Quantifier Elimination for Linear Modular Constraints

Ajith K John[1] and Supratik Chakraborty[2]

[1] Homi Bhabha National Institute, BARC, Mumbai, India
[2] Dept. of Computer Sc. & Engg., IIT Bombay, India

Abstract. Linear equalities, disequalities and inequalities on fixed-width bit-vectors, collectively called linear modular constraints, form an important fragment of the theory of fixed-width bit-vectors. We present an efficient and bit-precise algorithm for quantifier elimination from conjunctions of linear modular constraints. Our algorithm uses a layered approach, whereby sound but incomplete and cheaper layers are invoked first, and expensive but complete layers are called only when required. We have extended the above algorithm to work with boolean combinations of linear modular constraints as well. Experiments on an extensive set of benchmarks demonstrate that our techniques significantly outperform alternative quantifier elimination techniques based on bit-blasting and Presburger Arithmetic.

Keywords: Quantifier Elimination, Linear Modular Arithmetic

1 Introduction

A first-order theory T is said to admit quantifier elimination (henceforth called QE) if every quantified formula φ in the theory is T-equivalent to a quantifier-free formula ψ. The theory admits *effective* QE if there exists an algorithm that computes ψ on input φ. An example of a theory admitting effective QE is the theory of fixed-width bit-vectors. This theory is extremely important in the context of word-level verification and analysis of hardware and software systems. QE is a key operation in such verification and analysis tasks.

For ease of analysis, words in hardware and software systems are often abstracted as unbounded integers, and QE techniques for integers [5, 6] are used by verification and analysis tools. However the results of verification and analysis using QE for unbounded integers may not be sound [3, 9] if the underlying implementation uses fixed-width bit-vectors. Therefore, bit-precise QE techniques from fixed-width bit-vector constraints is an important problem.

Boolean combinations of linear equalities, disequalities and inequalities on fixed-width bit-vectors, collectively called linear modular constraints, form an important fragment of the theory of fixed-width bit-vectors. Let p be a positive integer constant, $x_1, \ldots, x_n$ be p-bit non-negative integer variables, and $a_0, \ldots, a_n$ be integer constants in $\{0, \ldots, 2^p - 1\}$. A linear term over $x_1, \ldots, x_n$ is a term of the form $a_1 \cdot x_1 + \cdots a_n \cdot x_n + a_0$. A linear modular equality (LME) is a formula of the form $t_1 = t_2 \pmod{2^p}$, where t_1 and t_2 are linear terms over $x_1, \ldots, x_n$. Similarly, a linear modular disequality (LMD) is a formula of the form $t_1 \neq t_2 \pmod{2^p}$, and a linear modular inequality (LMI) is a formula of the form $t_1 \bowtie t_2 \pmod{2^p}$, where $\bowtie \in \{<, \leq\}$. We will use linear modular constraint (LMC) when the distinction between LME, LMD and LMI is not important. Conventionally 2^p is called the modulus of the LMC. Since every variable in an LMC with modulus 2^p represents a p-bit integer, we will assume without loss of generality that whenever we consider a conjunction of LMCs sharing a variable, all the LMCs have the same modulus.

The most dominant technique used in practice for eliminating quantifiers from LMCs is conversion to bit-level constraints (also called bit-blasting [4]), followed by bit-level QE. However this technique scales poorly as the width of bit-vectors increases. In addition, the quantifier-eliminated formula appears more like a propositional logic formula on blasted bits instead of being a bit-vector formula. This reduces the scope for word-level reasoning on the quantifier-eliminated formula if it is used in further reasoning. Since LMCs can be expressed as formulae in Presburger Arithmetic (PA), QE techniques for PA such as Omega Test [6] can also be used to eliminate quantifiers from LMCs. However using PA-based techniques for QE from LMCs scales poorly in practice [4]. Moreover, these techniques destroy the word-level structure of the problem.

We present efficient and bit-precise techniques for QE from LMCs that overcome the above drawbacks in practice. In contrast to bit-blasting and PA-based techniques, our techniques keep the quantifier-eliminated formula in linear modular arithmetic, so that it is amenable to further bit-vector level reasoning.

Our techniques have applications in model checking, program analysis and counterexample guided abstraction refinement (CEGAR) of word-level RTL designs and embedded programs. Symbolic transition relations of word-level RTL designs and embedded programs involve boolean combinations of LMCs. LMEs arise from the assignment statements, whereas LMDs and LMIs arise primarily from branch and loop conditions that compare words/registers. QE from formulae involving symbolic transition relation is the key operation during image computation, computation of strongest post-conditions and computation of predicate abstractions in the verification of such word-level RTL designs and embedded programs. In a CEGAR framework, our techniques can be used to compute abstraction of symbolic transition relation by existentially quantifying out a selected set of variables from the transition relation, and to compute Craig interpolants from spurious counterexamples.

There are two fundamental technical contributions of this work. First, we present a practically efficient and bit-precise algorithm for QE from conjunctions of LMCs called *Project*. Secondly, we extend *Project* for eliminating quantifiers from boolean combinations of LMCs. The work presented here is a collation of our earlier works in [1] and [2]. We have skipped the details of the algorithms and proofs due to lack of space. For interested reader, these details can be found in [1, 2].

2 *Project*: Algorithm for QE from Conjunctions of LMCs

Project uses a layered approach to eliminate quantifiers from a conjunction of LMCs. Sound but incomplete, cheaper layers are invoked first, and expensive but complete layers are called only when required.

2.1 Layer1: Simplifications using LMEs

Layer1 is an extension of the work by Ganesh and Dill [8]. It involves simplification of the given conjunction of LMCs using LMEs present in the conjunction.

For example, consider the problem of computing $\exists x. ((6x+y = 4) \wedge (2x+z \neq 0) \wedge (4x + y \leq 3))$, where all LMCs have modulus 8. Note that $(6x + y = 4)$ can be equivalently expressed as $(2 \cdot 3x = 7y + 4)$. Multiplying both the sides of $(2 \cdot 3x = 7y + 4)$ by the multiplicative inverse of 3 modulo 8, i.e. 3, we get $(2x = 5y+4)$. Replacing the occurrences of $2x$ by $5y+4$, the original problem can be equivalently expressed as $\exists x. ((2x = 5y+4) \wedge (5y+4+z \neq 0) \wedge (2 \cdot (5y+4)+y \leq 3))$. Simplifying modulo 8, we get $(5y+z+4 \neq 0) \wedge (3y \leq 3) \wedge \exists x. (2x = 5y+4)$. Note that $\exists x. (2x = 5y + 4)$ is equivalent to $(4y = 0)$. Hence the result of QE is $(5y + z + 4 \neq 0) \wedge (3y \leq 3) \wedge (4y = 0)$.

Simplifications as above using LMEs present in the conjunction forms the crux of Layer1. It can be observed that Layer1 may not always eliminate the quantifier. For example, consider the problem of computing $\exists x. ((2x + 3y = 4) \wedge (x + y \leq 3))$ with modulus 8. Note that simplifications in Layer1 cannot eliminate the quantifier in this case. Such cases are handled by the following layers which are more expensive.

2.2 Layer2: Dropping Unconstraining LMIs and LMDs

Consider the problem of computing $\exists x. A$ obtained after Layer1, where A is a conjunction of LMCs. Let $A \equiv C \wedge D \wedge I$, where (i) D is a conjunction of (zero or more) LMDs in A, (ii) I is a conjunction of (zero or more) LMIs in A, (iii) C is the conjunction of the remaining LMCs in A, and (iv) $\exists x. (C) \Rightarrow \exists x. (C \wedge D \wedge I)$. Since $\exists x. (C \wedge D \wedge I) \Rightarrow \exists x. (C)$ always holds, this would mean that $\exists x. (C \wedge D \wedge I)$ is equivalent to $\exists x. C$. We say that D and I are "unconstraining" in such cases.

Given $\exists x. (C \wedge D \wedge I)$ satisfying conditions (i), (ii) and (iii) above, Layer2 uses efficiently computable conditions sufficient for condition (iv) to hold. Let $x[i]$ denote the i^{th} bit of the bit-vector x, where $x[0]$ denotes the least significant bit of x. For $i \leq j$, let $x[i : j]$ denote the slice of bit-vector x consisting of bits $x[i]$ through $x[j]$. Let each LMI in I be of the form $s_i \bowtie t_i$, where $\bowtie \in \{\leq, \geq\}$, s_i is a linear term with x in its support, and t_i is a linear term free of x. Let $s_1, \ldots, s_r$ be the distinct linear terms in I with x in their support. We assume without loss of generality that I contains the trivial LMIs $s_i \geq 0$ and $s_i \leq 2^p - 1$ for each linear term s_i. Suppose the LMIs in I are grouped into inequalities of the form $Z_i : u_i \leq s_i \leq v_i$, where u_i denotes the maximum among the lower bounds of s_i in I and v_i denotes the minimum among the upper bounds of s_i in I. Let $k_1, \ldots, k_r$ be the highest powers of 2 in the coefficients of x in $s_1, \ldots, s_r$.

Similarly, let k_0 and k_D be the highest powers of 2 in the coefficients of x in C and D respectively. Suppose further that $k_1 > \ldots > k_r$ and $k_0 > \max(k_D, k_1)$.

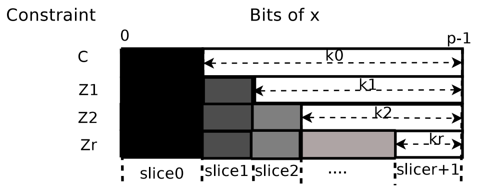

Fig. 1. Slicing of bits of x by $k_0, \ldots, k_r$

We can partition the bits of x into $r+2$ slices as shown in Fig. 1, where slice$_0$ represents $x[0 : p - k_0 - 1]$, slice$_i$ represents $x[p - k_{i-1} : p - k_i - 1]$ for $1 \leq i \leq r$, and slice$_{r+1}$ represents $x[p - k_r : p - 1]$. Note that the value of slice$_0$ potentially affects the satisfaction of C as well as that of Z_1 through Z_r, the value of slice$_i$ potentially affects the satisfaction of Z_i through Z_r for $1 \leq i \leq r$, and the value of slice$_{r+1}$ does not affect the satisfaction of any Z_i or C.

Suppose, given a solution θ_1 of C, there exists a solution θ_2 of $C \wedge Z_1$ that matches θ_1 except possibly in the bits of slice$_1$. In such cases, we say that the solution θ_1 of C can be "engineered" w.r.t. slice$_1$ to satisfy $C \wedge Z_1$. Suppose an arbitrary solution of C can be engineered w.r.t. slice$_1$ to satisfy $C \wedge Z_1$. This would mean that $\exists x.\,(C \wedge Z_1)$ is equivalent to $\exists x.\,C$. Following this argument, if an arbitrary solution of C can be engineered w.r.t. slice$_1$ through slice$_r$ to satisfy $C \wedge Z_1 \wedge \ldots \wedge Z_r$, then $\exists x.\,(C \wedge I)$ is equivalent to $\exists x.\,C$, and I is unconstraining. A similar argument as above can be used to identify unconstraining LMDs. Layer2 computes an efficiently computable *under-approximation* η of the number of ways in which an arbitrary solution of C can be engineered w.r.t. slice$_1$ through slice$_{r+1}$ to satisfy $C \wedge D \wedge I$. If $\eta \geq 1$, then D and I are unconstraining.

For example, consider the problem of computing $\exists x.\,((z = 4x + y) \wedge (6x + y \leq 4) \wedge (x \neq z))$ with modulus 8. Suppose $C \equiv (z = 4x + y)$, $D \equiv (x \neq z)$, and $I \equiv (6x + y \leq 4)$. Note that the bits of x can be partitioned into slice$_0$, slice$_1$ and slice$_2$, where slice$_0$ represents $x[0 : 0]$, slice$_1$ represents $x[1 : 1]$ and slice$_2$ represents $x[2 : 2]$. Slice$_1$ and slice$_2$ do not affect the satisfaction of C. Moreover, it can be observed that an arbitrary solution of C can be engineered w.r.t. slice$_1$ through slice$_2$ to satisfy $C \wedge D \wedge I$. Layer2 computes η as 1 in this case, and thus identifies that $\exists x.\,(C \wedge D \wedge I)$ is equivalent to $\exists x.\,(z = 4x + y)$. Note that $\exists x.\,(z = 4x + y)$ is equivalent to $(4y + 4z = 0)$. Hence the result of QE is $(4y + 4z = 0)$.

2.3 Layer3: Fourier-Motzkin Elimination for LMIs

There are two fundamental problems when trying to apply FM elimination for reals [4] to a conjunction of LMIs. The first step in FM elimination is "normalization" of each inequality l w.r.t. the variable x being quantified. This involves expressing l in an equivalent form $x \bowtie t$, where $\bowtie \in \{\leq, \geq\}$ and t is a term free of x. However, normalizing an LMI w.r.t. a variable is much more difficult than normalizing in the case for reals, since standard equivalences used for normalizing inequalities over reals do not hold in modular arithmetic [3]. Moreover, even if we could normalize LMIs w.r.t. the variable being quantified, due to the lack of density of integers, FM elimination cannot be directly lifted to integers.

Layer3 makes use of a weak normal form for LMIs. We say that an LMI l with x in its support is *normalized w.r.t.* x if it is of the form $a \cdot x \bowtie t$ (*first normal form*), or of the form $a \cdot x \bowtie b \cdot x$ (*second normal form*), where $\bowtie \in \{\leq, \geq\}$, and t is a linear term free of x. A boolean combination of LMCs φ is said to be normalized w.r.t. x if every LMI in φ with x in its support is normalized w.r.t. x.

Given $\exists x. I$, where I is a conjunction of LMIs, Layer3 converts I to an equivalent boolean combination of LMCs normalized w.r.t. x. For example, suppose we wish to normalize $x + 2 \leq y$ modulo 8 w.r.t. x. Consider adding the additive inverse of 2 modulo 8, i.e. 6 to both sides of $x + 2 \leq y$. Let ω_1 be the condition under which the addition of $x + 2$ with 6 overflows the 3-bit representation. Similarly, let ω_2 be the condition under which the addition of y with 6 overflows 3-bit representation. Note that if $\omega_1 \equiv \omega_2$, then $(x+2 \leq y) \equiv (x \leq y+6)$ holds; otherwise $(x+2 \leq y) \equiv (x > y+6)$ holds. $\omega_1 \equiv \omega_2$ can be equivalently expressed as $(x \leq 5) \equiv (y \geq 2)$. Hence, $(x + 2 \leq y)$ can be equivalently expressed in the normalized form $\mathsf{ite}(\varphi, (x \leq y + 6), (x > y + 6))$, where φ denotes $(x \leq 5) \equiv (y \geq 2)$, and $\mathsf{ite}(\alpha, \beta, \gamma)$ denotes $(\alpha \wedge \beta) \vee (\neg\alpha \wedge \gamma)$.

Layer3 applies a variant of FM elimination to achieve QE from the normalized LMIs. We illustrate the idea with help of an example. Consider the problem of computing $\exists x. C$, where $C \equiv (y \leq 4x) \wedge (4x \leq z)$ with modulus 16. Observe that $\exists x. C$ is "the condition under which there exists a multiple of 4 between y and z, where $y \leq z$". It can be shown that $\exists x. C$ is equivalent to the disjunction of the following three conditions: (i) $(y \leq z)$, and y is a multiple of 4, i.e., $(y \leq z) \wedge (4y = 0)$, (ii) $(y \leq z) \wedge (y \leq 12) \wedge (z \geq y+3)$, (iii) $(y \leq z)$, $(z < y+3)$, and $(y > z \pmod 4)$, i.e., $(y \leq z) \wedge (z < y+3) \wedge (4y > 4z)$. In general, suppose we wish to compute $\exists x. (l_1 \wedge l_2)$, where $l_1 : (t_1 \leq a \cdot x)$ and $l_2 : (a \cdot x \leq t_2)$ are LMIs in the *first normal form* w.r.t. x. Let k be the highest power of 2 in the coefficient a of x. Then, $\exists x. (l_1 \wedge l_2)$ is equivalent to $(t_1 \leq t_2) \wedge \varphi$, where φ is the disjunction of the formulas: (i) $(2^{p-k} \cdot t_1 = 0)$, (ii) $(t_2 \geq t_1 + 2^k - 1) \wedge (t_1 \leq 2^p - 2^k)$, and (iii) $(t_2 < t_1 + 2^k - 1) \wedge (2^{p-k} \cdot t_1 > 2^{p-k} \cdot t_2)$.

The conjunction of LMIs such as $(l_1 \wedge l_2)$ above, where all LMIs are in the *first normal form* w.r.t. x, and have the same coefficient of x are said to be "unified" w.r.t. x. Unfortunately, unifying a conjunction of LMIs I w.r.t. x is inefficient in general. Hence we unify I w.r.t. x and apply FM elimination only in the cases where the unification can be done efficiently (the details of unification can be found in [2]). In the other cases, we compute $\exists x. I$ using *model enumeration*, i.e., by expressing $\exists x. I$ in the equivalent form $I|_{x \leftarrow 0} \vee \ldots \vee I|_{x \leftarrow 2^p - 1}$ where $I|_{x \leftarrow i}$ denotes I with x replaced by the constant i.

3 QE from Boolean Combinations of LMCs

We extend *Project* to work with boolean combinations of LMCs using three approaches - a decision diagram (DD) based approach, an SMT-solving based approach and a hybrid approach that combines the strengths of the DD based and the SMT-solving based approaches.

The DD based approach makes use of a data structure called Linear Modular Decision Diagram (LMDD). LMDDs are BDDs [10] with nodes labeled with LMEs or LMIs. They represent boolean combinations of LMCs. Suppose we wish to compute $\exists X.f$, where f is an LMDD over a set of variables V and $X \subseteq V$. A naive algorithm to compute $\exists X.f$ is to apply *Project* to each path in f. However, this algorithm, similar to the Black-box QE algorithm [5] for Linear Decision Diagrams, has running time linear in the number of paths in f. We use an alternate algorithm *QE_LMDD* to compute $\exists X.f$, which is motivated by the White-box QE approach suggested in [5]. *QE_LMDD* makes use of a procedure *QE1_LMDD* that eliminates a single variable x from f. *QE1_LMDD* performs a recursive traversal of the LMDD f. In each recursive call, *QE1_LMDD* computes the LMDD for $\exists x. (g \wedge C_x)$, where g is the LMDD encountered during the traversal and C_x is the conjunction of LMCs containing x encountered in the path from the root node of f to the root node of g. If g is a 1-terminal, then *QE1_LMDD* computes $\exists x. (g \wedge C_x)$ by calling *Project* on $\exists x. C_x$. If the root node of g is a non-terminal, then *QE1_LMDD* first simplifies g using the LMEs in C_x and then traverses g recursively. The single variable elimination strategy gives opportunities for reuse of results through dynamic programming, and in practice significantly outperforms the Black-box QE algorithm.

The SMT-solving based approach is a straightforward extension of the work in [7] for QE from boolean combinations of linear inequalities on reals. Suppose we wish to compute $\exists X. f$, where f is a boolean combination of LMCs over a set of variables V and $X \subseteq V$. A naive way of computing this is by converting f to DNF by enumerating all satisfying assignments, and by invoking *Project* on each conjunction of LMCs in the DNF. We use an algorithm *QE_SMT* which generalizes a satisfying assignment to obtain a conjunction of LMCs, and projects the conjunction of LMCs on the variables in $V \setminus X$. The complement of the projected conjunction of LMCs is conjoined with f before further satisfying assignments are obtained. The interleaving of projection and model enumeration in *QE_SMT* helps in significant pruning of the solution space.

The hybrid approach tries to combine the strengths of the DD based and the SMT-solving based approaches. Suppose we wish to compute $\exists X.f$, where f is an LMDD over a set of variables V and $X \subseteq V$. The hybrid algorithm *QE_Combined* splits $\exists X.f$ into an equivalent disjunction of sub-problems $\bigvee_{i=1}^{n} (\exists X. (f_i \wedge C_i))$, where f_i denotes an internal LMDD node in f and C_i denotes the conjunction of LMCs in the path from the root node of f to f_i. *QE_Combined* now computes $g \equiv \bigvee_{i=1}^{n} (\exists X. (f_i \wedge C_i))$ in the following manner: if $f_i \wedge C_i \wedge \neg g$ is satisfiable, then $h \equiv \exists X. (f_i \wedge C_i)$ is computed using the DD-based approach, and then h is disjoined with g. Computing the sub-problems using the DD-based approach helps in achieving reuse of results through dynamic programming. Unlike

QE_SMT, *QE_Combined* does not explicitly interleave projections inside model enumeration. However disjoining the result of $\exists X.\, (f_i \wedge C_i)$ with g, and computing $\exists X.\, (f_i \wedge C_i)$ only if $f_i \wedge C_i \wedge \neg g$ is satisfiable helps in pruning the solution space of the problem, as achieved in *QE_SMT*.

4 Experiments and Comparison with Existing Software

In order to evaluate the performance of our algorithms and compare them with alternate QE techniques, we used a benchmark suite consisting of a set of *lindd* benchmarks from [5] and a set of *vhdl* benchmarks. Each benchmark is a boolean combination of LMCs with a subset of the variables in their support existentially quantified. The *lindd* benchmarks are boolean combinations of octagonal constraints over integers. These benchmarks are converted to boolean combinations of LMCs by assuming the size of integer as 16 bits. The *vhdl* benchmarks are obtained from transition relation abstraction. We derived the symbolic transition relations of a set of VHDL designs. All the internal variables in these symbolic transition relations are quantified out, which gives abstract transition relations of the vhdl designs.

We measured the time taken by *QE_LMDD*, *QE_SMT*, and *QE_Combined* for QE from each benchmark. We observed that (i) each approach performs better than the others for some benchmarks, (ii) *DD* and *SMT* based approaches are incomparable, and (iii) hybrid approach inherits the strengths of both *DD* and *SMT* based approaches. We also measured the contributions and costs of different layers of *Project* in performing QE from the benchmarks. Layer1 and Layer2 together eliminated 95% of the quantifiers in *lindd* benchmarks and 99.5% of the quantifiers in *vhdl* benchmarks. The remaining quantifiers were eliminated by Layer3. However, none of the benchmarks required model enumeration. Layer1 and Layer2 were cheap (on average, took 1-6 milliseconds per quantifier eliminated). Layer3 was comparatively expensive. On average, Layer3 took 13 seconds per quantifier eliminated for *lindd* benchmarks and 161 milliseconds per quantifier eliminated for *vhdl* benchmarks.

We compared the performance of *Project* with alternate QE techniques. This included comparison of *Project* with PA based QE using Omega Test [6] and with bit-level QE using BDDs [11]. Since Layer1 is a simple extension of the work in [8], we applied Layer1 as a pre-processing step before applying the PA based/ bit-level QE. The procedure that first quantifies out the variables using *Layer1*, and then uses conversion to PA and Omega Test for the remaining variables is called *Layer1_OT*. Similarly, the procedure that first quantifies out the variables using *Layer1*, and then uses bit-blasting and bit-level BDD based QE for the remaining variables is called *Layer1_Blast*. The instances of QE problem for conjunctions of LMCs arising from *QE_SMT* when QE is performed on each benchmark were collected. The procedures *Project*, *Layer1_Blast* and *Layer1_OT* were applied on these instances of the QE problem for conjunctions of LMCs. The results demonstrated that (see Fig.2) *Project* outperforms the alternative QE techniques.

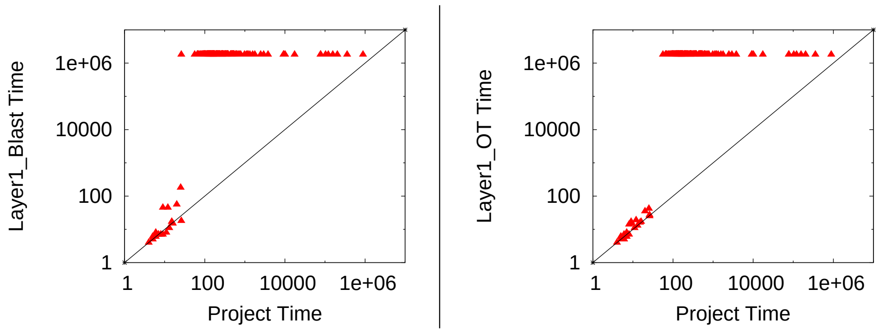

Fig. 2. Plots comparing (a) *Project* and *Layer1_Blast* and (b) *Project* and *Layer1_OT* (All times are in milliseconds)

5 Conclusion

We presented practically efficient and bit-precise techniques for QE from LMCs. Our experiments demonstrate that modular arithmetic based techniques for QE outperform PA and bit-blasting based QE techniques and keep the final result in modular arithmetic.

References

1. A. John, S. Chakraborty. *A quantifier elimination algorithm for linear modular equations and disequations*, In CAV 2011
2. A. John, S. Chakraborty. *Extending quantifier elimination to linear inequalities on bit-vectors*, In TACAS 2013
3. N. Bjørner, A. Blass, Y. Gurevich, M. Musuvathi. *Modular difference logic is hard*, In CoRR abs/0811.0987:(2008)
4. D. Kroening, O. Strichman. *Decision procedures: an algorithmic point of view*, Texts In Theoretical Computer Science, Springer 2008
5. S. Chaki, A. Gurfinkel, O. Strichman. *Decision diagrams for linear arithmetic*, In FMCAD 2009
6. W. Pugh. *The Omega Test: A fast and practical integer programming algorithm for dependence analysis.* Communications of the ACM, Pages 102-114, 1992
7. D. Monniaux. *A quantifier elimination algorithm for linear real arithmetic*, In LPAR 2008
8. V. Ganesh, D. Dill. *A decision procedure for bit-vectors and arrays*, In CAV 2007
9. M. Muller-Olm, H. Seidl. *Analysis of modular arithmetic*, ACM Transactions on Programming Languages and Systems, 29(5):29, 2007
10. R.E. Bryant. *Graph-based algorithms for boolean function manipulation.* IEEE Transactions on Computers, C-35(8):677-691, 1986
11. CUDD release 2.4.2 website, vlsi.colorado.edu/∼fabio/CUDD