Extending Quantifier Elimination to Linear Inequalities on Bit-vectors

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. We present an algorithm for existentially quantifying variables from conjunctions of linear modular equalities (LMEs), disequalities (LMDs) and inequalities (LMIs). We use sound but relatively less complete and cheaper heuristics first, and expensive but more complete techniques are used only when required. Our experiments demonstrate that our algorithm outperforms alternative quantifier elimination techniques based on bit-blasting and Omega Test. We also extend this algorithm to work with Boolean combinations of LMEs, LMDs and LMIs.

1 Introduction

Existential quantifier elimination (henceforth called QE) is the process of transforming a formula containing existential quantifiers into a semantically equivalent quantifier-free formula. This has a number of important applications in formal verification and program analysis, such as computing abstractions of symbolic transition relations, computing strongest postconditions of program statements, computing predicate abstractions and generating code fragments by automatic program synthesis techniques.

Verification and analysis tools often assume unbounded data types like `integer` or `real` for program variables. QE techniques for unbounded data types [4, 8] are therefore often used in program analysis, verification and synthesis. However, a program executing on a machine with fixed-width words really uses fixed-width bit-vector operations. It is known [2, 12] that program analysis assuming unbounded data types may not be sound if the implementation uses fixed-width words, and if overflows are not detected and accounted for. This motivates us to investigate QE techniques for constraints involving fixed-width words. Specifically, we present techniques for QE from Boolean combinations of linear modular (bit-vector) equalities, disequalities and inequalities.

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 (henceforth called 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 (henceforth called LMD) is a formula of the form $t_1 \neq t_2 \pmod{2^p}$, and a Linear Modular Inequality (henceforth called LMI) is a formula of the

form $t_1 \bowtie t_2 \pmod{2^p}$, where $\bowtie \in \{<, \leq\}$. For brevity, we will use "LMC" (for Linear Modular Constraint) when the distinction between LME, LMD and LMI is not important. In the LMCs given above, 2^p is conventionally 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.

In our earlier work [1], we had presented a QE algorithm for Boolean combinations of LMEs and LMDs that is efficient in practice. Unfortunately, techniques for dealing with LMIs involve significantly more technicalities than those for dealing with LMEs and LMDs, and require development of more sophisticated techniques. This paper presents results of our investigations in this direction.

Earlier Work: Efficient procedures for reasoning about LMEs and LMDs were discussed in [1, 10–12]. Bjørner et al [2] showed that the satisfiability problem for conjunctions of difference logic constraints in modular arithmetic is NP-complete. Their work also demonstrated that several intuitive equivalences that hold for inequalities over reals and integers do not necessarily hold for LMIs. QE from a conjunction of LMCs can be achieved by bit-blasting [3], followed by bit-level QE. However this technique irretrievably destroys the word-level structure of the problem, and scales poorly as the width of bit-vectors increases. A QE problem for a conjunction of LMCs can also be presented as a QE problem for a conjunction of inequalities in Integer Linear Arithmetic (ILA) and congruences [7]. Alternatively, each LMC can be reduced to a set of ILA constraints [3], and QE techniques for ILA, such as Omega Test [8], can be used to eliminate integers corresponding to specified bit-vectors. Unfortunately, these techniques have been found to scale poorly in practice [3]. In addition, recovering word-level constraints from the results is often difficult, especially when several variables are quantified. In this paper, we present an alternative approach that tries to overcome most of these drawbacks in practice.

2 QE from a Conjunction of LMCs

Let A denote a conjunction of LMCs over variables $x_1, \ldots, x_n$. We wish to compute a Boolean combination of LMCs, say φ, such that $\varphi \equiv \exists x_1 \cdots \exists x_t.\, A$. Let us initially focus on the simpler problem of existentially quantifying a single variable from a conjunction of LMCs. For clarity of exposition, we use x to denote the variable to be quantified

Notation and Preliminaries: To simplify notation, we assume that all LMCs in the remainder of the paper have modulus 2^p for some positive integer p, unless stated otherwise. We use letters x, y, z, x_1, $x_2, \ldots$ to denote variables, use a, a_1, $a_2, \ldots, b, b_1, b_2, \ldots$ to denote constants, and use s, s_1, $s_2, \ldots, t, t_1, t_2, \ldots$ to denote linear terms. The letters d, d_1, $d_2, \ldots$ are used to denote LMDs, l, l_1, $l_2, \ldots$ are used to denote LMIs, and c, c_1, $c_2, \ldots$ are used to denote LMCs. Furthermore, we use D, D_1, $D_2, \ldots$ to denote conjunctions of LMDs, I, I_1, $I_2, \ldots$ to denote conjunctions of LMIs, and $C, C_1, C_2, \ldots, A, A_1, A_2, \ldots$ to denote conjunctions of

LMCs. For a linear term t, we use $-t$ to denote the additive inverse of t modulo 2^p.

Since $(t_1 < t_2)$ is semantically equivalent to both $(t_2 \geq 1) \wedge (t_1 \leq t_2 - 1)$ and $(t_1 \leq 2^p - 2) \wedge (t_1 + 1 \leq t_2)$, there is no loss of generality in assuming that LMIs are restricted to be of the form $t_1 \leq t_2$. However, for clarity of exposition, we allow LMIs of the form $t_1 < t_2$, whenever convenient. An LME or LMD $t_1 \bowtie t_2$, where $\bowtie \in \{=, \neq\}$, can be equivalently expressed as $2^\mu \cdot x \bowtie t$, where t is a linear term free of x, and μ is an integer such that $0 \leq \mu \leq p$ (see [1]). Note that this does not sacrifice generality since we can set μ to p if the LME/LMD is free of x.

For every linear term t_1 and variable x, we define $\kappa(x, t_1)$ to be an integer in $\{0, \ldots, p\}$ such that t_1 is equivalent to $2^{\kappa(x,t_1)} \cdot e \cdot x + t$, where t is a linear term free of x, and e is an odd number in $\{1, \ldots, 2^p - 1\}$. Note that if t_1 is free of x, then $\kappa(x, t_1) = p$. The definition of $\kappa(x, \cdot)$ can be extended to (conjunctions of) LMCs as follows. Let c be an LME/LMD equivalent to $2^\mu \cdot x \bowtie t$, where $\bowtie \in \{=, \neq\}$ and t is free of x. We define $\kappa(x, c)$ to be μ in this case. If t_1, t_2 are linear terms, then $\kappa(x, t_1 \leq t_2)$ is defined to be $min(\kappa(x, t_1), \kappa(x, t_2))$. Finally, if $c_1, \ldots, c_m$ are LMCs, then $\kappa(x, \bigwedge_{i=1}^{m} (c_i))$ is defined to be $\min_{i=1}^{m}(\kappa(x, c_i))$. Observe that if C is a conjunction of (possibly one) LMCs and if $\kappa(x, C) = k$, then only the least significant $p - k$ bits of x affect the satisfaction of C. We will say that x is in the support of C if $\kappa(x, C) < p$.

Lemma 1. *Let A be a conjunction of LMCs containing at least one LME. Let $2^{k_1} \cdot x = t_1$ be the LME with the minimum $\kappa(x, \cdot)$ value among the LMEs in A. Then $\exists x.\, A \equiv C_1 \wedge \exists x.\, C_2$, where C_1 is a conjunction of LMCs free of x, and C_2 is a conjunction of $2^{k_1} \cdot x = t_1$ and (possibly zero) LMIs and LMDs, each of which has $\kappa(x, \cdot)$ less than k_1.*

We omit the proof of this and other lemmas due to space constraints. The reader is referred to [14] for all proofs.

Example: All LMCs in this example have modulus 8. Consider the problem of computing $\exists y.\, ((2^1 y = 5x + 2) \wedge (2^0 y \neq 6x + 7z) \wedge (2^0 \cdot 5y + z \leq 2^1 y) \wedge (2^1 \cdot 3y \leq x + z))$. This can be equivalently expressed as $\exists y.\, ((2y = 5x + 2) \wedge (y \neq 6x + 7z) \wedge (5y + z \leq 5x + 2) \wedge (3 \cdot (5x + 2) \leq x + z))$. Simplifying modulo 8, we get $(7x + 6 \leq x + z) \wedge \exists y.\, ((2y = 5x + 2) \wedge (y \neq 6x + 7z) \wedge (5y + z \leq 5x + 2))$. Note that the result is of the form $C_1 \wedge \exists x.\, C_2$, as specified in Lemma 1.

Our QE algorithm for conjunctions of LMCs uses a layered approach. Relatively less complete but sound and cheap heuristics are invoked first, and more complete but expensive techniques are used only when required. We now outline heuristic *QE1_Layer1* that forms the crux of the first (and also the cheapest) layer. Given a conjunction of LMCs A and a variable x to be quantified, *QE1_Layer1* computes $\exists x.\, A$ as $C_1 \wedge \exists x.\, C_2$ based on Lemma 1. If the $\kappa(x, \cdot)$ of all LMDs and LMIs in A are at least as large as k_1 (as in Lemma 1), then C_2 consists of the single LME $2^{k_1} \cdot x = t_1$. In this case, $\exists x.\, C_2$ simplifies to $2^{p-k_1} \cdot t_1 = 0$, and *QE1_Layer1* suffices to compute $\exists x.\, A$. However, in general,

C_2 may contain LMDs and LMIs with $\kappa(x, \cdot)$ values less than k_1. We describe techniques to address such cases in the following subsections.

2.1 Identifying Unconstraining LMIs and LMDs

Our goal in this subsection is to express C_2, obtained after application of *QE1_Layer1*, as $C \wedge D \wedge I$, where (i) D is a conjunction of (zero or more) LMDs in C_2, (ii) I is a conjunction of (zero or more) LMIs in C_2, (iii) C is the conjunction of the remaining LMCs in C_2, 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 allow us to compute $\exists x.\,C_2$, or equivalently $\exists x.\,(C \wedge D \wedge I)$, as $\exists x.\,C$. We call D and I as "unconstraining" LMDs and LMIs, respectively, in such cases.

Given C, D and I satisfying conditions (i), (ii) and (iii) above, we first focus on finding sufficient and efficiently checkable conditions for condition (iv) to hold. Let $x[i]$ denote the i^{th} bit of a bit-vector x, where $x[0]$ denotes its least significant bit. For $i \leq j$, let $x[i:j]$ denote the slice of bit-vector x consisting of bits $x[i]$ through $x[j]$. Given slice $x[i:j]$, its value is the natural number encoded by the bits in the slice. A key notion used in the subsequent discussion is that of "engineering" a solution of a constraint to make it satisfy another constraint. Formally, we say that a solution θ_1 of a conjunction φ of LMCs can be engineered with respect to slice $x[i:j]$ to satisfy a (possibly different) conjunction ψ of LMCs if there exists a solution θ_2 of ψ that matches θ_1 except possibly in the bits of slice $x[i:j]$. The central idea in the second layer of our QE algorithm is to efficiently compute an under-approximation η of the number of ways in which an arbitrary solution of C can be engineered to satisfy $C \wedge D \wedge I$. It is easy to see that if $\eta \geq 1$, then $\exists x.\,(C) \Rightarrow \exists x.\,(C \wedge D \wedge I)$.

Let I be $\bigwedge_{i=1}^{n}(l_i)$, where each l_i is an LMI of the form $s_i \bowtie t_i$, the operator $\bowtie$ is in $\{\leq, \geq\}$, s_i is a linear term with x in its support, and t_i is a linear term free of x. Note that this implies some loss of generality, since we disallow LMIs of the form $s \bowtie t$, where both s and t have x in their support. However, our experiments indicate that this is not very restrictive in practice. Let $s_1, \ldots, s_r$ be the distinct linear terms in I with x in their support. We partition I into $I_1, \ldots, I_r$, where each I_j is the conjunction of only those LMIs in I that contain the linear term s_j. We assume without loss of generality that each I_j contains the trivial LMIs $s_j \geq 0$ and $s_j \leq 2^p - 1$. Let I_j have n_j LMIs, of which the first $m_j (< n_j)$ are of the form $s_j \geq t_q$, where $1 \leq q \leq m_j$. Let the remaining LMIs in I_j be of the form $s_j \leq t_q$, where $m_j + 1 \leq q \leq n_j$.

Consider the inequality $Z_j : u_j \leq s_j \leq v_j$, where u_j denotes $\max_{q=1}^{m_j}(t_q)$ and v_j denotes $\min_{q=m_j+1}^{n_j}(t_q)$. Although Z_j is not a LMI, it is semantically equivalent to I_j. For notational convenience, let us denote $\kappa(x, s_j)$ by k_j. Clearly, the value of slice $x[p - k_j : p - 1]$ does not affect the satisfaction of Z_j. We wish to compute the number of ways, say N_j, in which an arbitrary solution of C can be engineered with respect to slice $x[0 : p - k_j - 1]$ to satisfy Z_j. Towards this end, we compute an integer δ_j in $\{0, \ldots, 2^p - 1\}$ such that $\delta_j \leq v_j - u_j + 1$. Intuitively, δ_j represents the minimum number of *consecutive* values that s_j can

take for every combination of values of other variables, if we were to treat s_j as a fresh p-bit variable and if Z_j were to be satisfied. In general, however, s_j is of the form $a_j \cdot x + w_j$, where w_j is a linear term free of x, and a_j is a multiple of 2^{k_j}. Therefore, for every combination of values of variables other than x, there exist at least $\lfloor \delta_j / 2^{k_j} \rfloor$ consecutive values that $x[0 : p - k_j - 1]$ can take while satisfying Z_j. Hence, $N_j \geq \lfloor \delta_j / 2^{k_j} \rfloor$. For notational convenience, let us denote $\lfloor \delta_j / 2^{k_j} \rfloor$ by $\widehat{N_j}$.

To understand how δ_j is computed, recall that for every g in $\{1 \ldots m_j\}$ and for every h in $\{m_j+1 \ldots n_j\}$, we have $t_g \leq s_j \leq t_h$. For every such pair of indices g and h, let $\delta_{g,h}$ be an integer in $\{0, \ldots, 2^p - 1\}$ such that $\delta_{g,h} \leq t_h - t_g + 1$. The value of δ_j can then be obtained as the minimum of all $\delta_{g,h}$ values. For reasons of simplicity and efficiency, we compute the values of $\delta_{g,h}$ conservatively as follows: (i) if t_g and t_h are constants, then $\delta_{g,h} = \max(t_h - t_g + 1, 0)$, (ii) if t_h is a constant and t_g can be expressed as $2^\tau \cdot t$, where $\tau \in \{0, 1, \ldots, p-1\}$, then $\delta_{g,h} = \max(t_h - (2^p - 2^\tau) + 1, 0)$, (iii) if t_g is a constant and t_h can be expressed as $2^\tau \cdot t + a$, where $\tau \in \{0, 1, \ldots, p-1\}$, then $\delta_{g,h} = \max(a \bmod 2^\tau - t_g + 1, 0)$, and (iv) $\delta_{g,h} = 0$ otherwise.

Let D be $\bigwedge_{i=1}^{m}(d_i)$, where each d_i is an LMD. Let k_0 denote $\kappa(x, C)$, and let C be such that k_0 is greater than both $\max_{i=1}^{m} \kappa(x, d_i)$ and $\max_{j=1}^{r} k_j$ (recall that $k_j = \kappa(x, s_j)$). To simplify the exposition, suppose further that $k_1 > \ldots > k_r$. We 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_j represents $x[p - k_{j-1} : p - k_j - 1]$ for $1 \leq j \leq r$, 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_j potentially affects the satisfaction of Z_j through Z_r for $1 \leq j \leq r$, and the value of slice_{r+1} does not affect the satisfaction of any Z_j or C.

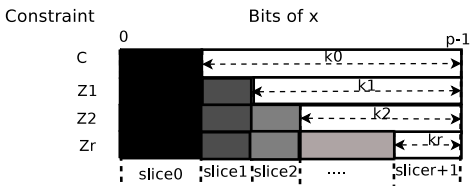

Fig. 1. Slicing of bits of x by $k_0, \ldots, k_r$

We have already seen that for every combination of values of variables other than x, there exist at least $\widehat{N_j}$ consecutive values that can be assigned to $x[0 : p - k_j - 1]$, while satisfying Z_j. Thus, if Z_0 denotes True, and if θ is a solution of $C \wedge \bigwedge_{i=0}^{j-1} Z_i$, where $0 \leq i < j \leq r$, then there exist at least $\lfloor \widehat{N_j} / 2^{p-k_i} \rfloor$ consecutive values that can be assigned to the slice $x[p - k_i : p - k_j - 1]$ while satisfying Z_j. Since slice_0 through slice_i are unchanged, each such engineered solution must also satisfy $C \wedge \bigwedge_{i=0}^{j-1} Z_i$.

Let $Y_{i,j}$ denote the number of ways in which an arbitrary solution of $C \wedge \bigwedge_{i=0}^{j-1} Z_i$ can be engineered with respect to bits in slice_{i+1} through slice_j, to satisfy $C \wedge \bigwedge_{i=0}^{j} Z_i$. By the argument given above, $Y_{i,j} \geq \lfloor \widehat{N_j} / 2^{p-k_i} \rfloor$, and the values of $x[p - k_i : p - k_j - 1]$ in the corresponding engineered solutions are consecutive. The latter fact implies that if we focus only on slice_{i+1}, then there are at least $\min(\lfloor \widehat{N_j} / 2^{p-k_i} \rfloor, 2^{k_i - k_{i+1}})$ consecutive values of slice_{i+1} in the cor-

responding engineered solutions. Note that the min expression is necessary since slice_{i+1} can only have one of $2^{k_i-k_{i+1}}$ distinct values. For notational convenience, let us denote $\min(\lfloor \widehat{N_j}/2^{p-k_i} \rfloor, 2^{k_i-k_{i+1}})$ by $\alpha_{i,j}$.

The above argument indicates that a solution θ of $C \wedge \bigwedge_{i=0}^{j-1} Z_i$ can be engineered to satisfy $C \wedge \bigwedge_{i=0}^{j} Z_i$ by using at least $\alpha_{i,j}$ different consecutive values of slice_{i+1}, for $0 \leq i < j \leq r$. Let the corresponding set of values of slice_{i+1} be denoted $S^{\theta}_{i+1,j}$. If $\bigcap_{j=i+1}^{r} S^{\theta}_{i+1,j}$ is non-empty, there exists a common value of slice_{i+1} that permits us to engineer θ with respect to slice_{i+1} through slice_r to satisfy Z_{i+1} through Z_r, respectively. It is therefore desirable to have $|\bigcap_{j=i+1}^{r} S^{\theta}_{i+1,j}| \geq 1$. Using the Inclusion-Exclusion principle, we find that $|\bigcap_{j=i+1}^{r} S^{\theta}_{i,j}| \geq (\sum_{j=i+1}^{r} \alpha_{i,j}) - (r-i-1) \cdot 2^{k_i-k_{i+1}}$. Note that the lower bound is independent of θ. For notational convenience, let us denote the lower bound by W_i.

If $W_i \geq 1$ for all $i \in \{1, \ldots r\}$, an arbitrary solution θ of C can be engineered to satisfy $C \wedge \bigwedge_{i=1}^{r} Z_i$ as follows. Since $W_1 \geq 1$, we choose a value of slice_1, say v_1, from $\bigcap_{j=1}^{r} S^{\theta}_{1,j}$. Let θ_1 denote θ with slice_1 (possibly) changed to have value v_1. Then θ_1 satisfies $C \wedge Z_1$. Since $W_2 \geq 1$, we can now choose a value of slice_2, say v_2, from $\bigcap_{j=2}^{r} S^{\theta_1}_{2,j}$, and repeat the procedure until we have chosen values for slice_1 through slice_r. Finally, since slice_{r+1} does not affect the satisfaction of C or of any Z_i, we can choose an arbitrary value for slice_{r+1}. Clearly, there are at least $(\prod_{i=0}^{r-1} |W_i|) \cdot 2^{k_r}$ ways in which values of different slices can be chosen, so as to engineer θ to satisfy $C \wedge \bigwedge_{i=1}^{r} Z_i$. Let us denote $(\prod_{i=0}^{r-1} |W_i|) \cdot 2^{k_r}$ by μ_I.

For every combination of values of variables other than x, let μ_D be an over-approximation of the number of values that can be assigned to slice_0 through slice_{r+1} such that D is violated. As shown in [1], $\mu_D = \sum_{i=1}^{m} (2^{\kappa(x,d_i)})$. Thus, we have at least $\mu_I - \mu_D$ ways of assigning values to slice_1 through slice_{r+1} when engineering a solution of C to satisfy $C \wedge D \wedge \bigwedge_{i=1}^{r} Z_i$. The details of extending these ideas to the general case, where $k_1 \geq \ldots \geq k_r$ can be found in [14].

Lemma 2. *If $\eta = \mu_I - \mu_D \geq 1$, then $\exists x.\,(C \wedge D \wedge I) \equiv \exists x.\,(C)$*

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 $I \equiv (6x + y \leq 4)$. Here $p = 3$, $k_0 = 2$, $k_1 = 1$, $r = 1$, $\delta_1 = 5$, and $\mu_D = 1$. Therefore $W_0 = \alpha_{0,1} = Y_{0,1} = 1$, and $\mu_Z = |W_0| \cdot 2^1 = 2$. Hence $\eta = 1$, which implies that $\exists x.\,(C \wedge D \wedge I) \equiv \exists x.\,(C)$.

We now present procedure *QE1_Layer2*, that applies the technique described above to problem instances of the form $\exists x.\,C_2$, obtained after invoking *QE1_Layer1*. *QE1_Layer2* initially expresses $\exists x.\,C_2$ as $\exists x.\,(C \wedge D \wedge I)$, where C denotes $2^{k_1} \cdot x = t_1$ and $D \wedge I$ denotes the conjunction of LMDs and LMIs in C_2. If η (as in Lemma 2) is at least 1, then $D \wedge I$ is dropped from C_2. Otherwise, the LMCs in $D \wedge I$ with the largest $\kappa(x, \cdot)$ value (i.e. LMCs whose satisfaction depends on the least number of bits of x) are identified and included in C, and the above process repeats. If all the LMIs and LMDs in $\exists x.\,C_2$ are dropped in this manner, then $\exists x.\,C_2$ reduces to $\exists x.\,(2^{k_1} \cdot x = t_1)$, and *QE1_Layer2* can return the equivalent form $2^{p-k_1} \cdot t_1 = 0$. Otherwise, *QE1_Layer2* returns $\exists x.\,C_3$,

where C_3 is a conjunction of possibly fewer LMCs compared to C_2, such that $\exists x.\, C_3 \equiv \exists x.\, C_2$. The next subsection describes techniques to eliminate quantifiers from such problem instances.

2.2 Fourier-Motzkin Elimination for LMIs

In this subsection, we present a Fourier-Motzkin (FM) style QE technique for conjunctions of LMIs. There are two obvious problems when trying to apply FM elimination for reals [3] to a conjunction of LMIs. Recall that FM elimination "normalizes" each inequality l w.r.t. the variable x being quantified by 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 requires greater care, since standard equivalences used for normalizing inequalities over reals do not carry over to LMIs [2]. Moreover, due to the lack of density of integers, FM elimination cannot be directly lifted to normalized LMIs. This motivates us to (i) define a weak normal form for LMIs, and (ii) adapt FM elimination to achieve QE from normalized LMIs.

Note that Omega Test [8] also defines a normal form for inequalities over integers, and adapts FM elimination over reals for QE from normalized inequalities over integers. However, our experiments indicate that our approach convincingly outperforms Omega Test.

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 $a \cdot x \bowtie t$, or of the form $a \cdot x \bowtie b \cdot x$, where $\bowtie \in \{\leq, \geq\}$, and t is a linear term free of x. We will henceforth use $NF1$ to refer to the first normal form $(a \cdot x \bowtie t)$ and $NF2$ to refer to the second normal form $(a \cdot x \bowtie b \cdot x)$. 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.

We will now show that every LMI with x in its support can be equivalently expressed as a Boolean combination of LMCs normalized w.r.t. x. Before going into the details of normalizing LMIs, it would be useful to introduce some notation. We define $\Omega(t_1, t_2)$ as the condition under which $t_1 + t_2$ overflows a p-bit representation, i.e., $t_1 + t_2$ interpreted as an integer exceeds $2^p - 1$. Note that $\Omega(t_1, t_2)$ is equivalent to both $(t_2 \neq 0) \wedge (t_1 \geq -t_2)$ and $(t_1 \neq 0) \wedge (t_2 \geq -t_1)$.

Suppose we wish to normalize $x + 2 \leq y$ modulo 8 w.r.t. x. Noting that 6 is the additive inverse of 2 modulo 8, if $\Omega(x + 2, 6) \equiv \Omega(y, 6)$, then $(x + 2 \leq y) \equiv (x \leq y + 6)$ holds; otherwise $(x + 2 \leq y) \equiv (x > y + 6)$ holds. Note that $\Omega(x + 2, 6) \equiv \Omega(y, 6)$ 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 $\mathsf{ite}(\alpha, \beta, \gamma)$ is a shorthand for $(\alpha \wedge \beta) \vee (\neg\alpha \wedge \gamma)$. The Ω predicate thus allows us to perform a case-split and normalize each branch. The following Lemma generalizes this idea.

Lemma 3. *Let $l_1 : (a \cdot x + t_1 \leq b \cdot x + t_2)$ be an LMI, where t_1 and t_2 are linear terms without x in their supports. Then, $l_1 \equiv \mathsf{ite}(\varphi, l_2, \neg l_2)$, where $l_2 \equiv (a \cdot x - b \cdot x \leq t_2 - t_1)$, and φ is a Boolean combination of LMCs normalized w.r.t. x.*

Modified FM for normalized LMIs: We begin by illustrating the primary idea through an example. Consider the problem of computing $\exists x.\, C$, where $C \equiv (y \leq 4x) \wedge (4x \leq z)$ with modulus 16. Note 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 true iff one of the following three conditions holds: (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., $(y \leq z) \wedge (z < y+3) \wedge (4y > 4z)$. Hence $\exists x.\, C$ is equivalent to $(y \leq z) \wedge \varphi$, where φ is the disjunction of the following three formulas: (i) $(4y = 0)$, (ii) $(z \geq y+3) \wedge (y \leq 12)$, (iii) $(z < y+3) \wedge (4y > 4z)$. Note that if x, y, z were reals, we would have obtained $(y \leq z)$ for $\exists x.\, C$. However, this would over-approximate $\exists x.\, C$ in the case of fixed width bit-vectors. The following Lemma generalizes this idea.

Lemma 4. *Let $l_1 : (t_1 \leq a \cdot x)$ and $l_2 : (a \cdot x \leq t_2)$ be LMIs in NF1 w.r.t. x. Let k be $\kappa(x, a \cdot x)$. Then, $\exists x.\, (l_1 \wedge l_2) \equiv (t_1 \leq t_2) \wedge \varphi$, 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) $(t_2 < t_1 + 2^k - 1) \wedge (2^{p-k} \cdot t_1 > 2^{p-k} \cdot t_2)$.*

Suppose we wish to compute $\exists x.\, I$, where I is a conjunction of LMIs normalized w.r.t. x. Let $I \equiv I_1 \wedge I_2$, where I_1 is the conjunction of LMIs in I that are in *NF1*, and I_2 is the conjunction of LMIs in I that are in *NF2*. In addition, let $a_1, \ldots, a_n$ be the distinct non-zero coefficients of x in LMIs in I_1, and let $I_{1,i}$ denote the conjunction of LMIs in I_1 in which the coefficient of x is a_i. Finally, let $\Delta(t_1, t_2, k)$ denote the result of computing $\exists x.\, ((t_1 \leq a \cdot x) \wedge (a \cdot x \leq t_2))$ using Lemma 4, where k denotes $\kappa(x, a \cdot x)$. It is easy to see that Lemma 4 can be used to compute $\exists x.\, I_{1i}$, for every $i \in \{1, \ldots n\}$. Similar to FM elimination, we partition the LMIs $l_{i,j} : a_i \cdot x \bowtie t_j$ in I_{1i} into two sets $\Lambda_{\leq}$ and $\Lambda_{\geq}$, where $\Lambda_{\bowtie} = \{l_{i,j} \mid l_{i,j} \text{ is of the form } a_i \cdot x \bowtie t_j\}$, for $\bowtie \in \{\leq, \geq\}$. We assume without loss of generality that the trivial LMIs $a_i \cdot x \leq 2^p - 1$ and $a_i \cdot x \geq 0$ are present in $\Lambda_{\leq}$ and $\Lambda_{\geq}$ respectively. We can now compute $\exists x.\, I_{1i}$ as $\bigwedge_{(a_i \cdot x \leq t_p) \in \Lambda_{\leq},\, (a_i \cdot x \geq t_q) \in \Lambda_{\geq}} (\Delta(t_q, t_p, \kappa(x, a_i \cdot x)))$.

Each conjunction of LMIs such as I_{1i} above, where all LMIs are in *NF1* w.r.t. x, and have the same coefficient of x are said to be "unified" w.r.t. x. A Boolean combination of LMCs φ is said to be unified w.r.t. x if all LMIs in φ with x in their support are in *NF1* w.r.t. x and have the same coefficient of x. Unfortunately, unifying I w.r.t. x is inefficient in general. Hence we propose unifying I w.r.t. x only in the following cases, where unification can be done efficiently: (a) $I_2 \equiv \mathsf{true}$, $n = 2$ and $a_2 = -a_1$, or (b) $I_2 \equiv \mathsf{true}$, and every a_i is of the form $2^{k_i} \cdot e$, where e is an odd number in $\{1, \ldots, 2^p - 1\}$ independent of i. In case (a) above, I can be equivalently expressed as a Boolean combination of LMCs unified w.r.t. x, by replacing each occurrence of a_2 by $-a_1$ using the equivalence $(-t_1 \leq -t_2) \equiv (t_1 = 0) \vee ((t_2 \neq 0) \wedge (t_1 \geq t_2))$. Case (b) deserves some additional explanation.

Consider the problem of computing $\exists x.\, I$, where $I \equiv (y \leq 2x) \wedge (x \leq z)$ with modulus 8. It can be shown that $x \leq z$ can be equivalently expressed as the disjunction of (i) $\Omega(x, x) \wedge \Omega(z, z) \wedge (2x \leq 2z)$, (ii) $\neg\Omega(x, x) \wedge \neg\Omega(z, z) \wedge (2x \leq 2z)$, and (iii) $\neg\Omega(x, x) \wedge \Omega(z, z)$. Hence, $\exists x.\, I$ can be equivalently expressed as $\exists x.\, \varphi'$,

where φ' is the disjunction of (i) $\Omega(x,x) \wedge \Omega(z,z) \wedge (2x \leq 2z) \wedge (y \leq 2x)$, (ii) $\neg\Omega(x,x) \wedge \neg\Omega(z,z) \wedge (2x \leq 2z) \wedge (y \leq 2x)$, and (iii) $\neg\Omega(x,x) \wedge \Omega(z,z) \wedge (y \leq 2x)$. Note that $\Omega(x,x)$ and $\Omega(z,z)$ can be equivalently expressed as $x \geq 4$ and $z \geq 4$ respectively. However, on closer inspection, it can be seen that occurrences of $x \geq 4$ in $\exists x.\, \varphi'$ arising from $\Omega(x,x)$ are unconstraining, and can therefore be dropped. Thus $\exists x.\, \varphi'$ can be equivalently expressed as $\exists x.\, \varphi$, where φ is the disjunction of $(2x \leq 2z) \wedge (y \leq 2x)$ and $(z \geq 4) \wedge (y \leq 2x)$. Note that $\exists x.\, \varphi$ is equivalent to $\exists x.\, I$ and is unified w.r.t. x. In general, given $\exists x.\, I$ such that $I_2 \equiv \mathsf{true}$ and the a_i's have the same e (as defined above), we make use of the above idea for unifying I w.r.t. x such that $\max_{i=1}^{n}(a_i)$ is the coefficient of x in all LMIs involving x. More details can be found in [14]. Note that normalizing and unifying a given conjunction of LMIs w.r.t. a variable converts it to a Boolean combination of LMCs in general. We make use of one of the techniques in section 3 for eliminating quantifiers from such Boolean combinations of LMCs.

In cases other than those covered in (a) and (b) above, we propose computing $\exists x.\, I$ 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.

The procedure that computes $\exists x.\, C_3$ (where C_3 is obtained from *QE1_Layer2*) using techniques mentioned in this subsection is called *QE1_Layer3*. Initially, LMEs and LMDs in C_3 are converted to LMIs using the equivalences $(t_1 = t_2) \equiv (t_1 \geq t_2) \wedge (t_1 \leq t_2)$ and $(t_1 \neq t_2) \equiv \neg(t_1 = t_2)$. Subsequently, $\exists x.\, C_3$ is computed either by normalizing and unifying C_3 w.r.t. x, followed by QE from the resulting Boolean combination of LMCs, or by model enumeration.

Recall that *QE1_Layer1*, *QE1_Layer2*, and *QE1_Layer3* try to eliminate a single quantifier from a conjunction of LMCs. These can be easily extended to eliminate multiple quantifiers by invoking them iteratively. Thus we have procedures *Layer1*, *Layer2*, and *Layer3* - extensions of *QE1_Layer1*, *QE1_Layer2*, and *QE1_Layer3* respectively, to eliminate multiple quantifiers.

Finally, we present our overall QE algorithm *Project* for computing $\exists X.\, A$, where A is a conjunction of LMCs over a set of variables V such that $X \subseteq V$. Initially *Project* tries to compute $\exists X.\, A$ using *Layer1*. This reduces $\exists X.\, A$ to an equivalent conjunction of A_1 and $\exists Y.\, A_2$, where A_1, A_2 are conjunctions of LMCs and $Y \subseteq X$. If all variables in X are eliminated by *Layer1*, then $\exists X.\, A \equiv A_1$. *Project* returns A_1 in this case. Otherwise, *Project* tries to compute $\exists Y.\, A_2$ using *Layer2*. *Layer2* reduces $\exists Y.\, A_2$ to an equivalent conjunction of A_3 and $\exists Z.\, A_4$, where A_3, A_4 are conjunctions of LMCs and $Z \subseteq Y$. If all variables in Y are eliminated by *Layer2*, then $\exists X.\, A \equiv A_1 \wedge A_3$. *Project* returns $A_1 \wedge A_3$ in this case. Otherwise, *Project* calls *Layer3* to compute $\exists Z.\, A_4$, and returns $A_1 \wedge A_3 \wedge \exists Z.\, A_4$. *Layer3*

3 QE from Boolean Combinations of LMCs

In [1], we explored a *Decision Diagram (DD)*-based approach and an *SMT solving (SMT)*-based approach for extending a QE algorithm for conjunctions of LMEs and LMDs to Boolean combinations of LMEs and LMDs. In this section,

we extend these approaches to Boolean combinations of LMEs, LMDs and LMIs. We also present a hybrid approach that tries to combine the strengths of the DD-based and SMT-based approaches.

Extending DD-based and SMT-based approaches: Linear Modular Decision Diagrams (LMDDs) [1] are BDD-like data structures used to represent Boolean combinations of LMEs and LMDs. By allowing nodes in LMDDs to be labeled with LMEs or LMIs, we can use LMDDs to represent Boolean combinations of LMEs, LMIs and LMDs. In the subsequent discussion, we represent a non-terminal LMDD node f as ($pred(f)$, $high(f)$, $low(f)$), where $pred(f)$ is the LME/LMI labeling the node, $high(f)$ is the high child, and $low(f)$ is the low child, as defined in [13]. For simplicity of notation, we will use f to denote both an LMDD and the Boolean combination of LMCs represented by it, when the context precludes any disambiguity in interpretation.

Given an LMDD f and a variable x, the DD-based approach for computing $\exists x.f$ is similar to that described in [1]. Specifically, we perform a recursive traversal of the LMDD f, collecting the set of LMCs containing x (henceforth called *context*) encountered along the path from the root node of LMDD f. We call the corresponding recursive procedure *QE1_LMDD*. In each recursive call, *QE1_LMDD* computes an 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 in the context. If g is a 1-terminal, then $\exists x.\,(g \wedge C_x)$ is computed by calling *Project* on $\exists x.\,C_x$. If the root node of g is a non-terminal, then *QE1_LMDD* simplifies g using the LMEs in C_x before traversing g, as described in[1]. Multiple variables can be eliminated by invoking *QE1_LMDD* repeatedly; this is implemented in the procedure *QE_LMDD*. The reader is referred to [14]for additional details of *QE_LMDD*.

In [1], we also proposed a procedure called *Monniaux* (originally introduced in [9]) that uses SMT solving to eliminate quantifiers from Boolean combinations of LMEs and LMDs. We extend *Monniaux* to handle Boolean combinations of LMCs involving LMIs. 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. *Monniaux* improves upon this by generalizing a satisfying assignment to obtain a cube of satisfying assignments, by projecting the cube on the remaining variables (not in X), and then conjoining its complement with f before additional satisfying assignments are found.

Combining DD-based and SMT-based approaches: The factors that contribute to the success of the DD-based approach are the presence of large shared sub-LMDDs and the strategy of eliminating one variable at a time. Both factors contribute to significant opportunities for reuse of results through dynamic programming. The success of the SMT-based approach is attributable primarily to pruning of the solution space achieved by interleaving of projection and model enumeration. In the following discussion, we present a hybrid approach that tries to combine the strengths of these two approaches.

The hybrid procedure, called *QE_Combined*, is shown in Fig. 2. The procedure uses the following helper functions: a) *qeddContext*: variant of *QE_LMDD* to compute $\exists X.\,(f \wedge C)$, where f is an LMDD and C is a conjunction of LMCs over a set of variables V, and $X \subseteq V$, b) *getConjunct*: computes the conjunction of LMCs in a given set, c) *Sat*: checks if a given Boolean combination of LMCs is satisfiable.

```
QE_Combined(f, X)
 π ← selectPath(f);
S ← ∅; /* set of sub-problems */
simplify(f, π, S, ∅);
g ← false;
for each (⟨f_i, C_i⟩ ∈ S)
  if (Sat(f_i ∧ C_i ∧ ¬g))
    h ← qeddContext(f_i, C_i, X);
    g ← g ∨ h;
return g;
```

```
simplify(f, π, S, C)
 /* C : set of LMCs encountered along π */
if (node f is a terminal)
  S ← S ∪ {⟨f, getConjunct(C)⟩};
else if (node high(f) is in π)
  S ← S ∪ {⟨low(f), getConjunct(C)∧¬pred(f)⟩};
  simplify(high(f), π, S, C ∪ {pred(f)});
else /* node low(f) is in π */
  S ← S ∪ {⟨high(f), getConjunct(C)∧pred(f)⟩};
  simplify(low(f), π, S, C ∪ {¬pred(f)});
```

Fig. 2. Algorithms *QE_Combined* and *simplify*

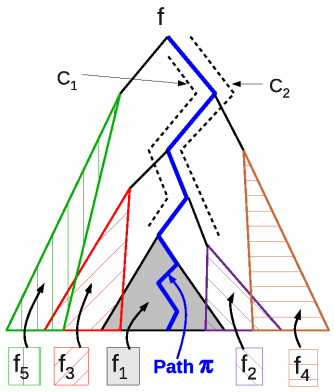

Fig. 3. Deriving $f_i \wedge C_i$ from path π

Procedure *QE_Combined* first selects a satisfiable path π in the LMDD f using a function *selectPath*. Subsequently, the procedure *simplify* is invoked, which traverses the path π, in order to convert (split) f into an equivalent disjunction $\bigvee_{i=1}^{n}(f_i \wedge C_i)$, where f_i denotes an LMDD and C_i denotes a conjunction of LMCs (represented in Fig. 2 as a set S of pairs, where each pair is of the form $\langle f_i, C_i \rangle$). Fig. 3(b) illustrates the splitting scheme followed by *simplify*. *QE_Combined* now computes $g \equiv \exists X.\,f$ as $\bigvee_{i=1}^{n}(\exists X.\,(f_i \wedge C_i))$ 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 *qeddContext*, and then h is disjoined with g. Note that unlike *Monniaux*, *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 *Monniaux*.

4 Experimental Results

We performed experiments to (i) evaluate the performance of *Monniaux*, *QE_LMDD*, and *QE_Combined*, (ii) evaluate the effectiveness of the layers in *Project*, and (iii) compare the performance of *Project* with alternative QE techniques. The experiments were performed on a 1.83 GHz Intel(R) Core 2 Duo machine with 2GB memory running Linux, with a timeout of 1800 seconds. We implemented our own LMDD package for carrying out QE experiments involving LMDDs.

Benchmarks: We used a benchmark suite consisting of 198 *lindd* benchmarks [4] and 23 *vhdl* benchmarks. Each of these benchmarks is a Boolean combination of LMCs with a subset of the variables in their support existentially quantified.

The *lindd* benchmarks reported in [4] are Boolean combinations of octagonal constraints over integers, i.e., constraints of the form $a \cdot x + b \cdot y \leq k$ where x, y are integer variables, k is an integer constant, and $a, b \in \{-1, 1\}$. We converted these benchmarks to Boolean combinations of LMCs by assuming the size of integer as 16 bits. Although these benchmarks had no LMEs explicitly, they contained LMEs encoded as conjunctions of the form $(x - y \leq k) \wedge \neg(x - y \leq k - 1)$. We converted each such conjunction to an LME $x - y = k$ as a pre-processing step. The total number of variables, the number of variables to be eliminated, and the number of bits to be eliminated in the *lindd* benchmarks ranged from 30 to 259, 23 to 207, and 368 to 3312 respectively.

The *vhdl* benchmarks were obtained in the following manner. We took a set of word-level VHDL designs. Some of these are publicly available designs obtained from [5], and the remaining are proprietary. We derived the symbolic transition relations of these VHDL designs. The *vhdl* benchmarks were obtained by quantifying out all the internal variables (i.e. neither input nor output of the top-level module) from these symbolic transition relations. Effectively this gives abstract transition relations of the designs. The coefficients of the variables in these benchmarks were largely odd. These benchmarks contained a significant number of LMEs (arising from assignment statements in the VHDL programs). The total number of variables, the number of variables to be eliminated, and the number of bits to be eliminated in the *vhdl* benchmarks ranged from 10 to 50, 2 to 21, and 10 to 672 respectively.

Evaluation of *Monniaux*, *QE_LMDD*, and *QE_Combined*: We measured the time taken by *Monniaux*, *QE_LMDD*, and *QE_Combined* for QE from each benchmark. For *QE_LMDD* and *QE_Combined*, this included the time to build the initial LMDD. We observed that each approach performed better than the others for some benchmarks (see Fig. 4). Note that the points in Fig. 4(a) are scattered, while the points in Fig. 4(b) and 4(c) are more clustered near the 45° line. This shows that *DD* and *SMT* based approaches are incomparable, whereas the hybrid approach inherits the strengths of both *DD* and *SMT* based approaches. Hence, given a problem instance, we recommend the hybrid approach, unless the approach which will perform better is known a-priori.

Evaluation of *Project*: Recall that *Layer3* converts a conjunction of LMCs to a Boolean combination of LMCs and calls *Monniaux*/ *QE_LMDD*/ *QE_Combined*

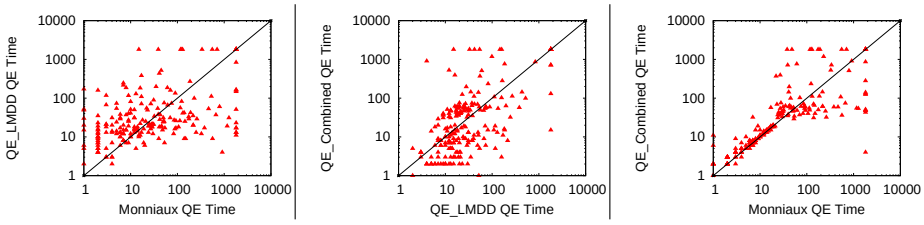

Fig. 4. Plots comparing (a) *Monniaux* and *QE_LMDD*, (b) *QE_LMDD* and *QE_Combined*, and (c) *Monniaux* and *QE_Combined* (All times are in seconds)

for QE from this Boolean combination, which results in new (recursive) *Project* calls. Hence two kinds of *Project* calls were generated while performing QE from the benchmarks: (i) the initial/original *Project* calls, and the (ii) aforementioned recursive *Project* calls. In the subsequent discussion, whenever we mention "*Project* calls", it refers to the initial/original *Project* calls, unless stated otherwise.

Table 1. Details of *Project* calls (figures are per *Project* call)

Type	**Vars**	**Quant**	**LMIs**	**LMEs**	**LMDs**	**Contr.**			**Time (milli seconds)**			
						L1	**L2**	**L3**	**L1**	**L2**	**L3**	***Project***
lindd	39.9	38.1	(88, 0, 18.9)	(60, 0, 10.1)	(35, 0, 8.1)	51	44	5	3	5	13149	674
vhdl	9.3	7.8	(4, 0, 0.4)	(16, 0, 6.3)	(31, 0, 1.8)	95	4.5	0.5	1	6	161	3

Vars : Average number of variables, **Quant** : Average number of quantifiers, **LMIs** : (Maximum, minimum, average) number of LMIs, **LMEs** : (Maximum, minimum, average) number of LMEs, **LMDs** : (Maximum, minimum, average) number of LMDs, **Contr.** : Average contribution of a layer, **L1** : *Layer1*, **L2** : *Layer2*, **L3** : *Layer3*, **Time** : Average time spent per quantifier eliminated

The total number of *Project* calls generated from the *lindd* and *vhdl* benchmarks were $52,836$ and $7,335$ respectively. Statistics of these *Project* calls are shown in Table 1. The contribution of a layer is measured as the ratio of the number of quantifiers eliminated by the layer to the number of quantifiers to be eliminated in the *Project* call, multiplied by 100. The contributions of the layers and the times taken by the layers per quantifier eliminated for individual *Project* calls from *lindd* benchmarks are shown in Fig. 5 and Fig. 6. The *Project* calls here are sorted in increasing order of contribution from *Layer1*.

Layer1 and *Layer2* were cheap and eliminated a large fraction of quantifiers in both *lindd* and *vhdl* benchmarks. This underlines the importance of our layered framework. The relatively large contribution of *Layer1* in the *Project* calls from *vhdl* benchmarks was due to significant number of LMEs in these problem instances. *Layer3* was found to be the most expensive layer. Most of the time spent in *Layer3* was consumed in the recursive *Project* calls. No *Layer3* call in our experiments required model enumeration. The large gap in the time per quantifier in *Layer2* and that in *Layer3* for both sets of benchmarks points to

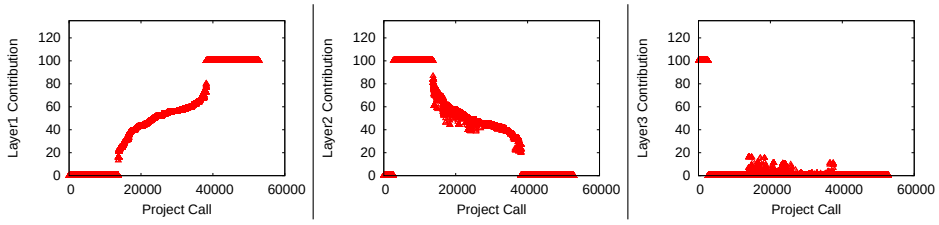

Fig. 5. Contribution of (a) *Layer1*, (b) *Layer2*, and (c) *Layer3* for *lindd* benchmarks

the need for developing additional cheap layers between *Layer2* and *Layer3* as part of future work.

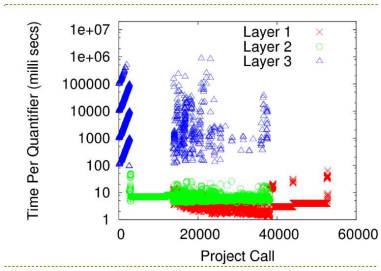

Fig. 6. Cost of layers for *lindd* benchmarks

Comparison of *Project* with alternative QE techniques: We compared the performance of *Project* with QE based on ILA using Omega Test, and also with QE based on bit-blasting. We implemented the following algorithms for this purpose: (i) *Layer1_Blast*: this procedure first quantifies out the variables using *Layer1* (recall that *Layer1* is a simple extension of [1]), and then uses bit-blasting and BDD based bit-level QE [6] for the remaining variables. (ii) *Layer1_OT*, *Layer2_OT*: *Layer1_OT* first quantifies out the variables using *Layer1*, and then uses conversion to ILA and Omega Test [8] for the remaining variables. *Layer2_OT* first quantifies out the variables using *Layer1* followed by *Layer2*, and then uses conversion to ILA and Omega Test for the remaining variables. *Layer2_OT* helps us to compare the performance of *Layer3* with that of Omega Test.

We collected the instances of QE problem for conjunctions of LMCs arising from *Monniaux* when QE is performed on each benchmark. We performed QE from such conjunction-level problem instances using *Project*, *Layer1_Blast*, *Layer1_OT*, and *Layer2_OT*. Fig. 7(a) and 7(b) compare the average QE times taken by *Project* against those taken by *Layer1_Blast* and *Layer1_OT* for QE from the conjunction-level problem instances for each benchmark. Subsequently, for each benchmark, we compared the average time consumed by *Layer3* in the *Project* calls with that consumed by Omega Test in the *Layer2_OT* calls (see Fig. 7(c)). The results clearly demonstrated that (i) *Project* outperforms both the alternative QE techniques and (ii) *Layer3* outperforms Omega Test. There were a few cases where Omega Test performed better than *Layer3*. This was due to the relatively larger number of recursive *Project* calls in these cases.

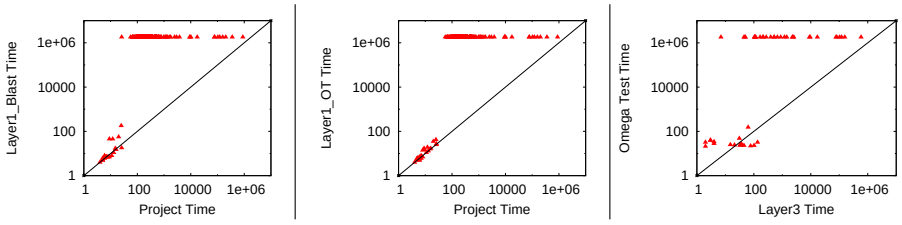

Fig. 7. Plots comparing (a) *Project* and *Layer1_Blast*, (b) *Project* and *Layer1_OT*, and (c) *Layer3* and Omega Test (All times are in milli seconds)

5 Conclusion

The need for efficient techniques for bit-precise quantifier elimination cannot be overemphasized. In this paper, we presented practically efficient techniques for eliminating quantifiers from Boolean combinations of LMCs. We propose to study quantifier elimination techniques for non-linear modular constraints as part of future work.

References

1. A. John, S. Chakraborty. *A quantifier elimination algorithm for linear modular equations and disequations*, In CAV 2011
2. N. Bjørner, A. Blass, Y. Gurevich, M. Musuvathi. *Modular difference logic is hard*, In CoRR abs/0811.0987:(2008)
3. D. Kroening, O. Strichman. *Decision procedures : an algorithmic point of view*, Texts In Theoretical Computer Science, Springer 2008
4. S. Chaki, A. Gurfinkel, O. Strichman. *Decision diagrams for linear arithmetic*, In FMCAD 2009
5. ITC'99 benchmarks, http://www.cad.polito.it/downloads/tools/itc99.html
6. CUDD release 2.4.2 website, vlsi.colorado.edu/∼fabio/CUDD
7. H. Enderton. *A mathematical introduction to logic*. Academic Press, 2001
8. W. Pugh. *The Omega Test: A fast and practical integer programming algorithm for dependence analysis*. Communications of the ACM, Pages 102-114, 1992
9. D. Monniaux. *A quantifier elimination algorithm for linear real arithmetic*, In LPAR 2008
10. V. Ganesh, D. Dill. *A decision procedure for bit-vectors and arrays*, In CAV 2007
11. H. Jain, E. M. Clarke, O. Grumberg. *Efficient Craig interpolation for linear diophantine (dis)equations and linear modular equations*, In CAV 2008
12. M. Muller-Olm, H. Seidl. *Analysis of modular arithmetic*, ACM Transactions on Programming Languages and Systems, 29(5):29, 2007
13. R.E. Bryant. *Graph-based algorithms for boolean function manipulation*. IEEE Transactions on Computers, C-35(8):677-691, 1986
14. A. John, S. Chakraborty. *Extending quantifier elimination to linear inequalities on bit-vectors*, Technical Report TR-12-35, CFDVS, IIT Bombay, http://www.cfdvs.iitb.ac.in/reports/reports/tacas2013_report.pdf