permit invalid solutions, this is combined with a subproblem $Q(\tilde{x})$ which check 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 is added to $P^\sharp$ (effectively 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* [25], the subproblem may be an arbitrary decision problem, and the feasibility cut is extracted from the unsatisfiability proof of Q.

3 From MLIA to equivalent LIA constraints

In this section, we describe a linear encoding over $\mathbb{Z}$ of the semantics of constraints over $\mathbb{Z}_m$. Recall the definition of $[k]_m$, being the unique value satisfying Equation 2. By introducing fresh variables for the quotients and remainders, we can encode a modular linear constraint $[E_1 \ \lhd\ E_2]_m$ as a conjunction of linear constraints on $\mathbb{Z}$. We define a mapping Γ as:

$$
\Gamma(E_1 \ \lhd\ E_2) = \exists q_1, q_2, e_1, e_2\ .\ \begin{pmatrix} 0 \leq e_1, e_2 < m \\ \wedge\ e_1 = E_1 + m \cdot q_1 \\ \wedge\ e_2 = E_2 + m \cdot q_2 \\ \wedge\ \quad e_1 \lhd e_2 \end{pmatrix}
$$

The first three conjuncts compute the interpretation of E_i under $\mathbb{Z}_m$, and the final constraint enforces the constraint of interest. 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. $\Gamma(E_1 \lhd E_2)$ can be simplified by eliminating the existential variables e_1 and e_2 leaving behind only the quotient variables.

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

We extend Γ to Boolean combinations of constraints in the natural manner (eliminating $\neq$ through disjunction). The above encoding ($\Gamma(F)$) preserves equi-satisfiability as stated in Lemma 1. Note that variables are not bounded in $\Gamma(F)$.

Lemma 1 (Equi-satisfiability). *Let F be a formula and $\Gamma(F)$ be its LIA encoding. Then F and $\Gamma(F)$ are equi-satisfiable. Further if $\Gamma(F)$ is satisfiable, then there exists a model of $\Gamma(F)$ such that $\bigwedge_{v \in vars(F)} 0 \leq v < m$.* □

Let $\Gamma_1(F)$ be $\Gamma(F) \wedge \bigwedge_{v \in vars(F)} 0 \leq v < m$. These bounds restrict the search space of LIA solvers; possibly resulting in a faster convergence. $\Gamma_1(F)$ correctly encodes the semantics of modular arithmetic as stated in Proposition 1.

Proposition 1 (Soundness of the encoding). *Let F be a formula and $\Gamma_1(F)$ be its LIA encoding. Then F and $\Gamma_1(F)$ are logically equivalent.* □

Note that $\Gamma_1(F)$ is a quantified-formula though not F.

The formula $\Gamma_1(E_1 \lhd E_2)$ contains: two constraints expressing the lower and upper bounds of each expression $E_i (i = 1, 2)$, a constraint computing the interpretation of each expression and a constraint of interest totalling 5. In addition, there are $2n$ constraints representing the upper and lower bounds of the variables, where $n = \#\ vars(E_1 \lhd E_2)$. Then, the size of the transformed formula is given by the following proposition.

Proposition 2 (Bound on the size of the transformed formula). *Given a formula F with n variables and m atomic constraints, $\Gamma_1(F)$ contains at most $2n + 5m$ atomic constraints.* □

Encoding simplification

While 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 as we show next. 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.

For example, given a modular expression $x + 2y$ over unsigned integer and its corresponding LIA expression $x + 2y + m.q$, we derive $-2 \leq q \leq 0$.

An expression bound can be extended to a constraint $E_1 \lhd E_2$ as follows.

Definition 1 (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 $E_1 \lhd E_2$.*

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

If we infer a fixed value for q_E (as will be the case for variables and constant expressions), we may replace $m \cdot q_E$ with the appropriate constant. If bounds on q_E are narrow but do not fix a value for q_E, we may eliminate q_E by introducing a disjunction for the possible values of q_E—moving the wrapping decision from the LIA solver to the SAT solver. In case the bounds are not tight, we apply Benders decomposition, as discussed in Section 4.

Example 3. Consider the constraint $c_2 : [x + 1 = y]_m$ from Example 1. Computing bounds for the LHS and RHS, we find that y (unsurprisingly) cannot overflow, and $x + 1$ overflows at most once. This yields the encoding

$$x + 1 + m \cdot q_x = y \ \wedge \ 0 \leq x + 1 + m \cdot q_x \leq 15 \ \wedge \ -1 \leq q_x \leq 0 \\ \wedge \ 0 \leq x \leq 15 \ \wedge \ 0 \leq y \leq 15$$

As the domain of q_x is small, we eliminate it by case-splitting:

$$\begin{aligned} & x - 15 = y \wedge 0 \leq x - 15 \leq 15 \wedge 0 \leq x \leq 15 \wedge 0 \leq y \leq 15 && \text{(where } q_x = -1\text{)} \\ \vee \ & x + 1 = y \wedge 0 \leq x + 1 \leq 15 \wedge 0 \leq y \leq 15 \wedge 0 \leq x \leq 15 && \text{(where } q_x = 0\text{)} \end{aligned}$$

Similarly, the constraint $[x \geq y]_m$ yields $x \geq y \wedge 0 \leq x, y \leq 15$. □

Challenges in solving LIA formula directly

However, the resulting encodings turn out to be difficult to solve directly with current MIP solvers. This is, in part, due to weakening of the linear relaxation.

Example 4. Consider the pair of constraints:

$$P = [x = y + 3 \wedge y = x - 4]_m$$

These are inconsistent for any modulus $m \geq 2, m \neq 7$. The transformed constraints are:

$$\Gamma(P) = (x = m \cdot q_y + y + 3 \wedge y = m \cdot q_x + x - 4 \wedge x, y \in [0, m))$$

If integrality constraints on q_x, q_y are relaxed, $\Gamma(P)$ is easily satisfied—with $\{x = 0, y = 0, q_x = \frac{4}{m}, q_y = -\frac{3}{m}\}$. Indeed, any pair of (x, y) values admits a corresponding solution to the relaxation of the quotients. However, for any fixed integer assignment to (q_x, q_y), the residual subproblem can easily be shown to be inconsistent. □

A second pragmatic difficulty to solving these problems is due to numerical behaviour of MIP solvers. Commercial MIP solvers are well known to return non-solutions or claim optimality for non-optimal solutions due to rounding errors [30, 31]. These problems are exacerbated in the presence of very large coefficients: not only is the solver forced to divide by large constants, intermediate computations and even integral solutions may have no exact floating-point representation. This becomes a significant problem when using moduli above 2^{32}.

We now consider how to solve these problems in practice.

4 Solving MLIA constraints

For the reasons discussed above, the transformed formulae are difficult to solve directly. We can ameliorate this in several ways. First, we can try to avoid introducing quotient variables altogether; optimistically solving under integer semantics, and transforming only when necessary (Algorithm 1). This can accelerate solving, but in the worst case still requires transforming all constraints. Second, we can reduce the impact of quotient variables by adopting a Benders decomposition approach (Algorithm 2) to isolate quotient selection from the rest of the problem. We detail these approaches below.

Solving algorithms

For simplicity of presentation we assume, without loss of generality, that the input is a conjunction of MLIA constraints (though our method applies more generally to a Boolean combination). Our algorithms assume the availability of the following methods.

LIA_DECIDE(H): An LIA solver (oracle) capable of generating either a model or an unsatisfiable core (a minimal set of unsatisfiable constraints). We assume that its output is a tuple of the form $\langle \mathit{Result}, \mathit{Witness} \rangle$, which can be either $\langle \mathsf{SAT}, \mathit{Model} \rangle$ or $\langle \mathsf{UNSAT}, \mathit{Unsat\_Core} \rangle$.

MLIA_MODEL(H, μ): Given an interpretation μ such that $\mu \models H$ (in this context, returned by LIA_DECIDE), the procedure checks whether $\mu \models_{\mathbb{Z}_m} H$ or not. It returns a tuple $\langle \mathit{Result}, \mathit{Witness} \rangle$, which can be either $\langle \mathit{Yes}, \mathit{Model} \rangle$ or $\langle \mathit{No}, \mathit{Conflict} \rangle$. We call c a conflict with respect to μ iff $\mu \not\models_{\mathbb{Z}_m} c$.

Putting all the pieces together, we present three algorithm for solving MLIA problems. The third is the final product; it combines Algorithms 1 and 2.

1. Transforming constraints lazily. Algorithm 1 proceeds as follows. A set H of MLIA constraints is passed to an LIA solver (line 7). The solver returns one of the following:

- SAT(line 8): We have $R \models H$. Then we check whether $R \models_{\mathbb{Z}_m} H$ or not. If so, the procedure returns SAT and a model (line 11). Otherwise there exists a constraint c such that $R \not\models_{\mathbb{Z}_m} c$, which is replaced by $\Gamma_1(c)$ (line 13).

Algorithm 1 Solves MLIA constraints lazily

```
1: function MLIA_Decide_Lazily(H)
2:     Input: A set of linear constraints of the form
3:             E1 ◁ E2, with ◁ ∈ {<, =, ≤}
4:     Output: ⟨SAT, Model⟩ or ⟨UNSAT, Unsat_Core⟩
5:     T ← ∅                                          ▷ transformed constraints
6:     while True do
7:         ⟨S, R⟩ ← LIA_Decide(H)
8:         if S = SAT then
9:             ⟨IS, r⟩ ← MLIA_Model(H, R)
10:            if IS = Yes then
11:                return ⟨SAT, R⟩
12:            else
13:                H ← H \ {r} ∪ Γ1(r); T ← T ∪ {Γ1(r)}
14:        else
15:            if R ⊆ T then return ⟨UNSAT, R⟩
16:            else
17:                for each c ∈ R do
18:                    if c ∉ T then
19:                        H ← H \ {c} ∪ Γ1(c); T ← T ∪ {Γ1(c)}
```

– UNSAT(line 14): In this case, there is an unsatisfiable core R over $\mathbb{Z}$. If all constraints in R are transformed then it is also the core over $\mathbb{Z}_m$ and the algorithm terminates (line 15). Otherwise, it replaces each $c \in R$ by $\Gamma_1(c)$ in the solver (line 17-19).

The algorithm runs until it exits from one of the above cases. During the run of the algorithm, we keep track of the set of the transformed constraints, which is initialized to empty set at the start of the algorithm (line 5).

Example 5. Let us run the algorithm on Example 1. Applied to the original set of constraints, the LIA solver returns UNSAT with $\{c_1, c_2\}$ as unsat core. Next we transform each MLIA constraint in the unsat core to LIA. Note that the resulting constraint system becomes equivalent to the system of constraints obtained by eager transformation as presented in Example 3. The resulting constraints are fed to the solver which finds them satisfiable, with $R = \{x = 15, y = 0\}$ as a model. One can easily verify that this is in fact a model of the original constraints over $\mathbb{Z}_m$. Then the algorithm terminates, returning $\langle \mathsf{SAT}, R \rangle$.

Applied instead to the constraints from Example 2, the algorithm finds these satisfiable over $\mathbb{Z}$, returning the model $\{x = 0, y = 9, z = 18\}$. However, the constraint $c_4 : y \leq z$ is not satisfied under this model considering $\mathbb{Z}_{16}$ semantics, as $9 \not\leq 2$. So we replace c_4 in the solver by $\Gamma_1(c_4)$. The resulting system of constraints would still be satisfiable over $\mathbb{Z}$ but would still violate another constraint. We continue transforming MLIA constraints and finally we reach a state where all the constraints are transformed and the LIA solver returns UNSAT, thus proving the original system of constraints unsatisfiable over $\mathbb{Z}_m$. □

Algorithm 2 Solves MLIA constraints using Benders decomposition

function MLIA_Decide_Benders(H)
  Input: A set of linear constraints of the form
    $E_1 \lhd E_2$, with $\lhd \in \{<, =, \leq\}$
  Output: $\langle \mathsf{SAT}, \mathit{Model} \rangle$ or $\langle \mathsf{UNSAT}, \mathit{Unsat\_Core} \rangle$
  $Q \leftarrow \mathit{true}$ ▷ Quotient formula
  $H_r \leftarrow \emptyset$
  for each $c \in H$ **do**
    $H_r \leftarrow H_r \cup \{\Gamma_1(c)\}$
    $Q \leftarrow Q \wedge$ Quotient_Bound(c) ▷ Definition 1
  $H_b \leftarrow H_r$
  while True **do**
    if UNSAT(Q) **then**
      return $\langle \mathsf{UNSAT}, H_b \rangle$
    Let μ_Q be a model of Q ($\mu_Q \models Q$) ▷ Q is satisfiable
    $H_b \leftarrow \{E + m \cdot \mu_Q(q_i) \lhd F + m \cdot \mu_Q(r_i) \mid (E + m \cdot q_i \lhd F + m \cdot r_i) \in H_r\}$
    $\langle S, R \rangle \leftarrow$ LIA_Decide(H_b)
    if $S = \mathsf{SAT}$ **then**
      return $\langle \mathsf{SAT}, R \rangle$
    else ▷ (R is the unsat core in this case)
      $C_R \leftarrow \bigvee\{\sigma(c) \mid c \in R\}$ ▷ (C_R: a cut generated from R, σ: Equation 3)
      $Q \leftarrow Q \wedge C_R$

2. Applying Benders decomposition. Though we can infer tight bounds on quotient variables, the presence of quotient variables with large coefficient (m) in the constraints causes problems for LIA solvers. To avoid this, we adopt a logic-based Benders decomposition strategy. The master problem Q assigns values to the quotient variables, and the subproblem H_b tests whether the chosen quotients can be extended to a model of H. If so, we terminate. Otherwise, we extract a feasibility cut from the unsatisfiable core of H_b. See Algorithm 2.

Feasibility 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 H_b, 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 $\bigvee_i r_i - q_i > \tilde{r}_i - \tilde{q}_i$.

Let σ be the mapping defined as follows:

$$\sigma(E_i + \tilde{q}_i m \lhd F_i + \tilde{r}_i m) = \begin{cases} r_i - q_i > \tilde{r}_i - \tilde{q}_i \text{ if } \lhd \in \{\leq, <\} \\ r_i - q_i \neq \tilde{r}_i - \tilde{q}_i \text{ if } \lhd \in \{=\} \end{cases} \tag{3}$$

Then for an 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 the formula $\bigvee_{i=1}^{n} \sigma(E_i + \tilde{q}_i m \lhd F_i + \tilde{r}_i m)$.

	ITERATION 1	ITERATION 2
	$Q_1 = [-1 \leq q_x \leq 0]$, $\mu_{Q_1} = \{q_x = 0\}$	$Q_2 = Q_1 \wedge [q_x \neq 0]$, $\mu_{Q_1} = \{q_x = -1\}$
H_b	$c_1 : x \geq y \wedge 0 \leq y \leq 15 \wedge 0 \leq x \leq 15 \wedge$ $c_2 : x + 1 + m \cdot 0 = y \ \wedge 0 \leq x + 1 + m \cdot 0 \leq 15 \wedge$ $0 \leq x \leq 15 \wedge 0 \leq y \leq 15$	$c_1 : x \geq y \wedge 0 \leq y \leq 15 \wedge 0 \leq x \leq 15 \wedge$ $c_2 : x - 15 = y \wedge 0 \leq x - 15 \leq 15 \wedge$ $15 \leq x \leq 15 \wedge 0 \leq y \leq 15$
R	$C = \{c_1, c_2\}$, $C_b = [q_x \neq 0]$	$\mu_{H_b} = \{x = 15, y = 0\}$

Fig. 2. Steps performed during Algorithm 2 solving $[x \geq y \wedge x + 1 = y]_{16}$.

Example 6. Consider again solving constraint $[c_1 : x \geq y \wedge c_2 : x + 1 = y]_{16}$, but this time using Algorithm 2. Encoding the constraints as LIA constraints, we obtain

$$\begin{aligned} c_1 : x \geq y \wedge 0 \leq y \leq 15 \wedge 0 \leq x \leq 15 \ \wedge \\ c_2 : x + 1 + m \cdot q_x = y \ \wedge \ 0 \leq x + 1 + m \cdot q_x \leq 15 \ \wedge \\ 0 \leq x \leq 15 \wedge 0 \leq y \leq 15 \end{aligned}$$

The progress of the algorithm is outlined in Figure 2. Computing bounds on q_x, we derive the master problem Q_1. Solving Q_1, we obtain a model $\mu_{Q_1} = \{q_x = 0\}$. Substituting μ_Q into H, we obtain the subproblem H_b. We find H_b is unsatisfiable, with an unsatisfiable core of $\{x + 1 + m \cdot 0_{q_x} = y, x \geq y\}$ (having noted occurrences of q_x). From this, we derive the feasibility cut $\{q_x > 0 \vee q_x < 0\}$ and derive a new master problem. Solving Q_2, we obtain $\mu_{Q_2} = \{q_x = -1\}$. We again substitute into H, obtaining H_{b_2}. Solving H_{b_2}, we obtain a model $\{x = 15, y = 0\}$, and terminate. □

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

3. Applying Benders decomposition lazily. Algorithms 1 and 2 are complementary: the first algorithm transforms constraints lazily, whereas the second deals with large coefficients of quotient variables. We now present Algorithm 3 which in a sense combines them. The first two algorithms are presentation-purpose stepping stones for Algorithm 3 so the reader understands the two innovations separately. We do not evaluate and compare those algorithms.

All three algorithms are guaranteed to terminate. For the lazy approach (Algorithm 1), each iteration causes at least one (of the finitely many) initial constraints to be transformed. For the Benders decomposition, Q has finitely many models (as all variables are integral and bounded), and each iteration adds a cut eliminating at least one model. For the lazy Benders decomposition, each iteration either transforms at least one initial constraint, or eliminates some model of Q (without changing the number of transformed constraints), which again yields a finite descending chain.

Algorithm 3 Solves MLIA constraints using Benders decomposition lazily

function MLIA_Decide_Mix(H)
  Input: A set of linear constraints of the form
    $E_1 \lhd E_2$, with $\lhd \in \{<, =, \leq\}$
  Output: $\langle \mathsf{SAT}, \mathit{Model} \rangle$ or $\langle \mathsf{UNSAT}, \mathit{Unsat\_Core} \rangle$
  $T \leftarrow \emptyset$ ▷ transformed constraints
  $Q \leftarrow \mathit{true}$ ▷ Quotient formula
  $H_r \leftarrow H$
  while True **do**
    if UNSAT(Q) **then**
      return $\langle \mathsf{UNSAT}, H_r \rangle$
    Let μ_Q be a model of Q ($\mu_Q \models Q$) ▷ Q is satisfiable
    $H_b \leftarrow \{E + m \cdot \mu_Q(q_i) \lhd F + m \cdot \mu_Q(r_i) \mid (E + m \cdot q_i \lhd F + m \cdot r_i) \in H_r\}$
    $\langle S, R \rangle \leftarrow$ LIA_Decide(H_b)
    if $S = \mathsf{SAT}$ **then**
      $\langle IS, r \rangle \leftarrow$ MLIA_Model(H, R)
      if $IS = \mathit{Yes}$ **then**
        return $\langle \mathsf{SAT}, R \rangle$
      else
        $H_r \leftarrow H_r \setminus \{r\} \cup \{\Gamma_1(r)\}$; $T \leftarrow T \cup \{\Gamma_1(r)\}$
        $Q \leftarrow Q \wedge$ Quotient_Bound(r) ▷ Definition 1
    else ▷ R is unsat core in this case
      if $R \subseteq T$ **then**
        $C_R \leftarrow \bigvee\{\sigma(c) \mid c \in R\}$ ▷ (C_R: a cut from R, σ: Equation 3)
        $Q \leftarrow Q \wedge C_R$
      else
        for each $s \in R$ **do**
          if $s \notin T$ **then**
            $H_r \leftarrow H_r \setminus \{s\} \cup \{\Gamma_1(s)\}$; $T \leftarrow T \cup \{\Gamma_1(s)\}$
            $Q \leftarrow Q \wedge$ Quotient_Bound(s) ▷ Definition 1

Example 7. Recall again the problem of Example 1, extended with additional variables and constraints:

$$H = [c_1 : x \geq y \wedge c_2 : x + 1 = y \wedge c_3 : z + y \leq 7x \wedge c_4 : w - 2z \leq 3y + 2x]_{16}$$

Under Algorithm 3, the Q is initially trivial, and $H_b = H_r = H$ (as no constraints are initially transformed).

As before, this is unsatisfiable, with a core of $R = \{c_1, c_2\}$. R contains some constraint c_2 which is not yet transformed, so we replace c_2 with $\Gamma(c_2)$, introducing quotient variable q_x and appropriate bounds:

$$\begin{aligned} H' = (c_1 :& x \geq y \wedge c_2' : x + 1 + m \cdot q_x = y \wedge c_3 : z + y \leq 7x \wedge \\ & c_4 : w - 2z \leq 3y + 2x \wedge c_5 : -1 \leq q_x \leq 0 \wedge c_6 : 0 \leq x \wedge y < m) \end{aligned}$$

Re-solving Q, we obtain $\mu_Q = \{q_x = 0\}$. As in Example 6, we find this to be unsatisfiable, having a core of $R = \{c_1, c_2'\}$, yielding a feasibility cut $q_x \neq 0$.

Solving again, we obtain a model $\mu_Q = \{q_x = -1\}$. Solving H_b now gives us a model: $\mu_{H_b} = \{x = 15, y = 0, z = 105, w = 240\}$. Evaluating this model under $\mathbb{Z}_{16}$, we obtain $\mu_H = \{x = 15, y = 0, z = 9, w = 0\}$. As μ_H is a valid model of H, we terminate. □

Deciding Boolean combinations of MLIA constraints. Algorithms 2 and 3 can be extended to support Boolean combinations of MLIA constraints by adding a selection of 'active' constraints to the quotient problem, implicitly enumerating the Boolean skeleton.

5 Implementation and Experiments

Implementation. The algorithms are implemented in SoMoLIA (**So**lver for **Mo**dular **LIA**) which is written in Java. It uses Z3 [14] for input reading and pre-processing, and Gurobi [23] for solving LIA problems. The use of Gurobi is driven by unsatisfiable core extraction; in Z3 this incurs severe performance penalties (several orders of magnitude). Also, the Z3 LIA engine is ill-suited to our problems (optimized for incrementality, not for hard instances). To mitigate unsoundness, we fall back to Z3 if precision limits are exceeded as indicated by Gurobi (e.g., `pspace` instances) or non-integral solutions are obtained. The pre-processing we used includes simple word-level rewriting, constant propagation, Gaussian elimination and elimination of unconstrained variables [9], which are available from Z3 as tactics [14, 15]. Such pre-processing, including the more advanced ones are common to all bit-blasting solvers (see [24]). Our tool supports input in SMT-LIB2 format [3] expressed over `QF_BV` or `QF_LIA`.

Benchmarks. Our main intent is to handle conjunctions of "pure" integer constraints efficiently. Accordingly, we chose a set of 1271 benchmarks from the `CAV_2009`, `dillig`, `check`, `pb2010`, `pidgeons`, `miplib2003` and `cut_lemmas` sub-categories of the `QF_LIA` category of SMT-COMP'16 [11] and interpreted them over $\mathbb{Z}_m$ (not over $\mathbb{Z}$). These are first translated into `QF_BV` logic so that we can reuse the pre-processing for bit-vector formulas as well as compare our results with the bit-vector solvers. Ideally, we would like to evaluate our approach on some challenging benchmarks from the `QF_BV` category, but unfortunately they contain problems with bit-wise operations (`bvand, bvor`, etc.), not purely word level operations (`bvsub, bvmul, bvadd etc.`) supported by SoMoLIA. However we selected the 41 problems from the `pspace` sub-category of the `QF_BV` category (containing only word level operations) and experimented with them. These contain very large bit-vectors (in the order of $\approx$ 23k bits).

Experimental Setting. We have conducted experiments on these benchmarks and compared the results (using Algorithm 3) with four state-of-the-art SMT(BV) solvers (the best of the currently available): Boolector [8] (winner SMT-COMP'16, `QF_BV` main track), Yices2 [16] (winner SMT-COMP'16, `QF_BV` application track),

Table 1. Experimental results. Time (sec) is the average time over # solved instances of each category (timeout 10 min). † indicates SoMoLIA run using Z3 as LIA solver.

Category (#inst)	**Boolector**		**Yices2**		**CVC4**		**Z3**		**SoMoLIA**	
	#correct	Time	#correct	Time	#correct	Time	#correct	Time	#correct	Time
`cut_lemmas` (93)	**93**	14.33	**93**	7.45	91	0.81	**93**	**5.41**	34	10.14
`dillig` (233)	230	3.47	**233**	**0.08**	217	16.73	126	69.00	**233**	0.37
`miplib2003` (16)	**16**	6.25	**16**	9.93	11	208.36	**16**	**3.81**	**16**	5.38
`CAV_2009` (591)	**591**	3.62	**591**	0.45	544	3.71	343	27.51	**591**	**0.34**
`check` (5)	**5**	0.20	**5**	**0.01**	**5**	0.20	**5**	**0.01**	**5**	0.40
`pb2010` (81)	**81**	2.62	**81**	**0.77**	53	161.90	**81**	1.49	**81**	1.71
`pidgeons` (19)	13	15.00	10	4.10	7	4.28	10	32.40	**19**	**0.78**
`pspace` (41)	**41**	155.70	25	92.04	**41**	**0.02**	0	0.00	**41**†	0.34

Z3 [14] and CVC4 [2]. The `pspace` instances use very large moduli, making Gurobi unsound. Therefore, for these instances (and these instances only), SoMoLIA uses Z3 in place of Gurobi as the LIA solver. 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 10 minutes and the memory limit is 6 GB. The benchmarks and the tool itself are available at `http://people.eng.unimelb.edu.au/gkgange/mod-arith/`. The results are summarized in Table 1. The first column indicates the sub-category (and the number of problem instances in that sub-category). The columns that follow present the number of correctly solved instances (#correct) and average time taken (Time) to solve that many instances for the solvers Boolector [8] Yices2 [16], CVC4 [2], Z3 [14] and SoMoLIA respectively. The best result in each sub-category is bold-faced.

Discussion of the results. The different tools are seen to have complementary success or timeout profiles. No solver consistently outperforms the others. We note that SoMoLIA performed well on all sub-categories except `cut_lemmas`, owing to its ability to propagate "pure integer" level information, avoiding *bit-blasting* completely. We find that performance quite remarkable for a tool which is currently nothing more than a proof-of-concept. To us, it shows a great potential of a lazy form of Benders decomposition for this type of constraint problems.

Note that SoMoLIA leads in the `CAV-2009` and `pidgeons` cases, and it is the only solver which solves all the instances from `pidgeons`. The margin is significant in some cases. The problems from the `cut_lemmas` sub-category contain large coefficients, resulting in a large range of values for the quotient variables and requiring many iterations. The computation of unsat-cores also hindered performance. We note that 101 instances were solved by pre-processing alone. However we also note that it made a small number of problems timeout.

The problems from `pspace` sub-category are hard for all *bit-blasting* solvers (since they often make solvers run out of memory). Our approach, which is independent on the size of bit vectors, performed well on this subset and CVC4

performed extremely well. We believe CVC4's strong performance is due to its word level pre-processing of the formula. The approach taken by Zeljic *et al.* [37] also appears to perform well on SAT instances of this subset (albeit not on UNSAT instances as their experiment shows), but unfortunately we were not able to experiment with their tool and include it in the evaluation. A comparison of our approach against other solvers (with pre-processing turned off) allows us to measure the impact of pre-processing as well as the performance of purely bit-blasting approach against ours and we leave this task as future work.

6 Related work

The ubiquity of two-complement semantics has raised considerable interest in the efficient solution of modular arithmetic constraints. Unfortunately, logics over modular arithmetic also resist efficient analysis and decision procedures.

Systems of linear arithmetic equations, as well as systems of linear congruences, can be solved in polynomial time [12]. However, the presence of inequalities hampers tractability; the general integer programming problem is NP-complete in the strong sense. For the special case of sets of integer difference constraints of the form $x - y \leq k$ and $x \leq k$ (as used in our examples), there are well-known efficient algorithms (for example, utilising the Bellman-Ford algorithm [12]), but the modular arithmetic version is already intractable for this special case [5, 21]. In the absence of efficient decision procedures for even restricted subclasses of constraints over modular arithmetic, it is typical to either use general bit-vector solvers, or to simply interpret the problem over $\mathbb{Z}$ (rather than $\mathbb{Z}_m$).

Bit-vector constraints. The theory of bit-vectors offers a natural encoding of modular arithmetic constraints (over modulus 2^w), yet is no more amenable to reasoning. The standard approach to solving bit-vector problems is *bit-blasting*: mapping bit-vector operations down to the corresponding Boolean circuits, and using a SAT solver to solve the resulting constraint system. They implement lazy/eager *bit-blasting* procedures; see Hadarean *et al.* [24] for comparison of these approaches. The most effective SMT(BV) solvers, such as Boolector [32], Yices2 [16], MathSAT [10], Z3 [14] CVC [2] and STP [20] also apply word-level simplification and rewriting, abstraction [32] and presolving over tractable sub-theories [24]. An alternate approach which avoids bit-blasting is to use a constraint programming (CP) approach—maintaining a compact abstraction of feasible assignments for bit-vector variables, and applying word-level filtering algorithms to prune inconsistent assignments. Such techniques have been integrated into pure CP [29] and CLP [1] frameworks. These approaches share the same domain abstraction, tracking which individual bits are fixed to particular values (and therefore have expressiveness equivalent to the bit-blasted representation). They offer compact representations and efficient local pruning, but cannot reason about *relationships* between variables. More recently, hybrid approaches which combine word-level filtering with SAT-style conflict reasoning have also arisen from both CP [36] and SMT [37] lineages.

Integer arithmetic. A common alternative is to simply decide that mathematical integers are "near enough" to the desired semantics, and ignore wrapping effects altogether. Of course this is unsound, but it has the pragmatic advantage of allowing use of existing procedures for reasoning over $\mathbb{Z}$ and $\mathbb{R}$—particularly abstract domains (for static analysis), decision procedures (for constraint solving) and interpolation algorithms (for verification). As a result, interpretation over $\mathbb{Z}$ is a strategy adopted by many abstract interpretation-based static analysis [27, 13] and program verification [26, 22] tools.

LIA encoding. Solving bit-vector constraints by translation into arithmetic constraints (linear and non-linear) is not new and has been studied for a while, though their efficient solving has been a challenge [1, 6, 7, 17, 34, 35]. Bozzano et al. [6] discuss encoding of bit-vector formulae into LIA. Their focus, however, is on linearizing non-linear bit-vector constraints (e.g. bitwise expressions, non-constant multiplication) by decomposing a word v of width w into individual bits $b_{w-1}, \ldots, b_0$ with $w = \sum_i 2^i b_i$, then encoding bit-vector constraints using the introduced b_i variables. This approach effectively emulates bit-blasting with 0–1 variables; it allows handling of a more extensive range of operations, but suffers both the weak linear relaxation discussed in Section 3 and the encoding blow-up of bit-blasting.

7 Conclusion

We have presented a practical and efficient algorithm for solving MLIA constraints and evaluated it on a set of SMT-COMP'16 benchmarks. The main characteristic of the tool, SoMoLIA, is that it utilises Benders decomposition. Importantly, unlike bit-vector solvers, our approach uniformly handles (large) moduli, not necessarily powers of 2, including large primes, and the LIA encoding is bit-width independent. In spite of its increased scope, we find that our proof-of-concept implementation is competitive with state of the art bit-vector solvers, even when benchmarks are restricted to using moduli of form 2^w.

The experimental results are promising though there are many avenues for improvement. For example, additional simplification and pre-solving may lead to significant performance improvements (as has been seen in other solvers). Moreover, the current feasibility cuts are disjunctive, and somewhat weak; methods for deriving stronger cuts should greatly reduce the number of necessary iterations. Embedding this approach in a lazy DPLL(T) framework would provide several advantages: early detection of inconsistent quotient assignments, more efficient handling of Boolean combinations of constraints, and access to complementary theories, such as the theory of arrays.

Currently, we are also exploring ways to extend our work to non-linear bit-vectors problems. We can either encode non-linear bit-vector problems as non-linear integer arithmetic [6] and use non-linear integer solvers or combine linear integer and bit-blasted non-linear constraints and use a solver like IntSat [33], which is good at handling clausal linear constraints.

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. S. Bardin, P. Herrmann, and F. Perroud. An alternative to SAT-based approaches for bit-vectors. In J. Esparza and R. Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems: Proceedings of the 16th International Conference (TACAS'10)*, volume 6015 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 2010.
2. C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. CVC4. In G. Gopalakrishnan and S. Qadeer, editors, *Computer Aided Verification: Proceedings of the 23rd International Conference (CAV'11)*, volume 6806 of *Lecture Notes in Computer Science*, pages 171–177. Springer, 2011.
3. C. Barrett, P. Fontaine, and C. Tinelli. The SMT-LIB Standard: Version 2.5. Technical report, Department of Computer Science, The University of Iowa, 2015. Available at `www.SMT-LIB.org`.
4. J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, Dec. 1962.
5. N. Bjørner, A. Blass, Y. Gurevich, and M. Musuvathi. Modular difference logic is hard, November 2008. Unpublished, arXiv:0811.0987v1.
6. M. Bozzano, R. Bruttomesso, A. Cimatti, A. Franzén, Z. Hanna, Z. Khasidashvili, A. Palti, and R. Sebastiani. Encoding RTL constructs for MathSAT: A preliminary report. *Electronic Notes in Theoretical Computer Science*, 144(2):3–14, 2006.
7. R. Brinkmann and R. Drechsler. RTL-datapath verification using integer linear programming. In *Proceedings of the ASPDAC/VLSI Design Conference 2002*, pages 741–746. IEEE Comp. Soc. Press, 2002.
8. R. Brummayer and A. Biere. Boolector: An efficient SMT solver for bit-vectors and arrays. In S. Kowalewski and A. Philippou, editors, *Tools and Algorithms for the Construction and Analysis of Systems: Proceedings of the 15th International Conference (TACAS'09)*, volume 5505 of *Lecture Notes in Computer Science*, pages 174–177. Springer, 2009.
9. R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, Z. Hanna, A. Nadel, A. Palti, and R. Sebastiani. A lazy and layered SMT(BV) solver for hard industrial verification problems. In W. Damm and H. Hermanns, editors, *Computer Aided Verification: Proceedings of the 19th International Conference (CAV 2007)*, volume 4590 of *Lecture Notes in Computer Science*, pages 547–560. Springer, 2007.
10. A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani. The MathSAT5 SMT solver. In N. Piterman and S. A. Smolka, editors, *Tools and Algorithms for the Construction and Analysis of Systems: Proceedings of the 19th International Conference (TACAS'13)*, volume 7795 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2013.
11. S. Conchon, D. Déharbe, M. Heizmann, and T. Weber. SMT-COMP, 2016. Available online at `http://smtcomp.sourceforge.net/2016/`.

12. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 2009.
13. P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. The ASTREÉ analyzer. In S. Sagiv, editor, *Programming Languages and Systems: Proceedings of the 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes in Computer Science*, pages 21–30. Springer, 2005.
14. L. M. de Moura and N. Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems: Proceedings of the 14th International Conference (TACAS'08)*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.
15. L. M. de Moura and G. O. Passmore. The strategy challenge in SMT solving. In M. P. Bonacina and M. E. Stickel, editors, *Automated Reasoning and Mathematics: Essays in Memory of William W. McCune*, volume 7788 of *Lecture Notes in Computer Science*, pages 15–44. Springer, 2013.
16. B. Dutertre. Yices 2.2. In A. Biere and R. Bloem, editors, *Computer Aided Verification: Proceedings of the 26th International Conference*, volume 8559 of *Lecture Notes in Computer Science*, pages 737–744. Springer, 2014.
17. F. Ferrandi, M. Rendine, and D. Sciuto. Functional verification for SystemC descriptions using constraint solving. In *2002 Design, Automation and Test in Europe Conference and Exposition (DATE'02)*, pages 744–751. IEEE Comp. Soc. Press, 2002.
18. A. Fröhlich, G. Kovásznai, and A. Biere. Efficiently solving bit-vector problems using model checkers. In *SMT Workshop*, 2013.
19. A. Fröhlich, G. Kovásznai, and A. Biere. More on the complexity of quantifier-free fixed-size bit-vector logics with binary encoding. In A. A. Bulatov and A. M. Shur, editors, *Computer Science—Theory and Applications: Proceedings of the 8th International Computer Science Symposium in Russia (CSR'13)*, volume 7913 of *Lecture Notes in Computer Science*, pages 378–390. Springer, 2013.
20. V. Ganesh and D. L. Dill. A decision procedure for bit-vectors and arrays. In W. Damm and H. Hermanns, editors, *Computer Aided Verification: Proceedings of the 19th International Conference (CAV'07)*, volume 4590 of *Lecture Notes in Computer Science*, pages 519–531. Springer, 2007.
21. G. Gange, H. Søndergaard, P. J. Stuckey, and P. Schachte. Solving difference constraints over modular arithmetic. In M. P. Bonacina, editor, *Proceedings of the 24th International Conference on Automated Deduction (CADE'13)*, volume 7898 of *Lecture Notes in Computer Science*, pages 215–230. Springer, 2013.
22. A. Gurfinkel, T. Kahsai, A. Komuravelli, and J. A. Navas. The SeaHorn verification framework. In D. Kroening and C. S. Păsăreanu, editors, *Computer Aided Verification: Proceedings of the 27th International Conference (CAV'15)*, volume 9206 of *Lecture Notes in Computer Science*, pages 343–361. Springer, 2015.
23. Gurobi Optimization, Inc. Gurobi optimizer reference manual, 2016. Available online at `http://www.gurobi.com`.
24. L. Hadarean, K. Bansal, C. Barrett, and C. Tinelli. A tale of two solvers: Eager and lazy approaches to bit-vectors. In A. Biere and R. Bloem, editors, *Computer Aided Verification: Proceedings of the 26th International Conference (CAV'14)*, volume 8559 of *Lecture Notes in Computer Science*, pages 680–695. Springer, 2014.
25. J. N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96(1):33–60, 2003.
26. D. Jackson. *Software Abstractions: Logic, Language and Analysis.* MIT Press, 2006.

27. B. Jeannet and A. Miné. Apron: A library of numerical abstract domains for static analysis. In A. Bouajjani and O. Maler, editors, *Computer-Aided Verification: Proceedings of the 21st International Conference (CAV'09)*, volume 5643 of *Lecture Notes in Computer Science*, pages 661–667. Springer, 2009.
28. G. Kovásznai, H. Veith, A. Fröhlich, and A. Biere. On the complexity of symbolic verification and decision problems in bit-vector logic. In E. Csuhaj-Varjú, M. Dietzfelbinger, and Z. Ésik, editors, *Mathematical Foundations of Computer Science: Proceedings of the 39th International Symposium (MFCS'14)*, volume 8635 of *Lecture Notes in Computer Science*, pages 481–492. Springer, 2014.
29. L. D. Michel and P. V. Hentenryck. Constraint satisfaction over bit-vectors. In *Principles and Practice of Constraint Programming: Proceedings of the 18th International Conference (CP'12)*, volume 7514 of *Lecture Notes in Computer Science*, pages 527–543. Springer, 2012.
30. A. Neumaier and O. Shcherbina. Safe bounds in linear and mixed-integer linear programming. *Mathematical Programming*, 99(2):283–296, 2004.
31. A. Neumaier, O. Shcherbina, W. Huyer, and T. Vinkó. A comparison of complete global optimization solvers. *Mathematical Programming*, 103(2):335–356, 2005.
32. A. Niemetz, M. Preiner, and A. Biere. Boolector 2.0 system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 9:53–58, 2014 (published 2015).
33. R. Nieuwenhuis. The IntSat method for integer linear programming. In B. O'Sullivan, editor, *Principles and Practice of Constraint Programming: Proceedings of the 20th International Conference (CP'14)*, volume 8656 of *Lecture Notes in Computer Science*, pages 574–589. Springer, 2014.
34. G. Parthasarathy, M. K. Iyer, K. Cheng, and L. Wang. An efficient finite-domain constraint solver for circuits. In S. Malik, L. Fix, and A. B. Kahng, editors, *Proceedings of the 41th Design Automation Conference (DAC'04)*, pages 212–217. ACM Publ., 2004.
35. R. Vemuri and R. Kalyanaraman. Generation of design verification tests from behavioral VHDL programs using path enumeration and constraint programming. *IEEE Transactions on VLSI Systems*, 3(2):201–214, 1995.
36. W. Wang, H. Søndergaard, and P. J. Stuckey. A bit-vector solver with word-level propagation. In *Proceedings of the 13th International Conference on Integration of AI and OR Techniques in Constraint Programming (CPAIOR'16)*, volume 9676 of *Lecture Notes in Computer Science*, pages 374–391. Springer, 2016.
37. A. Zeljić, C. M. Wintersteiger, and P. Rümmer. Deciding bit-vector formulas with mcSAT. In N. Creignou and D. Le Berre, editors, *Theory and Applications of Satisfiability Testing (SAT 2016)*, volume 9710 of *Lecture Notes in Computer Science*, pages 249–266. Springer, 2016.