Form Methods Syst Des (2016) 49:272–323
DOI 10.1007/s10703-016-0260-9

A layered algorithm for quantifier elimination from linear modular constraints

Ajith K. John[1] · Supratik Chakraborty[2]

Published online: 7 December 2016

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 a practically 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 then extend this algorithm to work with arbitrary 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 linear integer arithmetic.

Keywords Quantifier elimination · Linear modular arithmetic · Bit-precise verification · Decision diagrams · Layered algorithm

1 Introduction

Quantifier elimination (QE) is the process of converting a logic formula containing quantifiers into a semantically equivalent quantifier-free formula. Formally, let F be a quantifier-free formula over a set V of free variables in a first-order theory T. Consider the quantified formula $Q_1x_1\ Q_2x_2\ \cdots\ Q_nx_n.F$, where $X = \{x_1, \ldots x_n\}$ is a subset of V, and $Q_i \in \{\exists, \forall\}$

This is an extended version of our earlier works in CAV 2011 [36] and TACAS 2013 [37].

✉ Ajith K. John
ajithkj.barc@gmail.com

Supratik Chakraborty
supratik@cse.iitb.ac.in

[1] Homi Bhabha National Institute, BARC, Mumbai, India

[2] Department of Computer Science and Engineering, IIT Bombay, Bombay, India

for $i \in \{1, \ldots n\}$. QE involves computing a quantifier-free formula F' over variables in $V \setminus X$ such that F' is semantically equivalent to $Q_1x_1\ Q_2x_2 \cdots Q_nx_n.F$ in theory T. QE has a number of important applications in formal verification and analysis of hardware and software systems. Example applications include image computation [13], computation of strongest post-conditions [38] and computation of predicate abstractions [21].

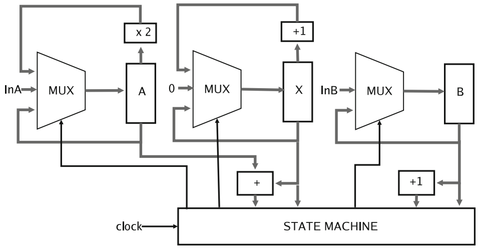
This paper focuses on existential QE from formulas in an important fragment of theory of bit-vectors [40] called linear modular arithmetic. Formulas in linear modular arithmetic are Boolean combinations of linear equalities, disequalities and inequalities on fixed-width bit-vectors. Let p be a positive 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 $a_1 \cdot x_1 + \cdots a_n \cdot x_n + a_0$, where $\cdot$ denotes multiplication modulo 2^p and $+$ denotes addition modulo 2^p. A linear modular equality (LME) is a constraint 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 constraint of the form $t_1 \neq t_2 \pmod{2^p}$, and a linear modular inequality (LMI) is a constraint of the form $t_1 \bowtie t_2 \pmod{2^p}$, where $\bowtie \in \{<, \leq\}$. We will use linear modular constraint (LMC) to refer to an LME, LMD or LMI. 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, it follows that a set of LMCs sharing a variable must have the same modulus. Hence 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 semantics of LMCs differs from that of linear constraints over integers in two aspects:

1. *Wrap-around behaviour* The successor of $2^p - 1$ in modular arithmetic is 0. Hence, if $x = 2^p - 1$, then $x + 1$ modulo 2^p overflows and wraps to 0. Due to this wrap-around behaviour, the formula $(x = 3) \wedge (x + 1 \leq 2)$ is satisfiable in linear modular arithmetic with modulus 4 whereas it is unsatisfiable over integers.
2. *Finite domain* Domain of variables in modular arithmetic has finite/bounded cardinality unlike integer arithmetic where the variables are unbounded. Hence the formula $(x = 3) \wedge (x < y)$ is unsatisfiable in linear modular arithmetic with modulus 4 whereas it is satisfiable over integers.

Efficient techniques for QE from LMCs have applications in formal verification and analysis of hardware and software systems. Formal verification and analysis tools reason about symbolic transition relations of hardware and software systems expressed as formulas in appropriate logic. Symbolic transition relations of word-level RTL designs and embedded programs involve constraints in linear modular arithmetic. LMEs arise from the assignment statements, whereas LMDs and LMIs arise primarily from branch and loop conditions that compare words/registers. Key operations such as image computation [13], computation of strongest post-conditions [38] and computation of predicate abstractions [21] performed by formal verification and analysis algorithms essentially reduce to QE from formulas involving symbolic transition relation. Symbolic transition relations of RTL designs and embedded programs in general may involve signed variables with signed operations and comparisons on them. There are standard techniques to convert constraints with signed semantics into equisatisfiable constraints with unsigned semantics (for example, see page 2 of [29]). In the remainder of this paper, we assume that all variables and all operations, comparisons are unsigned.

Our primary motivation for studying QE from LMCs arises from bounded model checking [3] of word-level RTL designs. As an example, consider the synchronous circuit shown in Fig. 1, with the relevant part of its functionality described in VHDL. The circuit comprises a controller and three 8-bit registers, A, B, and X. The controller switches between three states,



```
 ...
if (clock’event and clock = ’1’) then
 case state is
  when "00" => A <= InA;
    B <= InB; X <= x"00"; state <= "01";
  when "01" => if (X + A <= B) then
   X <= X+’1’; A <= x"02"*A;
   elsif (X = B+’1’) then state <= "10";
   else state <= "00"; end if;
  when others => state <= "00";
 end case;
end if;
   ....
```

Fig. 1 An example circuit

0, 1, and 2. In state 0, the values of A and B are read from inputs InA and InB respectively, and are stored in corresponding registers. In addition, the value of X is initialized to 0, and the control moves to state 1. State 1 implements the iterative algorithm: if $\mathsf{X} + \mathsf{A} \leq \mathsf{B}$, the value of X is incremented, that of A is doubled, and the circuit continues to iterate in state 1. If, however, $\mathsf{X} + \mathsf{A} > \mathsf{B}$, the circuit checks if the value of X equals $\mathsf{B} + 1$. If so, the control moves to state 0 via state 2. Otherwise, the control moves directly to state 0 from state 1.

The symbolic transition relation, R, for this circuit can be obtained by conjoining the following equality relations, where primed variables refer to values of the corresponding unprimed variables after the next rising edge of the clock.

$$\begin{aligned}
\mathsf{state}' &= \mathsf{ite}(\mathsf{state} = 0, 1, \mathsf{ite}(\mathsf{state} = 1, \mathsf{ite}(\mathsf{X} + \mathsf{A} \leq \mathsf{B}, 1, \mathsf{ite}(\mathsf{X} = \mathsf{B} + 1, 2, 0)), 0))\\
\mathsf{A}' &= \mathsf{ite}(\mathsf{state} = 0, \mathsf{InA}, \mathsf{ite}(\mathsf{state} = 1, \mathsf{ite}(\mathsf{X} + \mathsf{A} \leq \mathsf{B}, 2 \cdot \mathsf{A}, \mathsf{A}), \mathsf{A}))\\
\mathsf{B}' &= \mathsf{ite}(\mathsf{state} = 0, \mathsf{InB}, \mathsf{B})\\
\mathsf{X}' &= \mathsf{ite}(\mathsf{state} = 0, \mathsf{0x00}, \mathsf{ite}(\mathsf{state} = 1, \mathsf{ite}(\mathsf{X} + \mathsf{A} \leq \mathsf{B}, \mathsf{X} + 1, \mathsf{X}), \mathsf{X}))
\end{aligned}$$

In the above equalities, $\mathsf{A}, \mathsf{A}', \mathsf{B}, \mathsf{B}', \mathsf{InA}, \mathsf{InB}, \mathsf{X}$, and X' are bit-vectors of width 8, whereas state and state' are bit-vectors of width 2. Furthermore, all operations and comparisons involving $\mathsf{A}, \mathsf{A}', \mathsf{B}, \mathsf{B}', \mathsf{InA}, \mathsf{InB}, \mathsf{X}$, and X' are unsigned operations modulo 2^8, and those involving state and state' are unsigned operations modulo 2^2. Since $a = ite(b, c, d)$ represents $(b \wedge (a = c)) \vee (\neg b \wedge (a = d))$, the transition relation R above is a Boolean combination of LMCs.

The above circuit computes the smallest 8-bit non-negative integer X such that $2^{\mathsf{X}} \cdot \mathsf{InA} + \mathsf{X} > \mathsf{InB}$, where all the operations are modulo 2^8. If the smallest value of X thus computed is $\mathsf{InB} + 1$, the control enters state 2; otherwise it returns to state 0. For example, suppose $\mathsf{InA} = 1$ and $\mathsf{InB} = 150$. Inside state 1, the value of A overflows to zero after 8 iterations and remains as zero thereafter. The value of X is incremented in each iteration until it becomes 151. Now that $\mathsf{X} + \mathsf{A} \leq \mathsf{B}$ is false and $\mathsf{X} = \mathsf{B} + 1$ is true, the control moves to state 2. Observe that 151 is the smallest 8-bit non-negative integer X such that $2^{\mathsf{X}} \cdot 1 + \mathsf{X} > 150$ modulo 2^8.

This circuit has the property that if it starts in state 0, then the value of A is always less than $255 \cdot \mathsf{X}$ when it visits state 2. The value of A may exceed $255 \cdot \mathsf{X}$ and even overflow during the modulo 2^8 multiplications in state 1. However, when it reaches state 2, A is less than $255 \cdot \mathsf{X}$. To see why this is true, observe that in state 2, both $\mathsf{X} + \mathsf{A} > \mathsf{B}$ and $\mathsf{X} = \mathsf{B} + 1$ are true; hence $\mathsf{X} + \mathsf{A} > \mathsf{X} + 255$ is true, where 255 is the additive inverse of 1 in modulo 2^8. Note that since $\mathsf{A} \leq 255$, $\mathsf{X} + \mathsf{A} > \mathsf{X} + 255$ implies $\mathsf{X} \neq 0$. Moreover, since $\mathsf{A} \leq 255$, if the operation $\mathsf{X} + \mathsf{A}$ overflows, then $\mathsf{X} + \mathsf{A} \leq \mathsf{X} + 255$ holds for $\mathsf{X} \neq 0$. But we have $\mathsf{X} + \mathsf{A} > \mathsf{X} + 255$. Hence the operation $\mathsf{X} + \mathsf{A}$ should not overflow. This implies that A is less than the additive inverse of X modulo 2^8. Since $255 \cdot \mathsf{X}$ is the additive inverse of X modulo 2^8, we have $\mathsf{A} < 255 \cdot \mathsf{X}$.

Suppose we wish to verify this property for the first N time steps of operation of the circuit using bounded model checking. This involves unrolling the transition relation N times, conjoining the unrolled relation with the negation of the property, and feeding the resulting formula to an SMT solver. Observe that R contains primed and unprimed versions of all variables in the circuit. Hence, unrolling R a large number of times can give a formula with a very large number of variables. While the number of variables in an SMT formula is not the sole determinant of performance of SMT solving, formulas with large numbers of variables typically lead to performance bottlenecks in SMT solving.

A common approach to circumventing this problem is to use an abstract transition relation R' that relates values of only a chosen subset of variables relevant to the property being checked, while abstracting the relation between the other variables. In general, the set of states reached using R' overapproximates the exact set of reachable states. Therefore, if N-step bounded model checking using R' fails to give a counterexample, then the property holds in N steps of operation of the circuit.

In our example, an abstract transition relation R' can be obtained by computing $\exists \mathsf{B}.\exists \mathsf{B}'.\exists \mathsf{InB}.\ R$. An equivalent quantifier-free version of R' is given below.

$$\begin{array}{l}
((\mathsf{state} = 0) \wedge (\mathsf{state}' = 1) \wedge (\mathsf{A}' = \mathsf{InA}) \wedge (\mathsf{X}' = \mathsf{0x00})) \vee \\
((\mathsf{state} = 1) \wedge (\mathsf{state}' = 1) \wedge (\mathsf{A}' = 2 \cdot \mathsf{A}) \wedge (\mathsf{X}' = \mathsf{X} + 1)) \vee \\
((\mathsf{state} = 1) \wedge (\mathsf{state}' = 2) \wedge (\mathsf{A}' = \mathsf{A}) \wedge (\mathsf{X}' = \mathsf{X}) \wedge (\mathsf{X} + \mathsf{A} > \mathsf{X} + 255)) \vee \\
((\mathsf{state} = 1) \wedge (\mathsf{state}' = 0) \wedge (\mathsf{A}' = \mathsf{A}) \wedge (\mathsf{X}' = \mathsf{X}) \wedge \varphi) \vee \\
((\mathsf{state} \neq 0) \wedge (\mathsf{state} \neq 1) \wedge (\mathsf{state}' = 0) \wedge (\mathsf{A}' = \mathsf{A}) \wedge (\mathsf{X}' = \mathsf{X}))
\end{array}$$

where φ is the disjunction of the formulas $(\mathsf{X} + \mathsf{A} \neq 0) \wedge (\mathsf{X} \neq 1)$ and $(\mathsf{X} + \mathsf{A} \neq 0) \wedge (\mathsf{X} \neq 0) \wedge (\mathsf{X} \leq \mathsf{X} + \mathsf{A} + 255)$.

It can indeed be verified that bounded model checking using R' (instead of R) suffices to show that if the circuit starts in state 0, then the value of A is always less than $255 \cdot \mathsf{X}$ when it visits state 2. Since R' does not contain B, B' or InB, the number of variables in N unrollings of R' is less than that in N unrollings of R. This is likely to lead to better performance of SMT solving during bounded model checking using R' than during bounded model checking using R. In practice, this often translates to a problem being solved within given time constraints, as opposed to timing out. Since transition relations of word-level RTL designs involve Boolean combinations of LMCs, building an abstract transition relation requires existentially quantifying variables from Boolean combinations of LMCs.

The above example illustrates the potential advantages of using an abstract transition relation obtained by existentially quantifying a subset of variables from the original transition relation. However, the effectiveness of this approach depends crucially on the choice of variables to quantify, on the availability of efficient techniques to obtain a quantifier-free version of the abstract transition relation, and on the quality of the abstract transition relation obtained.

For ease of computation, formal verification and analysis algorithms abstract variables in the system to be verified as integers, and use QE techniques for integers [51]. However the underlying system implementation often uses modular arithmetic, and as mentioned earlier, the semantics of integer arithmetic differs from that of modular arithmetic. Hence, as observed in [7], the results of verification and analysis by abstracting variables as integers and using QE for integers may not be sound or complete if the underlying implementation uses modular arithmetic. Therefore, developing *bit-precise* and *practically efficient* QE techniques for LMCs is an important problem.

1.1 Contributions

There are two key technical contributions of this work.

1. We present a bit-precise and practically efficient algorithm for eliminating quantifiers from conjunctions of LMCs. Our algorithm is based on a layered approach, whereby sound but incomplete and cheaper layers are invoked first, and expensive but complete layers are called only when required. While our algorithm uses a final layer of model enumeration for the sake of theoretical completeness, extensive experiments indicate that we do not need to invoke this layer on a wide range of benchmarks arising in practice. Experiments also demonstrate the effectiveness of our algorithm over alternative QE techniques based on bit-blasting and conversion to linear integer arithmetic.
2. We present approaches to extend this algorithm to eliminate quantifiers from Boolean combinations of LMCs. We introduce a new decision diagram called Linear Modular Decision Diagram (LMDD) that represents Boolean combinations of LMCs, and present algorithms for QE from LMDDs. We then present an SMT solving based approach for QE from Boolean combinations of LMCs, and a hybrid approach that tries to combine the strengths of the LMDD and SMT solving based approaches. Experiments demonstrate the effectiveness of these approaches and utility of these approaches in bounded model checking of word-level RTL designs.

2 Related work

Currently, the dominant technique for eliminating quantifiers from LMCs involves *blasting* bit-vector variables into individual bits (also called bit-blasting [40]), followed by elimination of the blasted bit-level variables using bit-level QE tools [53]. However, blasting involves a bitwidth-dependent blow-up in the size of the problem. This can present scaling problems in the usage of bit-level QE tools, especially when reasoning about wide words. Similarly, if quantified variables and non-quantified variables appear as arguments of the same function or predicate, then blasting quantified variables may transitively require blasting non-quantified variables as well. This can cause the quantifier-eliminated formula to appear like a propositional formula on blasted bits, instead of being a modular arithmetic formula. Since reasoning at the level of modular arithmetic is often more efficient in practice than reasoning at the level of bits, QE using bit-blasting might not be the best option if the quantifier-eliminated formula is intended to be used in further modular arithmetic level reasoning.

Another technique for eliminating quantifiers from LMCs is converting the LMCs to equivalent constraints in linear integer arithmetic [8], and then using QE techniques for linear integer arithmetic such as Omega Test [51]. Similarly, automata-theoretic approaches for eliminating quantifiers from linear integer arithmetic constraints [28] can also be used. However, this approach scales poorly in practice and destroys the modular arithmetic structure of the problem. The resulting formula is a linear integer arithmetic formula and converting this formula back to modular arithmetic is often difficult.

The problem of extending a QE algorithm for conjunctions of constraints to Boolean combinations of constraints is encountered in other first order theories such as linear real arithmetic and linear integer arithmetic as well. In the following, we first focus on existing approaches to solve this problem for these theories. We then provide a brief account of the existing complexity results on QE and related problems for LMCs. Note that the related works we survey below arise from a range of applications. Some of these applications such as SMT solving, generation of Craig interpolants [17] etc., may not directly require QE. Nevertheless these works are included here for completeness, since there is overlap between the objectives of QE and what these works achieve.

2.1 Existing techniques for extending QE to Boolean combinations

Cavada et al.'s work [11] addresses the problem of existentially quantifying out all numeric variables from formulas involving linear arithmetic constraints and Boolean variables. Their work uses BDDs [10] to represent Boolean structure of the formulas. QE is done by recursively traversing the BDD, carrying along each path, the linear arithmetic constraints encountered on it so far (called the context). Paths with theory-inconsistent contexts are removed. Because of the dependence of the result of a recursive call on the context, if the same BDD node is encountered following two different paths, the results of the calls are not the same in general. Hence this procedure is not amenable to dynamic programming usually employed in the implementation of BDD operations. In particular, the number of recursive calls in the worst-case is linear in the number of paths, and not the number of nodes, of the original BDD.

Chaki et al. [12] present a practically efficient algorithm for QE from formulas in the theory of Octagons (a fragment of linear real arithmetic for which Fourier–Motzkin algorithm [20] is sufficient for conjunction-level QE). Their work introduces decision diagrams for linear arithmetic called LDDs. QE from LDDs makes use of a single variable elimination procedure that recursively applies Fourier–Motzkin style elimination on the LDD nodes. This procedure can be implemented with dynamic programming, which helps in achieving considerable performance improvement as reported in [12].

Suppose we wish to quantify a set of variables X from a formula F in linear real arithmetic. A straightforward algorithm to compute $\exists X.F$ is All-SMT algorithm (also called All-SMT loop) that works as follows (versions of this algorithm can be found in [41,44]). An SMT solver call is used to check if F is satisfiable. If F is unsatisfiable, then $\exists X.F$ is false. Otherwise, the solution of F is generalized to a conjunction C_1 of constraints such that $C_1 \Rightarrow F$. The SMT solver is now called to check if $F \wedge \neg C_1$ is satisfiable. If $F \wedge \neg C_1$ is unsatisfiable, then $\exists X.F$ is equivalent to $\exists X.C_1$. Otherwise, the solution of $F \wedge \neg C_1$ is generalized to a conjunction C_2 such that $C_2 \Rightarrow F$. This loop is repeated until the formula given to the SMT solver becomes unsatisfiable. Each iteration i of the loop generates a conjunction C_i such that $C_i \Rightarrow F$, for $1 \leq i \leq n$ (C_i is also called implicant). Finally, $\exists X.F$ is equivalent to $\exists X.C_1 \vee \cdots \vee \exists X.C_n$.

The work by Lahiri et al. [41] improves the All-SMT algorithm by considering $\neg C_i$ as a conflicting clause and then performing conflict-driven back-jumping inside the SMT solver. Monniaux [44] improves the All-SMT algorithm in the following ways. First, instead of $\neg C_i$, $\neg \exists X.C_i$ is conjoined with the formula given to the SMT solver. This is called "interleaving projection and model enumeration" in [44]. Secondly, an SMT solver based procedure is used to further generalize the implicant C_i by dropping constraints from C_i wherever possible, before $\exists X.C_i$ is computed. It is observed in [44] that these optimizations help in early termination of the algorithm, and yield significant performance improvements on a wide range of benchmarks. The later work by Monniaux [45] and the work by Phan et al. [50] improves this algorithm further in handling of quantifier alternations.

Techniques for finding generalized implicants are crucial in scalable application of the All-SMT algorithm. Many interesting approaches are proposed recently for deriving such generalized implicants from a given solution of an SMT formula. De Moura et al. [23] present a variation of Boolean constraint propagation in order to identify constraints whose truth values are not essential for determining the satisfiability of a formula. Déharbe et al. [25]

present algorithms for generating prime implicants from solutions of formulae by iterative removal of assignments that are not necessary. Niemetz et al. [47] present a dual propagation based technique to extract partial solutions from "full" solutions of SMT formulas. Given a solution m of a formula F, the assignments to variables in m are presented as assumptions to a dual solver which maintains $\neg F$. The assumptions that are inconsistent with $\neg F$ identify the assignments sufficient to satisfy F.

Test point based QE algorithms such as Ferrante and Rackoff's algorithm [26], Loos and Wiespfenning's algorithm [42] for linear real arithmetic and Cooper's Algorithm [16] for linear integer arithmetic can be directly applied on arbitrary Boolean combinations of constraints. However, scalability of these algorithms in practice often depends on underlying representation of Boolean structure of the formulas and implementation heuristics used.

LinAIG tool [19] implements Loos and Wiespfenning's algorithm using a data structure called LinAIG. Boolean structure of the formulas is represented using FRAIGs [43], and Craig interpolants are used to identify and remove redundant constraints generated during application of Loos and Wiespfenning's algorithm. Bjørner's work in [4] avoids application of substitutions in the formulation of Loos and Wiespfenning's algorithm and Cooper's algorithm. The effect of substitutions is encoded as an additional constraint called *pivot* which is conjoined with the input formula F. Satisfying assignments to $F \wedge pivot$ are generated using a DPLL(T) framework, which are then generalized to disjuncts in the formulation of Loos and Wiespfenning's algorithm or Cooper's algorithm. Nipkow's work [48] provides implementations of Ferrante and Rackoff's algorithm, Loos and Wiespfenning's algorithm, and Cooper's algorithm that are verified in the theorem prover Isabelle.

Komuravelli et al. [39] introduce model based projection that involves computing model-based under-approximations of existentially quantified formulas. Their work also gives procedures for computing such under-approximations for existentially quantified formulas in linear arithmetic as disjuncts in the formulation of Loos and Wiespfenning's algorithm or Cooper's algorithm. Bjørner et al. [6] give an algorithm that makes use of model based projections for deciding the satisfiability of quantified linear arithmetic formulas. Their algorithm conceptually works as a two-player satisfiability game and can be extended for QE from linear arithmetic formulas.

The work by Veanes et al. [56] focuses on automatically constructing monadic decompositions of formulas in quantifier free fragments of first order logic. Monadic decomposition involves transforming a given formula into an equivalent Boolean combination of unary predicates. Veanes et al. give an algorithm for constructing monadic decompositions in Disjunctive Normal Form (DNF). Once such a decomposition is constructed, QE can be achieved by distributing the existential quantifiers over disjunctions in the DNF. This effectively reduces the problem of eliminating quantifiers from a general formula to the problem of eliminating quantifiers from conjunctions involving only unary predicates.

2.2 Complexity results on LMCs

The satisfiability problem for a conjunction of LMEs is known to be polynomial-time [27]. However, the satisfiability problem for conjunctions of even very limited fragments of LMDs or LMIs are proved to be NP-hard as discussed below.

Jain et al. [35] prove that the satisfiability problem for a conjunction of LMDs is NP-hard even when the modulus is fixed to 4. Bjørner et al.'s work [7] introduces Modular Difference Logic (MDL) constraints. MDL constraints are a fragment of LMIs of the form $x_1 + k_1 \leq x_2 + k_2 \pmod{2^p}$, where x_1, x_2 are variables, and k_1, k_2 are constants. Bjørner et al. prove that the satisfiability problem for conjunctions of MDL constraints of the form

$x_1 + 1 \leq x_2 \pmod{2^p}$ or of the form $x_1 \leq x_2 + 2^p - 1 \pmod{2^p}$ with $2^p \geq 4$ is NP-hard.

Gange et al.'s work [29] proves that the satisfiability problem for conjunctions of LMIs involving LMIs of the form $x_1 - x_2 \geq 1 \pmod{2^p}$ and $x_1 - x_2 \leq 2 \pmod{2^p}$ is NP-hard, where $2^p \geq 4$ and $-x_2$ represents additive inverse of x_2 modulo 2^p. Since $x_1 - x_2 \geq 1 \pmod{2^p}$ is equivalent to $x_1 \neq x_2 \pmod{2^p}$, this result also implies that the satisfiability problem for conjunctions of LMCs involving LMDs of the form $x_1 \neq x_2 \pmod{2^p}$ and LMIs of the form $x_1 - x_2 \leq 2 \pmod{2^p}$ with $2^p \geq 4$ is NP-hard.

Since the satisfiability problem is a special case of QE problem (checking satisfiability of a formula is equivalent to existentially quantifying all free variables in the formula), the above results imply that QE problem for a conjunction of LMCs is NP-hard in general.

2.3 Decision procedures and interpolation procedures for LMCs

There are several techniques (see [33,54]) on solving conjunctions of LMEs using variants of Gaussian elimination. Müller-Olm et al. [46] and Huang et al. [34] give Gaussian elimination based algorithms for deriving "solved form" for conjunctions of LMEs. A solved form captures all possible solutions of a given conjunction of LMEs. Ganesh et al. [27] give a solve-and-substitute algorithm to derive a solved form for a conjunction of LMEs.

Most SMT solvers decide the satisfiability of conjunctions of LMDs and/or LMIs by bit-blasting followed by SAT solving. However, as mentioned earlier, because of the bitwidth-dependent blow-up during bit-blasting, this approach suffers from scaling problems for problem instances with large moduli. Hadarean et al. [32] proposes an extension of the congruence closure algorithm [40] for deciding the satisfiability of conjunctions of LMDs. Their work also proposes an algorithm to decide the satisfiability of conjunctions of a special class of MDL constraints that do not have the wrap-around behaviour, viz. constraints of the form $x_1 \lhd x_2 \pmod{2^p}$ where $\lhd \in \{<, \leq\}$. Gange et al. [29] propose a sound heuristic to check the satisfiability of MDL constraints that makes use of wrapped intervals [30] to represent over-approximations of the relations between variables.

Modern SMT solvers, such as, Z3 [24] and theorem-provers such as PVS [49] use specialized heuristics [57] to solve quantified bit-vector formulas by Skolemization followed by use of appropriate choices of Skolem functions. The use of p-adic expansions [1,15] is explored in [2,55] to solve *non-linear* modular equations. Bruttomesso et al. [9] present a polynomial time algorithm for solving conjunctions of constraints in the core bit-vector theory consisting of only equalities, extractions and concatenations. Their algorithm first generates an equi-satisfiable conjunction of equalities on non-overlapping slices of variables involved in the constraints. Congruence closure algorithm is then used for checking the satisfiability of this conjunction of equalities on non-overlapping slices. Similar slicing based ideas for solving conjunctions of bit-vector constraints can be found in [5,18].

Jain et al. [35] give a polynomial-time algorithm for computing Craig interpolants for conjunctions of LMEs. Griggio [31] presents a layered framework for computing interpolants for bit-vector formulas that tries to keep the word-level structure of the problem as much as possible. The cheaper layers use interpolation in EUF (equality + uninterpreted functions) and interpolation by equality substitution. The more expensive layers use conversion to linear integer arithmetic and bit-blasting. Their layered framework has similarity to our layered approach. However the individual layers used are different.

3 QE for conjunctions of LMCs

The problem we wish to solve in this section can be formally stated as follows. Let A denote a conjunction of LMCs over a set of variables V. We wish to compute a Boolean combination of LMCs φ, such that $\varphi \equiv \exists X.A$, where $X \subseteq V$. We present a layered algorithm called *Project* to solve this problem. In the following, after the notation and preliminaries, we give an overview of the techniques used in each layer; details of these techniques are presented in the following subsections.

We will initially focus on the simpler problem of existentially quantifying a single variable from a conjunction of LMCs. We use x to denote the variable to be quantified. For clarity of exposition, in most of the lemmas and propositions presented in this section, we give illustrative examples before presenting the detailed proofs.

3.1 Notation and preliminaries

We assume that all LMCs have modulus 2^p for some positive integer p, unless stated otherwise. For notational clarity, we will henceforth omit mentioning " $\pmod{2^p}$" with LMCs. 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.

Proposition 1 *$(t_1 < t_2)$ is equivalent to both $(t_1 \leq 2^p - 2) \wedge (t_1 + 1 \leq t_2)$ and $(t_2 \geq 1) \wedge (t_1 \leq t_2 - 1)$.*

Proof of Proposition 1 is obvious from the definition of $t_1 < t_2$ and the fact that the operations are modulo 2^p. Proposition 1 implies that 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.

Proposition 2 *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 $0 \leq \mu \leq p$.*

Example All LMCs in this example have modulus 8. Consider the LME $7x + 4y = x + z$. Rearranging the terms modulo 8, we get $7x - x = z - 4y$. Simplifying modulo 8, we get $6x = 4y + z$, which can be written as $2^1 \cdot 3x = 4y + z$. Multiplying by 3 (multiplicative inverse of 3 modulo 8) and simplifying gives, $2^1 x = 4y + 3z$. Similarly, the LMD $7x + 4y \neq x + z$ with modulus 8 can be equivalently expressed as $2^1 x \neq 4y + 3z$.

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 $2^{\kappa(x,t_1)} \cdot b \cdot x + t$, where t is a linear term free of x, and b is an odd number. 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 $\kappa(x, C) < p$.

3.2 Overview of layers in *Project*

The first layer of *Project* (Layer1) involves simplification of the given conjunction of LMCs using the LMEs present in the conjunction. For example, consider the problem of computing $\exists x.((6x+y=4)\wedge(2x+z\neq 0))$ with modulus 8. Note that $(6x+y=4)$ can be equivalently expressed as $(2x=5y+4)$ in modulo 8 using modular arithmetic operations. The variable x can be eliminated from the conjunction by replacing the occurrences of $2x$ in the conjunction by $5y+4$. Layer1 performs elimination of quantifiers by simplifications as above using LMEs present in the conjunction.

The second layer (Layer2) makes use of an efficient combinatorial heuristic to identify unconstraining LMIs and LMDs that can be dropped from the problem instance. For example, consider the problem of computing $\exists x.((2x=5y+4)\wedge(x+y\leq 3))$ with modulus 8. Note that x, y are 3-bit variables here. The LME $(2x=5y+4)$ is independent of the most significant bit of x, denoted as $x[2]$. It can be observed that every solution of $(2x=5y+4)$ can be "adapted" by possibly modifying the value of $x[2]$ to become a solution of $(2x=5y+4)\wedge(x+y\leq 3)$. This means that $\exists x.((2x=5y+4))\Rightarrow\exists x.((2x=5y+4)\wedge(x+y\leq 3))$. The converse, i.e., $\exists x.((2x=5y+4)\wedge(x+y\leq 3))\Rightarrow\exists x.((2x=5y+4))$ obviously holds. Hence $(x+y\leq 3)$ is unconstraining in $\exists x.((2x=5y+4)\wedge(x+y\leq 3))$ and it can be dropped. Layer2 computes sufficient and polynomial time computable conditions that identify such unconstraining LMDs and LMIs and drops them.

The cases that are not computed by the application of the above computationally cheap layers are handled by expensive but more complete techniques in the third layer (Layer3). Layer3 primarily involves a variant of Fourier–Motzkin algorithm adapted to work for LMIs. First the LMIs in the problem instance are converted to a "coefficient-matched" form $a\cdot x\bowtie t$, where $\bowtie\in\{\leq,\geq\}$, and t is a linear term free of x. Then a Fourier–Motzkin style variable elimination algorithm is applied on the coefficient-matched LMIs to eliminate the quantified variable. For example, consider the problem of computing $\exists x.((y\leq 4x)\wedge(4x\leq z))$ with modulus 16. Note that $\exists x.((y\leq 4x)\wedge(4x\leq z))$ expresses the condition under which there exists a multiple of 4 between y and z, where $y\leq z$. Our algorithm computes $\exists x.((y\leq 4x)\wedge(4x\leq z))$ as $(y\leq z)\wedge\varphi$, where φ is the disjunction of $(z\geq y+3)\wedge(y\leq 12)$, $(z<y+3)\wedge(4y=0)$, and $(z<y+3)\wedge(4y>4z)$.

Finally Layer3 uses model enumeration as the last resort. Model enumeration involves elimination of the quantified variable by enumerating of all possible values of the variable. Our experiments however indicate that we do not need to invoke model enumeration on a wide range of benchmarks arising in practice.

Techniques in Layer1 and Layer2 can be considered as preprocessing or simplification steps that preprocess or simplify the given conjunction of LMCs and eliminate quantifiers if possible. However inside Layer3, converting LMIs to coefficient-matched form, in general generates a Boolean combination of LMCs. Elimination of quantifiers from this Boolean combination of LMCs results in new recursive *Project* calls. Because of this feedback, the control flow inside *Project* is not linear. Hence we choose to call Layer1 and Layer2 as layers, not as preprocessing or simplification steps.

It is well known that order of elimination of variables crucially affects the running time of QE algorithms in general. Inside the layers, when there are multiple variables to eliminate, any ordering heuristic can be used. However the focus of this work does not include finding the best possible order of elimination. The specific order of elimination of variables we have used inside the layers is elaborated in Sect. 5.1.

Time complexities of layers Layer1 and Layer2 have polynomial worst-case time complexities. Let n be the number of constraints in the conjunction given as input, v be the number of variables in the conjunction, and let e be the number of variables to be eliminated. Assuming that additions, multiplications, and finding multiplicative inverses on p-bit numbers take time $O(Q(p))$ in the worst-case, where $Q(p)$ is a polynomial on p such that $p \leq Q(p) \leq p^3$, Layer1 has a worst-case time complexity of $O(e \cdot n \cdot v \cdot Q(p))$, and Layer2 has a worst-case time complexity of $O(e \cdot n^2 \cdot Q(p) + n \cdot p \cdot v)$. Layer3 resorts to model enumeration as the last resort, and has a worst-case time complexity of $O(n \cdot Q(p) \cdot 2^{(e+1) \cdot p} + n \cdot p \cdot v \cdot 2^{e \cdot p})$. Recall that algorithms for QE from linear arithmetic have exponential complexities [16,26,42].

3.3 Layer1: simplification using LMEs

Layer1 involves simplification of the given conjunction of LMCs using the LMEs present in the conjunction. It is an extension of the work by Ganesh et. al. [27]. The following Proposition and Lemmas form the crux of Layer1.

Proposition 3 *Let c be an LME $2^k \cdot x = t$, where k denotes $\kappa(x, c)$. Then $\exists x.c \equiv (2^{p-k} \cdot t =$* 0).

Example All LMCs in this example have modulus 8. We have $\exists x.(2^1 x = 5y + 2) \equiv (2^{3-1}(5y + 2) = 0) \equiv (4y = 0)$.

Proof Let φ_1 and φ_2 denote the formulas $\exists x.(2^k \cdot x = t)$ and $2^{p-k} \cdot t = 0$ respectively. To see that $\varphi_1 \Rightarrow \varphi_2$, we simply multiply both sides of $2^k \cdot x = t$ by 2^{p-k}, and simplify modulo 2^p. To see why $\varphi_2 \Rightarrow \varphi_1$, note that φ_2 implies that the least significant k bits of t evaluate to zero. Also recall that t is free of x. Given any value of variables in t such that the least significant k bits of t evaluate to zero, we can always find a value of x such that $2^k \cdot x = t$. This can be done by choosing the least significant $p - k$ bits of x to be the same as the most significant $p - k$ bits of t. Hence, $\varphi_2 \Rightarrow \varphi_1$, and therefore $\varphi_1 \equiv \varphi_2$. □

Lemma 1 *Let A be a conjunction of LMEs. Then $\exists x.A$ can be equivalently expressed as a conjunction of LMEs each of which is free of x.*

Example All LMCs in this example have modulus 8. Consider the problem of computing $\exists x.((2^1 x = 5y+2) \wedge (2^2 x = 5y+6z) \wedge (2^1 x = 2y+4))$. This can be equivalently expressed as $\exists x.((2x = 5y + 2) \wedge (2 \cdot (5y + 2) = 5y + 6z) \wedge (5y + 2 = 2y + 4))$. Simplifying modulo 8, we get $\exists x.((2x = 5y + 2)) \wedge (5y + 2z = 4) \wedge (3y = 2)$. Using Proposition 3, we obtain the final result as $(4y = 0) \wedge (5y + 2z = 4) \wedge (3y = 2)$.

Proof Let A be $\bigwedge_{i=1}^{m}(q_i)$, where each q_i is an LME. Let each LME q_i be of the form $2^{k_i} \cdot x = t_i$, where $k_i = \kappa(x, q_i)$ and $1 \leq i \leq m$. Without loss of generality, let k_1 be the minimum of $k_1, \ldots, k_m$. It can be observed that the LME $2^{k_1} \cdot x = t_1$ can be used to eliminate the occurrences of x in other LMEs by expressing each LME $2^{k_i} \cdot x = t_i$ for $2 \leq i \leq m$ as $2^{\mu_i} \cdot t_1 = t_i$, where each $\mu_i = k_i - k_1$. Hence, $\exists x.A$ can be equivalently expressed as $C_1 \wedge \exists x.(2^{k_1} \cdot x = t_1)$, where C_1 is the conjunction of the LMEs $2^{\mu_i} \cdot t_1 = t_i$. Using Proposition 3, it follows that $C_1 \wedge \exists x.(2^{k_1} \cdot x = t_1)$ is equivalent to $C_1 \wedge (2^{p-k_1} \cdot t_1 = 0)$. □

Lemma 2 *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.*

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

Proof Let A be equivalent to $E \wedge D \wedge I$, where E is a conjunction of LMEs, D is a conjunction of LMDs, and I is a conjunction of LMIs. Let E be $\bigwedge_{i=1}^{m}(q_i)$, where each q_i is an LME, D be $\bigwedge_{i=m+1}^{n}(d_i)$, where each d_i is an LMD, and I be $\bigwedge_{i=n+1}^{r}(l_i)$, where each l_i is an LMI.

Suppose each LME q_i is of the form $2^{k_i} \cdot x = t_i$, where $k_i = \kappa(x, q_i)$ and $1 \leq i \leq m$. Suppose each LMD d_i is of the form $2^{k_i} \cdot x \neq t_i$, where $k_i = \kappa(x, d_i)$ and $m+1 \leq i \leq n$. In addition, suppose each LMI l_i is of the form $(a_i \cdot x + u_i \leq b_i \cdot x + v_i)$, where a_i, b_i are constants such that $(a_i \neq 0) \vee (b_i \neq 0)$, u_i, v_i are linear terms free of x, and $n+1 \leq i \leq r$. Let us express each $a_i \cdot x$ appearing in the LMIs such that $a_i \neq 0$ in the equivalent form $2^{k_i} \cdot e_i \cdot x$, where $k_i = \kappa(x, a_i \cdot x)$ and e_i is an odd number. Similarly, let us express each $b_i \cdot x$ appearing in the LMIs such that $b_i \neq 0$ in the equivalent form $2^{k'_i} \cdot e'_i \cdot x$, where $k'_i = \kappa(x, b_i \cdot x)$ and e'_i is an odd number.

Without loss of generality, let k_1 be the minimum of $k_1, \ldots, k_m$. It can be observed that the LME $2^{k_1} \cdot x = t_1$ can be used to eliminate the occurrences of x in other LMEs, and in the LMDs and the LMIs with $\kappa(x, .)$ at least as large as k_1 in the following way.

- Each LME $2^{k_i} \cdot x = t_i$ for $2 \leq i \leq m$ can be equivalently expressed as $2^{\mu_i} \cdot t_1 = t_i$ where each $\mu_i = k_i - k_1$.
- Each LMD $2^{k_i} \cdot x \neq t_i$ for $m+1 \leq i \leq n$, such that $k_1 \leq k_i$ can be equivalently expressed as $2^{\mu_i} \cdot t_1 \neq t_i$ where each $\mu_i = k_i - k_1$.
- Each occurrence of x of the form $2^{k_i} \cdot e_i \cdot x$ in the LMIs for $n+1 \leq i \leq r$ such that $k_1 \leq k_i$ can be equivalently expressed as $2^{\mu_i} \cdot t_1 \cdot e_i$ where each $\mu_i = k_i - k_1$.
- Each occurrence of x of the form $2^{k'_i} \cdot e'_i \cdot x$ in the LMIs for $n+1 \leq i \leq r$ such that $k_1 \leq k'_i$ can be equivalently expressed as $2^{\mu'_i} \cdot t_1 \cdot e'_i$ where each $\mu'_i = k'_i - k_1$.

Hence, it can be observed that $\exists x.A$ can be equivalently expressed as $C_1 \wedge \exists x.C_2$, where C_1 is a conjunction of LMCs free of x, and C_2 is a conjunction of the LME $2^{k_1} \cdot x = t_1$ and those LMIs and LMDs from A with $\kappa(x, .)$ less than k_1, after substitution of the occurrences of $2^{k_1} \cdot x$ by t_1. □

Proposition 3, Lemma 1, and Lemma 2 yield us a simple heuristic *QE1_Layer1* that forms the core of Layer1. 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 2. If the $\kappa(x, \cdot)$ of all LMDs and LMIs in A are at least as large as k_1 (as in Lemma 2), then C_2 consists of the single LME $2^{k_1} \cdot x = t_1$. In this case, $\exists x.C_2$ simplifies to $2^{p-k_1} \cdot t_1 = 0$ (see Proposition 3), 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.

Analysis of complexity Consider a conjunction of LMCs with a subset of variables in its support to be eliminated. Let n be the number of LMCs in the conjunction, v be the number of variables its support, and e be the number of variables to be eliminated. It can be observed that for a variable x to be eliminated, Layer1 performs $O(n \cdot v)$ multiplications and additions in the worst-case. Assuming that arithmetic operations on p-bit numbers take time $O(Q(p))$ in the worst-case, where $Q(p)$ is a polynomial on p such that $p \leq Q(p) \leq p^3$, elimination of a variable hence has a worst-case time complexity of $O(n \cdot v \cdot Q(p))$. Observe that eliminating a variable does not increase the number of LMCs in the conjunction. Hence eliminating e variables has a worst-case time complexity of $O(e \cdot n \cdot v \cdot Q(p))$. Note that reading and writing an LMC with v variables in support takes $O(v \cdot p)$ time. Hence reading n LMCs as input and writing them back after eliminating the variables takes $O(n \cdot v \cdot p)$ time. Hence Layer1 has a worst-case time complexity of $O(e \cdot n \cdot v \cdot Q(p) + n \cdot v \cdot p)$. Since $p \leq Q(p) \leq p^3$, this reduces to $O(e \cdot n \cdot v \cdot Q(p)$.

3.4 Layer2: dropping unconstraining LMIs and LMDs

Formally, 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 say that D and I are *unconstraining* LMDs and LMIs, respectively, in such cases.

Given C, D and I satisfying conditions (i), (ii) and (iii) above, checking if condition (iv) holds requires solving a quantified bit-vector formula in general. This can be done by using an SMT solver such as Z3 that supports quantified bit-vector formulae. Alternatively bit-blasting followed by QBF solving or bit-level QE can be used. However applying such techniques can be expensive, as demonstrated in our experiments. In the following discussion, we focus on finding sufficient and polynomial time computable conditions for condition (iv) to hold.

Let $x[i]$ denote the ith 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 "adapting" 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 adapted 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 $x[i : j]$. For example, consider the LMCs $(x = y + z) \pmod 8$ and $(4y + z \leq x) \pmod 8$. Let θ_1 be the solution $x = 1, y = 1, z = 0$ of $(x = y + z) \pmod 8$, and let θ_2 be the solution $x = 5, y = 1, z = 0$ of $(4y + z \leq x) \pmod 8$. Note that θ_2 matches θ_1 except in the bits of slice $x[2 : 2]$. Hence we can say that θ_1 can be adapted with respect to slice $x[2 : 2]$ to satisfy $(4y + z \leq x) \pmod 8$.

The central idea in the second layer is to efficiently compute an under-approximation η of the number of ways in which an *arbitrary* solution of C can be adapted to satisfy $C \wedge D \wedge I$. It is easy to see that if $\eta \geq 1$, then $\exists x.(C) \Rightarrow \exists x.(C \wedge D \wedge I)$. We illustrate this idea below through an example. We will use this as a running example throughout this subsection.

Consider the problem of computing $\exists x.(C \wedge D \wedge I)$, where $C \equiv (z = 4x + y)$, $D \equiv (x \neq z + 7)$, and $I \equiv (6x + y \leq 4)$ and all LMCs have modulus 8. We claim that an arbitrary solution of C can be adapted to satisfy $C \wedge D \wedge I$. Note that C constrains only slice $x[0 : 0]$, whereas I constrains slice $x[0 : 1]$ and D constrains slice $x[0 : 2]$. Therefore, the value of slice $x[1 : 2]$ does not affect satisfaction of C, and the value of slice $x[2 : 2]$ does not affect satisfaction of $C \wedge I$. It can be observed that *any solution* of C can be adapted with respect to slice $x[1 : 1]$ to satisfy I by choosing value of slice $x[1 : 1]$ such that $6x$ lies between $-y$ and $4 - y$. Since $x[0 : 0]$ is unchanged, each such adapted solution must also satisfy $C \wedge I$. For example, the solution $x = 1, y = 0, z = 4$ of C can be adapted with respect to slice $x[1 : 1]$ to obtain the solution $x = 3, y = 0, z = 4$ of $C \wedge I$. Moreover, notice that *any solution* of $C \wedge I$ can be adapted with respect to slice $x[2 : 2]$ to satisfy D by choosing value for slice

$x[2:2]$ that differs from the most significant bit of $z+7$. Since $x[0:1]$ is unchanged, each such adapted solution also satisfies $C \wedge D \wedge I$. For example, the solution $x=3$, $y=0$, $z=4$ of $C \wedge I$ can be adapted with respect to slice $x[2:2]$ to obtain the solution $x=7$, $y=0, z=4$ of $C \wedge D \wedge I$. In this case, Layer2 computes the under-approximation η of the number of ways in which an arbitrary solution of C can be adapted to satisfy $C \wedge D \wedge I$ as ≥ 1, thus inferring that $\exists x.(C) \Rightarrow \exists x.(C \wedge D \wedge I)$.

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 defined above) 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 $\exists x.C_3 \equiv \exists x.C_2$.

Before presenting the details of computing η, we present the following proposition.

Proposition 4 *Let $x_1, \ldots, x_n$ be r-bit numbers and b be an r-bit odd number such that $b \cdot x_1, \ldots, b \cdot x_n$ take distinct consecutive values. Let ℓ be a number such that $1 \leq \ell \leq r$. If $n < 2^\ell$, then the values of $x_1[0:\ell-1], \ldots, x_n[0:\ell-1]$ are distinct. Otherwise, if $n \geq 2^\ell$, then the values of $x_1[0:\ell-1], \ldots, x_n[0:\ell-1]$ span the entire range $0, 1, \ldots, 2^\ell - 1$.*

Example Let x_1, x_2, x_3, x_4, x_5 respectively be 6, 1, 4, 7, 2, which are 3-bit numbers. Here $n=5$ and $r=3$. Suppose $b=3$. Note that $b \cdot x_1, b \cdot x_2, b \cdot x_3, b \cdot x_4, b \cdot x_5$ take distinct consecutive values 2, 3, 4, 5, 6 respectively.

– Case 1: Let ℓ be 3. Hence $n < 2^\ell$. The values of $x_1[0:\ell-1], x_2[0:\ell-1], x_3[0:\ell-1], x_4[0:\ell-1], x_5[0:\ell-1]$ are 6, 1, 4, 7, 2 respectively, which are distinct.
– Case 2: Let ℓ be 2. Hence $n \geq 2^\ell$. The values of $x_1[0:\ell-1], x_2[0:\ell-1], x_3[0:\ell-1], x_4[0:\ell-1], x_5[0:\ell-1]$ are 2, 1, 0, 3, 2 respectively, which span the entire range $0, 1, \ldots, 2^\ell - 1$.

Proof The proof is based on the following observations:

1. The values of $(b \cdot x_1)[0:\ell-1], \ldots, (b \cdot x_n)[0:\ell-1]$ are consecutive.
2. $(b \cdot x_i)[0:\ell-1]$ is equivalent to $b[0:\ell-1] \cdot x_i[0:\ell-1]$ for $1 \leq i \leq n$.
3. $b[0:\ell-1]$ is odd.

Since $b[0:\ell-1]$ is odd, it has a multiplicative inverse $(b[0:\ell-1])'$ modulo 2^ℓ. Note that $(b[0:\ell-1])'$ is also odd. Since $(b \cdot x_i)[0:\ell-1]$ is equivalent to $b[0:\ell-1] \cdot x_i[0:\ell-1]$ for $1 \leq i \leq n$, we get values of $x_1[0:\ell-1], \ldots, x_n[0:\ell-1]$ by multiplying the values of $(b \cdot x_1)[0:\ell-1], \ldots, (b \cdot x_n)[0:\ell-1]$ by $(b[0:\ell-1])'$ modulo 2^ℓ.

Observe that for $1 \leq i \leq n$ and $1 \leq j \leq n$ such that $i \neq j$, $x_i[0:\ell-1] = x_j[0:\ell-1]$ iff $(b \cdot x_i)[0:\ell-1] = (b \cdot x_j)[0:\ell-1]$. Since the values of $(b \cdot x_1)[0:\ell-1], \ldots, (b \cdot x_n)[0:\ell-1]$ are consecutive, it follows that, if $n < 2^\ell$, then the values of $x_1[0:\ell-1], \ldots, x_n[0:\ell-1]$ are distinct. If $n \geq 2^\ell$, then the values of $(b \cdot x_1)[0:\ell-1], \ldots, (b \cdot x_n)[0:\ell-1]$ are consecutive and they span the range $0, 1, \ldots, 2^\ell - 1$. Hence it is obvious that the values of $x_1[0:\ell-1], \ldots, x_n[0:\ell-1]$ also span the range $0, 1, \ldots, 2^\ell - 1$. □

Let I be $\bigwedge_{i=1}^{n}(l_i)$, where each l_i is an LMI of the form $s \bowtie t$, the operator $\bowtie$ is in $\{\leq, \geq\}$, s is a linear term with x in its support, and t 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 $s_j \leq 2^p - 1$. Suppose each I_j has 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 adapted 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 $\delta_j \leq \max(v_j - u_j + 1, 0)$ for every combination of values of other variables. 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 our running example, where $C \equiv (z = 4x+y)$, $D \equiv (x \neq z+7)$, and $I \equiv (6x+y \leq 4)$, we have $s_1 = 6x + y$ and $I_1 \equiv (6x + y \geq 0) \wedge (6x + y \leq 4) \wedge (6x + y \leq 7)$. Hence Z_1 is $(0 \leq 6x + y \leq 4)$ and thus $u_1 = 0$ and $v_1 = 4$. Note that $p = 3$, $k_1 = 1$, and the value of slice $x[2 : 2]$ does not affect the satisfaction of $(0 \leq 6x + y \leq 4)$. We are trying to compute N_1, the number of ways in which an arbitrary solution of $(z = 4x + y)$ can be adapted with respect to slice $x[0 : 1]$ to satisfy $(0 \leq 6x + y \leq 4)$. Treating $6x + y$ as a fresh variable f gives us $(0 \leq f \leq 4)$. As f can take five *consecutive* values in $(0 \leq f \leq 4)$, δ_1 is 5.

Let s be a linear term with x in its support. Let k be $\kappa(x, s)$. Let u, v respectively be arbitrary terms free of x which serve as lower and upper bounds of s. Let δ be the minimum number of *consecutive* values that s can take for every combination of values of other variables, if we were to treat s as a fresh p-bit variable and if $Z : u \leq s \leq v$ were to be satisfied. The following Lemma gives a lower bound for the number of distinct values that $x[0 : p - k - 1]$ can take while satisfying Z.

Lemma 3 *For every combination of values of variables other than x, there exist at least $\lfloor \delta/2^k \rfloor$ distinct values that $x[0 : p - k - 1]$ can take while satisfying Z.*

Example Let Z be $Z_1 : (0 \leq 6x + y \leq 4)$ from our running example. We have $p = 3$, $k = 1$ and $\delta = 5$. Note that, for every value of y, there are at least $\lfloor \delta/2^k \rfloor = \lfloor 5/2^1 \rfloor = 2$ distinct values that $x[0 : 1]$ can take while satisfying $(0 \leq 6x + y \leq 4)$.

Proof δ is the minimum number of *consecutive* values that s can take for every combination of values of other variables, if we were to treat s as a fresh p-bit variable and if $Z : u \leq s \leq v$ were to be satisfied. However, in general, s is of the form $2^k \cdot b \cdot x + w$, where w is a linear term free of x, and b is an odd number.

There are at least $\lfloor \delta/2^k \rfloor$ multiples of 2^k among δ consecutive values. Hence, for every combination of values of other variables, there exist at least $\lfloor \delta/2^k \rfloor$ values that $2^k \cdot b \cdot x$ can take while satisfying Z. The least significant k bits of these values are all zeros. Moreover, the values of the most significant $p - k$ bits, i.e., the values of slice $(2^k \cdot b \cdot x)[k : p - 1]$ are consecutive. Note that slice $(2^k \cdot b \cdot x)[k : p - 1]$ is the same as slice $(b \cdot x)[0 : p - k - 1]$. Also $(b \cdot x)[0 : p - k - 1]$ is equivalent to $b[0 : p - k - 1] \cdot x[0 : p - k - 1]$. Therefore, for every combination of values of variables other than x, there exist at least $\lfloor \delta/2^k \rfloor$ *consecutive* values that $b[0:p-k-1] \cdot x[0:p-k-1]$ can take while satisfying Z.

Since b is odd, $b[0:p-k-1]$ is odd. Let us apply Proposition 4 on these *consecutive* values of $b[0:p-k-1] \cdot x[0:p-k-1]$ with $n = \lfloor \delta/2^k \rfloor$, $r = \ell = p-k$ and $b = b[0:p-k-1]$. Note that $n = \lfloor \delta/2^k \rfloor < 2^\ell = 2^{p-k}$ here, since $\delta < 2^p$. Therefore, using Proposition 4, we have: for every combination of values of variables other than x, there exist at least $n = \lfloor \delta/2^k \rfloor$ *distinct* values that $x[0:p-k-1]$ can take while satisfying Z.

□

Lemma 3 indicates that there are at least $\lfloor \delta_j/2^{k_j} \rfloor$ ways in which an arbitrary solution of C can be adapted with respect to slice $x[0:p-k_j-1]$ to satisfy Z_j. Hence, $N_j \geq \lfloor \delta_j/2^{k_j} \rfloor$. For notational convenience, we denote $\lfloor \delta_j/2^{k_j} \rfloor$ by $\widehat{N_j}$.

To understand how δ_j is computed in general, 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 $\delta_{g,h} \leq \max(t_h - t_g + 1, 0)$ for every combination of values of t_h and t_g. 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 using the following Proposition.

Proposition 5 *1. If t_g and t_h are constants and $t_h \geq t_g$, then $\delta_{g,h} = t_h - t_g + 1$.*

2. *If t_h is a constant, t_g can be expressed as $2^\tau \cdot t$, where τ is an integer such that* $0 \leq \tau \leq p-1$, *and* $t_h \geq 2^p - 2^\tau$, *then* $\delta_{g,h} = t_h - (2^p - 2^\tau) + 1$.
3. *If t_g is a constant, t_h can be expressed as $2^\tau \cdot t + a$, where τ is an integer such that* $0 \leq \tau \leq p-1$, *and* $a \bmod 2^\tau \geq t_g$, *then* $\delta_{g,h} = a \bmod 2^\tau - t_g + 1$.
4. *Otherwise* $\delta_{g,h} = 0$.

Example 1. Suppose $t_g = 1$ and $t_h = 6$. Therefore, $\max(t_h - t_g + 1, 0) = t_h - t_g + 1 = 6$. Since $\delta_{g,h} \leq \max(t_h - t_g + 1, 0)$, we can set $\delta_{g,h}$ to 6.

2. Suppose $t_g = 4y$, $t_h = 14$, and $p = 4$. Here t_g is of the form $2^\tau \cdot t$, where $\tau = 2$ and $t = y$. Observe that the maximum possible value of $4y$ with modulus 16 is $2^p - 2^\tau = 12$, i.e., $4y \leq 12$. Therefore, $t_h - t_g + 1 = 14 - 4y + 1 \geq 14 - 12 + 1 = 3$. Hence $\max(t_h - t_g + 1, 0) \geq 3$. Therefore 3 can be used as $\delta_{g,h}$.
3. Suppose $t_g = 0$, $t_h = 4y + 7$, and $p = 4$. Here t_h is of the form $2^\tau \cdot t + a$, where $\tau = 2$, $t = y$, and $a = 7$. Observe that the minimum possible value of $4y + 7$ with modulus 16 is $a \bmod 2^\tau = 7 \bmod 4 = 3$, i.e., $4y + 7 \geq 3$. Therefore, $t_h - t_g + 1 = (4y + 7) - 0 + 1 \geq 3 - 0 + 1 = 4$. Hence $\max(t_h - t_g + 1, 0) \geq 4$. Therefore 4 can be used as $\delta_{g,h}$.
4. Suppose $t_g = y$, $t_h = z$. In such cases we set $\delta_{g,h}$ to 0.

Proof $\delta_{g,h}$ is an integer in $\{0, \ldots, 2^p - 1\}$ such that $\delta_{g,h} \leq \max(t_h - t_g + 1, 0)$ for every combination of values of t_h and t_g.

1. If t_g and t_h are constants and $t_h \geq t_g$, then $\max(t_h - t_g + 1, 0)$ reduces to $t_h - t_g + 1$. Therefore, it is obvious that $t_h - t_g + 1$ can be used as $\delta_{g,h}$.
2. Consider the case when t_h is a constant, t_g can be expressed as $2^\tau \cdot t$, where τ is an integer such that $0 \leq \tau \leq p-1$, and $t_h \geq 2^p - 2^\tau$. Since t_g is a multiple of 2^τ, the possible values of t_g are $0, 2^\tau, \ldots, 2^p - 2^\tau$. Hence the maximum possible value of t_g is $2^p - 2^\tau$, i.e., $t_g \leq 2^p - 2^\tau$. This implies that $t_h - t_g + 1 \geq t_h - (2^p - 2^\tau) + 1$. Therefore $\max(t_h - t_g + 1, 0) \geq t_h - (2^p - 2^\tau) + 1$. Hence $t_h - (2^p - 2^\tau) + 1$ can be used as $\delta_{g,h}$.
3. Consider the case when t_g is a constant, t_h can be expressed as $2^\tau \cdot t + a$, where τ is an integer such that $0 \leq \tau \leq p-1$, and $a \bmod 2^\tau \geq t_g$. Let $a = 2^\tau \cdot a_1 + a_2$, where

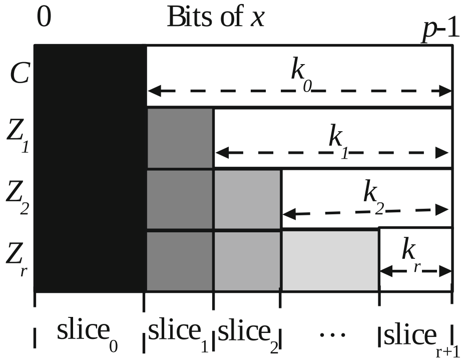


Fig. 2 Slicing of bits of x by $k_0, \dots, k_r$

$a_2 = a \bmod 2^\tau$ and $a_1 \geq 0$. Hence t_h can be expressed as $2^\tau \cdot (t + a_1) + a_2$. Since $2^\tau \cdot (t + a_1)$ is a multiple of 2^τ, the possible values of $2^\tau \cdot (t + a_1)$ are $0, 2^\tau, \dots, 2^p - 2^\tau$. Hence the possible values of t_h are $a_2, 2^\tau + a_2, \dots, 2^p - 2^\tau + a_2$. Therefore, the minimum possible value of t_h is a_2, i.e., $t_h \geq a_2$, which implies that $t_h - t_g + 1 \geq a_2 - t_g + 1$. Therefore $\max(t_h - t_g + 1, 0) \geq a_2 - t_g + 1$. Hence $a_2 - t_g + 1$, i.e., $a \bmod 2^\tau - t_g + 1$ can be used as $\delta_{g,h}$.

4. Consider the case when none of the above conditions is true. Since $0 \leq \max(t_h - t_g + 1, 0)$, we can use $\delta_{g,h}$ as 0 in this case. □

Let D be $\bigwedge_{i=1}^{m}(d_i)$, where each d_i is an LMD of the form $2^{\kappa(x,d_i)} \cdot x \neq t_{d_i}$, where t_{d_i} is a linear term free of x. 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 $k_1 > \dots > k_r$. We partition the bits of x into $r + 2$ slices as shown in Fig. 2, where $\mathsf{slice_0}$ represents $x[0 : p - k_0 - 1]$, $\mathsf{slice_j}$ represents $x[p - k_{j-1} : p - k_j - 1]$ for $1 \leq j \leq r$, and $\mathsf{slice_{r+1}}$ represents $x[p - k_r : p - 1]$. Note that the value of $\mathsf{slice_0}$ potentially affects the satisfaction of C as well as that of Z_1 through Z_r, the value of $\mathsf{slice_j}$ potentially affects the satisfaction of Z_j through Z_r for $1 \leq j \leq r$, and the value of $\mathsf{slice_{r+1}}$ does not affect the satisfaction of any Z_j or C. Recall that similar slicing schemes were used in [5,9,18] for converting conjunctions of bit-vector constraints into equisatisfiable constraints on slices of variables. However such slicing schemes were used for a different objective of simplifying constraints and solving them.

Let Z_0 denote True. Let θ be a solution of $C \wedge Z_0 \wedge \dots \wedge Z_i$, where $0 \leq i < r$. Note that bits in $\mathsf{slice_{i+1}}$ through $\mathsf{slice_{r+1}}$ do not affect satisfaction of $C \wedge Z_0 \wedge \dots \wedge Z_i$. Let $Y_{i,j}$ denote the number of ways in which θ can be adapted with respect to bits in $\mathsf{slice_{i+1}}$ through $\mathsf{slice_j}$, to satisfy Z_j, where $i < j \leq r$. Since $\mathsf{slice_0}$ through $\mathsf{slice_i}$ are unchanged, each such adapted solution must also satisfy $C \wedge Z_0 \wedge \dots \wedge Z_i$.

Lemma 4 *An arbitrary solution of* $C \wedge Z_0 \wedge \dots \wedge Z_i$ *for* $0 \leq i < r$ *can be adapted with respect to bits in* $\mathsf{slice_{i+1}}$ *through* $\mathsf{slice_j}$, *to satisfy* Z_j *for* $i < j \leq r$ *in at least* $\lfloor \widehat{N_j} / 2^{p-k_i} \rfloor$ *ways. Moreover, if we focus only on* $\mathsf{slice_{i+1}}$, *then there are at least* $\min(\lfloor \widehat{N_j} / 2^{p-k_i} \rfloor, 2^{k_i - k_{i+1}})$ *distinct values of* $\mathsf{slice_{i+1}}$ *in the corresponding adapted solutions.*

Example In our running example, since $p = 3$, $k_0 = 2$, $k_1 = 1$, the bits of x are partitioned into three slices: $\mathsf{slice_0}$ is $x[0 : 0]$, $\mathsf{slice_1}$ is $x[1 : 1]$ and $\mathsf{slice_2}$ is $x[2 : 2]$. Clearly, the value of $\mathsf{slice_0}$ potentially affects the satisfaction of $C : (z = 4x + y)$ as well as that of $Z_1 : (0 \leq 6x + y \leq 4)$. The value of $\mathsf{slice_1}$ potentially affects the satisfaction of Z_1, but not that of C, and the value of slice_2 does not affect the satisfaction of C or Z_1. Let θ be a solution of C. Using Lemma 4, there exists at least $\lfloor \widehat{N_1}/2^{p-k_1} \rfloor = \lfloor 2/2^{3-2} \rfloor = 1$ way in which θ can be adapted with respect to bits in slice_1 to satisfy Z_1. Since slice_0 is unchanged, the adapted solution must satisfy $C \wedge Z_1$.

Proof Recall from Proof of Lemma 3 that for every combination of values of variables other than x, there exist at least $\widehat{N_j}$ *consecutive* values that $b_j[0 : p - k_j - 1] \cdot x[0 : p - k_j - 1]$ can take while satisfying Z_j, where $b_j[0 : p - k_j - 1]$ is odd. Note that $i < j$, $k_i > k_j$ and $p - k_i < p - k_j$. We make use of the following claims.

Claim 1 *For every combination of values of variables other than x, (i) if* $\widehat{N_j} < 2^{p-k_i}$*, then there exist at least* $\widehat{N_j}$ *distinct values that* $x[0 : p - k_i - 1]$ *can take while satisfying* Z_j*, and (ii) if* $\widehat{N_j} \geq 2^{p-k_i}$*, the values that* $x[0 : p - k_i - 1]$ *can take while satisfying* Z_j *span the entire range* $0, 1, \ldots, 2^{p-k_i} - 1$.

Claim 2 *For every combination of values of variables other than x, (i) if* $\widehat{N_j} < 2^{p-k_{i+1}}$*, then there exist at least* $\widehat{N_j}$ *distinct values that* $x[0 : p - k_{i+1} - 1]$ *can take while satisfying* Z_j*, and (ii) if* $\widehat{N_j} \geq 2^{p-k_{i+1}}$*, the values that* $x[0 : p - k_{i+1} - 1]$ *can take while satisfying* Z_j *span the entire range* $0, 1, \ldots, 2^{p-k_{i+1}} - 1$.

Claim 1 can be proved by applying Proposition 4 on the *consecutive* values of $b_j[0 : p-k_j-1] \cdot x[0 : p-k_j-1]$ with $n = \widehat{N_j}$, $r = p-k_j$, $\ell = p-k_i$ and $b = b_j[0 : p-k_j-1]$. Similarly, Claim 2 can be proved by applying Proposition 4 on the *consecutive* values of $b_j[0 : p - k_j - 1] \cdot x[0 : p - k_j - 1]$ with $n = \widehat{N_j}$, $r = p - k_j$, $\ell = p - k_{i+1}$ and $b = b_j[0 : p - k_j - 1]$.

Using Lemma 3, we know that, for every combination of values of variables other than x, there exist at least $\widehat{N_j}$ *distinct* values that can be assigned to $x[0 : p - k_j - 1]$ (i.e., bits in slice_0 through $\mathsf{slice}_\mathsf{j}$) while satisfying Z_j. Lemma 3 and Claim 1 together imply that for every combination of values of variables other than x and for any arbitrary value of $x[0 : p - k_i - 1]$ (i.e., bits in slice_0 through $\mathsf{slice}_\mathsf{i}$), there exist at least $\lfloor \widehat{N_j}/2^{p-k_i} \rfloor$ *distinct* values that can be assigned to $x[p - k_i : p - k_j - 1]$ (i.e., bits in $\mathsf{slice}_{\mathsf{i+1}}$ through $\mathsf{slice}_\mathsf{j}$) while satisfying Z_j. Hence, an arbitrary solution of $C \wedge Z_0 \wedge \cdots \wedge Z_i$ for $0 \leq i < r$ can be adapted with respect to bits in $\mathsf{slice}_{\mathsf{i+1}}$ through $\mathsf{slice}_\mathsf{j}$, to satisfy Z_j for $i < j \leq r$ in at least $\lfloor \widehat{N_j}/2^{p-k_i} \rfloor$ ways.

In order to prove our claim on values of $\mathsf{slice}_{\mathsf{i+1}}$ in the corresponding adapted solutions, note that, from Claim 2 we know that, for every combination of values of variables other than x, there exist at least $\min(\widehat{N_j}, 2^{p-k_{i+1}})$ *distinct* values that $x[0 : p - k_{i+1} - 1]$ can take while satisfying Z_j. Hence Claim 1 and Claim 2 together imply that, for every combination of values of variables other than x and for any arbitrary value of $x[0 : p - k_i - 1]$ (i.e., bits in slice_0 through $\mathsf{slice}_\mathsf{i}$), there exist at least $\min(\lfloor \widehat{N_j}/2^{p-k_i} \rfloor, \lfloor 2^{p-k_{i+1}}/2^{p-k_i} \rfloor) = \min(\lfloor \widehat{N_j}/2^{p-k_i} \rfloor, 2^{k_i-k_{i+1}})$ *distinct* values that can be assigned to $x[p - k_i : p - k_{i+1} - 1]$ (i.e., bits in $\mathsf{slice}_{\mathsf{i+1}}$) while satisfying Z_j. Therefore, if we focus only on $\mathsf{slice}_{\mathsf{i+1}}$ in the aforementioned adapted solutions, then there are at least $\min(\lfloor \widehat{N_j}/2^{p-k_i} \rfloor, 2^{k_i-k_{i+1}})$ *distinct* values of $\mathsf{slice}_{\mathsf{i+1}}$. □

Using Lemma 4, we have $Y_{i,j} \geq \lfloor \widehat{N_j}/2^{p-k_i} \rfloor$. For notational convenience, let us denote $\min(\lfloor \widehat{N_j}/2^{p-k_i} \rfloor, 2^{k_i-k_{i+1}})$ by $\alpha_{i,j}$.

Lemma 4 indicates that a solution θ of $C \wedge Z_0 \wedge \cdots \wedge Z_i$ for $0 \leq i < r$ can be adapted to satisfy $C \wedge Z_0 \wedge \cdots \wedge Z_i \wedge Z_j$ for $i < j \leq r$ by using at least $\alpha_{i,j}$ different values of $\mathsf{slice}_{\mathsf{i+1}}$. Let the corresponding set of values of $\mathsf{slice}_{\mathsf{i+1}}$ 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 $\mathsf{slice}_{\mathsf{i+1}}$ that permits us to adapt θ with respect to $\mathsf{slice}_{\mathsf{i+1}}$ through $\mathsf{slice}_{\mathsf{r}}$ to satisfy Z_{i+1} through Z_r, respectively. It is therefore desirable to have $|\bigcap_{j=i+1}^{r} S_{i+1,j}^{\theta}| \geq 1$. Using the Inclusion-Exclusion principle, we find that $|\bigcap_{j=i+1}^{r} S_{i+1,j}^{\theta}| \geq (\sum_{j=i+1}^{r} \alpha_{i,j}) - (r - i - 1) \cdot 2^{k_i - k_{i+1}}$. Note that the lower bound is independent of θ. For notational convenience, let us denote the lower bound by W_{i+1}.

If $W_{i+1} \geq 1$ for all $i \in \{0, \ldots r - 1\}$, an arbitrary solution θ of C can be adapted to satisfy $C \wedge Z_0 \wedge \cdots \wedge Z_r$ as follows. Since $W_1 \geq 1$, we choose a value of $\mathsf{slice}_{\mathsf{1}}$, say v_1, from $\bigcap_{j=1}^{r} S_{1,j}^{\theta}$. Let θ_1 denote θ with $\mathsf{slice}_{\mathsf{1}}$ (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 $\mathsf{slice}_{\mathsf{2}}$, say v_2, from $\bigcap_{j=2}^{r} S_{2,j}^{\theta_1}$, and repeat the procedure until we have chosen values for $\mathsf{slice}_{\mathsf{1}}$ through $\mathsf{slice}_{\mathsf{r}}$. Finally, since $\mathsf{slice}_{\mathsf{r+1}}$ does not affect the satisfaction of C or of any Z_i, we can choose an arbitrary value for $\mathsf{slice}_{\mathsf{r+1}}$. Clearly, there are at least $(\prod_{i=0}^{r-1} |W_{i+1}|) \cdot 2^{k_r}$ ways in which values of different slices can be chosen, so as to adapt θ to satisfy $C \wedge Z_0 \wedge \cdots \wedge Z_r$. Let us denote $(\prod_{i=0}^{r-1} |W_{i+1}|) \cdot 2^{k_r}$ by μ_I.

In our running example, we have, $Y_{0,1} \geq \lfloor \widehat{N_1}/2^{p-k_0} \rfloor = 1$. Also $\alpha_{0,1} = \min(\lfloor \widehat{N_1}/2^{p-k_0} \rfloor, 2^{k_0-k_1}) = \min(1, 2^{2-1}) = 1$. Hence $W_1 = (\sum_{j=1}^{1} \alpha_{0,1}) - (1 - 0 - 1) \cdot 2^{k_0-k_1} = \alpha_{0,1} = 1$. Note that there is at least one way of adapting an arbitrary solution of $(z = 4x + y)$ with respect to $\mathsf{slice}_{\mathsf{1}}$ to satisfy $(z = 4x + y) \wedge (0 \leq 6x + y \leq 4)$. Moreover, there are at least two ways of adapting an arbitrary solution of $(z = 4x + y)$ with respect to $\mathsf{slice}_{\mathsf{1}}$ through to $\mathsf{slice}_{\mathsf{2}}$ to satisfy $(z = 4x + y) \wedge (0 \leq 6x + y \leq 4)$ as indicated by $\mu_I = W_1 \cdot 2^{k_1} = 1 \cdot 2^1 = 2$.

Let us now consider each LMD d_i in D. Recall that each d_i is of the form $2^{\kappa(x,d_i)} \cdot x \neq t_{d_i}$. Note that d_i constrains only slice $x[0 : p - \kappa(x, d_i) - 1]$. It can be observed that for every combination of values of variables other than x, the only way to violate d_i is to choose value of slice $x[0 : p - \kappa(x, d_i) - 1]$ to be the same as the value of $t_{d_i}[\kappa(x, d_i) : p - 1]$. Hence, for every combination of values of variables other than x, there is at most one way of choosing value for slice $x[0 : p - \kappa(x, d_i) - 1]$ such that d_i is violated. Since slice $x[p - \kappa(x, d_i) : p - 1]$ is not constrained by d_i, this means that for every combination of values of variables other than x, there are at most $2^{\kappa(x,d_i)}$ ways of choosing values for $\mathsf{slice}_{\mathsf{0}}$ through $\mathsf{slice}_{\mathsf{r+1}}$ such that d_i is violated. Therefore, for every combination of values of variables other than x, $\sum_{i=1}^{m}(2^{\kappa(x,d_i)})$ is an over-approximation of the number ways of choosing values for $\mathsf{slice}_{\mathsf{0}}$ through $\mathsf{slice}_{\mathsf{r+1}}$ such that D is violated. Let us denote $\sum_{i=1}^{m}(2^{\kappa(x,d_i)})$ by μ_D. We have already seen that there are at least μ_I ways of adapting an arbitrary solution θ of C to satisfy $C \wedge Z_0 \wedge \cdots \wedge Z_r$. As μ_D is an over-approximation of the number of such adapted solutions that can violate D, there are at least $\mu_I - \mu_D$ ways of adapting θ to satisfy $C \wedge Z_0 \wedge \cdots \wedge Z_r \wedge D$. We denote $\mu_I - \mu_D$ by η.

In the running example, we have, $d_1 \equiv (x \neq z + 7)$ and $\kappa(x, d_1) = 0$. Note that for every value of $z + 7$, there is at most one way of choosing value for slice $x[0 : 2]$ such that d_1 is violated. Here $\mu_D = 2^{\kappa(x,d_1)} = 1$, and hence $\eta = \mu_I - \mu_D = 1$. Thus there is at least one way of adapting an arbitrary solution of $(z = 4x + y)$ to satisfy $(z = 4x + y) \wedge (0 \leq 6x + y \leq 4) \wedge (x \neq z + 7)$.

The above reasoning can be extended to the general case $k_1 \geq \cdots \geq k_r$. Let π_i for $0 \leq i < r$ be the number of Z_j's with $k_j < k_i$ for $i < j \leq r$. Using the Inclusion-Exclusion principle, W_{i+1} above then changes to $(\sum_{j=i+1}^{r} \alpha_{i,j}) - (\pi_i - 1) \cdot 2^{k_i - k_{i+1}}$.

Theorem 1 *If* $\eta \geq 1$, *then* $\exists x.(C \wedge D \wedge I) \equiv \exists x.(C)$

Proof There are at least η ways of adapting an arbitrary solution of C to satisfy $C \wedge Z_0 \wedge \cdots \wedge Z_r \wedge D$. If $\eta \geq 1$, then an arbitrary solution of C can be adapted to satisfy $C \wedge Z_0 \wedge \cdots \wedge Z_r \wedge D$, and hence $\exists x.(C) \Rightarrow \exists x.(C \wedge D \wedge I)$. Since $\exists x.(C \wedge D \wedge I) \Rightarrow \exists x.(C)$ always holds, we have $\exists x.(C \wedge D \wedge I) \equiv \exists x.(C)$ if $\eta \geq 1$. □

It can be observed that η is computable in polynomial time. The difficult step is computation of μ_I. Let r be the number of distinct linear terms in I with x in their support. Computing μ_I requires $O(r^2)$ arithmetic operations in the worst-case.

As mentioned earlier, the procedure *QE1_Layer2* applies this technique to problem instances of the form $\exists x.C_2$, obtained after invoking *QE1_Layer1* to find unconstraining LMDs and LMIs. If all the LMIs and LMDs in $\exists x.C_2$ are unconstraining, then $\exists x.C_2$ reduces to $\exists x.(2^{k_1} \cdot x = t_1)$, and *QE1_Layer2* returns the equivalent form $2^{p-k_1} \cdot t_1 = 0$.

In the running example, *QE1_Layer2* drops the LMI $(6x+y \leq 4)$ and the LMD $(x \neq z+7)$ as they are unconstraining in $\exists x.((z = 4x + y) \wedge (6x + y \leq 4) \wedge (x \neq z + 7))$. The problem instance thus reduces to $\exists x.(z = 4x + y)$, which is equivalent to $(4y + 4z = 0)$. Hence the final result is $(4y + 4z = 0)$.

In general, *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.

Analysis of complexity Consider a conjunction of LMCs with a subset of variables in its support to be eliminated. Let n be the number of LMCs in the conjunction, v be the number of variables in its support, and e be the number of variables to be eliminated. Consider the elimination of a variable x inside Layer2. Recall that Layer2 can be applied only when all LMIs involving x are of the form $s \bowtie t$, where $\bowtie \in \{\leq, \geq\}$, s is a linear term with x in its support, and t is a linear term free of x. Let r be the number of distinct linear terms with x in the support appearing in the LMIs. As observed above, computing η requires $O(r^2)$ arithmetic operations in the worst-case. Note that $r \leq n$. Assuming that each arithmetic operation on p-bit numbers take time $O(Q(p))$ in the worst-case, where $p \leq Q(p) \leq p^3$, elimination of a variable hence has a worst-case time complexity of $O(n^2 \cdot Q(p))$. Observe that eliminating a variable does not increase the number of LMCs in the conjunction. Hence eliminating e variables has a worst-case time complexity of $O(e \cdot n^2 \cdot Q(p))$. Since reading n LMCs as input and writing the result takes $O(n \cdot v \cdot p)$ time, Layer2 has a worst-case time complexity of $O(e \cdot n^2 \cdot Q(p) + n \cdot p \cdot v)$.

3.5 Layer3: Fourier–Motzkin elimination for LMIs

In this subsection, we present a Fourier–Motzkin (FM) style QE algorithm for computing $\exists x.C_3$ obtained above. Recall that C_3 obtained above, in general, contains LMDs, LMIs, and a single LME. We propose converting the LMDs and the LME in C_3 to LMIs using the equivalences $(t_1 = t_2) \equiv (t_1 \geq t_2) \wedge (t_1 \leq t_2)$ and $(t_1 \neq t_2) \equiv \neg(t_1 = t_2)$. This, in general, converts C_3 to a Boolean combination of LMIs. However, as we will see in Sect. 4, a QE algorithm for conjunctions of LMIs can be extended to a QE algorithm for Boolean combinations of LMIs. Hence, in the remainder of this subsection, we will focus on QE from *conjunctions of LMIs*.

There are two fundamental problems when trying to apply FM elimination for reals [20] to a conjunction of LMIs:

1. *Wrap-around behaviour* 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, due to wrap-around behaviour, the equivalences (i) $(t_1 \leq t_2) \equiv (t_1 + t_3 \leq t_2 + t_3)$ and (ii) $(t_1 \leq t_2) \equiv (a \cdot t_1 \leq a \cdot t_2)$ used for normalizing inequalities do not hold for LMIs in general. For example, $(2 \leq 3 \pmod 4)$, but $(2 + 1 > 3 + 1 \pmod 4)$. Similarly, $(1 \leq 2 \pmod 4)$, but $(1 \cdot 2 > 2 \cdot 2 \pmod 4)$. Hence, normalizing an LMI w.r.t. a variable is much more difficult than normalizing in the case of reals. Moreover, unlike in the case of reals and integers, presence of equalities does

not always simplify QE in modular arithmetic. For example, $\exists x.((2x = 3y + 2) \wedge (3x > 4z + 3))$ can be simplified to $\exists x.((6x = 9y + 6) \wedge (6x > 8z + 6))$ on integers. However this simplification cannot be done in modular arithmetic in general.

2. *Lack of density* 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 normalized LMIs. For example $\exists x.((y \leq 4x) \wedge (4x \leq z))$ is equivalent to $(y \leq z)$ in reals, whereas this is not true in modular arithmetic in general.

This motivates us to (i) define a (weak) normal form for LMIs, and (ii) adapt FM elimination to achieve QE from normalized LMIs. Recall that Omega Test [51] also defines a normal form for inequalities over integers, and adapts FM elimination over reals for QE from normalized inequalities over integers. However, Omega Test cannot be directly used for QE from LMIs—using Omega Test for QE from LMIs requires converting the LMIs to equivalent constraints in linear integer arithmetic; the resulting formula is in linear integer arithmetic, and converting the resulting formula back to modular arithmetic is difficult. Moreover our experiments in Sect. 5 indicate that, using Omega Test for QE from the linear integer arithmetic constraints arising from LMIs incurs considerable performance overhead.

3.5.1 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 $\Theta(t_1, t_2)$ 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 $\Theta(t_1, t_2)$ 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 the LMI $(x + 2 \leq y)$ modulo 8 w.r.t. x. Adding the additive inverse of 2 modulo 8, i.e, 6 to both sides of the LMI, the left-hand side $x + 2$ changes to x and the right-hand side y changes to $y + 6$. However, note that $(x + 2 \leq y)$ is not equivalent to $(x \leq y + 6)$. If $\Theta(x + 2, 6) \equiv \Theta(y, 6)$, then $(x + 2 \leq y) \equiv (x \leq y + 6)$ holds; otherwise $(x + 2 \leq y) \equiv (x > y + 6)$ holds. Note that $\Theta(x + 2, 6) \equiv \Theta(y, 6)$ 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 $(\alpha \wedge \beta) \vee (\neg\alpha \wedge \gamma)$.

In this example, the Θ predicate allowed us to perform a case-split and normalize each branch. The following Lemma generalizes this idea.

Lemma 5 *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*.*

Before we present the proof of Lemma 5, it would be useful to present a proposition.

Proposition 6 *Let* l_1 *be an LMI* $t_1 \leq t_2$*, and let* t_3 *be a linear term. Then* $l_1 \equiv ite(\varphi_1 \wedge (\varphi_2 \oplus \varphi_3), (t_1 + t_3 > t_2 + t_3), (t_1 + t_3 \leq t_2 + t_3))$*, where* $\varphi_1 \equiv (t_3 \neq 0)$*,* $\varphi_2 \equiv (-t_3 \leq t_1)$*,* $\varphi_3 \equiv (-t_3 \leq t_2)$ *and* $\varphi_2 \oplus \varphi_3$ *denotes exclusive-or of* φ_2 *and* φ_3*.*

Proof Note that $(t_1 \leq t_2) \equiv \psi_1 \vee \psi_2 \vee \psi_3 \vee \psi_4$, where

- $\psi_1 \equiv (t_1 \leq t_2) \wedge \Theta(t_1, t_3) \wedge \Theta(t_2, t_3)$
- $\psi_2 \equiv (t_1 \leq t_2) \wedge \Theta(t_1, t_3) \wedge \neg\Theta(t_2, t_3)$
- $\psi_3 \equiv (t_1 \leq t_2) \wedge \neg\Theta(t_1, t_3) \wedge \Theta(t_2, t_3)$
- $\psi_4 \equiv (t_1 \leq t_2) \wedge \neg\Theta(t_1, t_3) \wedge \neg\Theta(t_2, t_3)$

It can be seen that,

- $\psi_1 \equiv (t_1 + t_3 \leq t_2 + t_3) \wedge \Theta(t_1, t_3) \wedge \Theta(t_2, t_3)$
- $\psi_2 \equiv \mathsf{false}$, since $\Theta(t_1, t_3) \wedge \neg\Theta(t_2, t_3) \Rightarrow (t_1 > t_2)$. However, we can write ψ_2 as $(t_1 + t_3 > t_2 + t_3) \wedge \Theta(t_1, t_3) \wedge \neg\Theta(t_2, t_3)$ as well, which is equivalent to false, since $\Theta(t_1, t_3) \wedge \neg\Theta(t_2, t_3) \Rightarrow (t_1 + t_3 < t_2 + t_3)$.
- $\psi_3 \equiv (t_1 + t_3 > t_2 + t_3) \wedge \neg\Theta(t_1, t_3) \wedge \Theta(t_2, t_3)$
- $\psi_4 \equiv (t_1 + t_3 \leq t_2 + t_3) \wedge \neg\Theta(t_1, t_3) \wedge \neg\Theta(t_2, t_3)$

Expressing $\psi_1 \vee \psi_2 \vee \psi_3 \vee \psi_4$ in terms of ites, we have,

$$(t_1 \leq t_2) \equiv \mathsf{ite}(\Theta(t_1, t_3) \oplus \Theta(t_2, t_3), (t_1 + t_3 > t_2 + t_3), (t_1 + t_3 \leq t_2 + t_3))$$

Expanding the Θ's using the formula $\Theta(\alpha, \beta) \equiv (\beta \neq 0) \wedge (\alpha \geq -\beta)$, where α, β are linear terms, we have,

$$(t_1 \leq t_2) \equiv \mathsf{ite}(\varphi_1 \wedge (\varphi_2 \oplus \varphi_3), (t_1 + t_3 > t_2 + t_3), (t_1 + t_3 \leq t_2 + t_3))$$

where, $\varphi_1 \equiv (t_3 \neq 0)$, $\varphi_2 \equiv (-t_3 \leq t_1)$, and $\varphi_3 \equiv (-t_3 \leq t_2)$. □

We can now prove Lemma 5.

Proof (*Proof of Lemma* 5) Consider an LMI $l_1 : a \cdot x + t_1 \leq b \cdot x + t_2$, where t_1 and t_2 are linear terms without x in their supports. Using Proposition 6, with $a \cdot x + t_1$ in place of t_1, $b \cdot x + t_2$ in place of t_2 and $-b \cdot x - t_1$ in place of t_3,

$$l_1 \equiv \mathsf{ite}(\varphi_1 \wedge (\varphi_2 \oplus \varphi_3), (a \cdot x - b \cdot x > t_2 - t_1), (a \cdot x - b \cdot x \leq t_2 - t_1))$$

where, $\varphi_1 \equiv (b \cdot x + t_1 \neq 0)$, $\varphi_2 \equiv (b \cdot x + t_1 \leq a \cdot x + t_1)$, and $\varphi_3 \equiv (b \cdot x + t_1 \leq b \cdot x + t_2)$.

Note that the LMIs $(a \cdot x - b \cdot x > t_2 - t_1)$ and $(a \cdot x - b \cdot x \leq t_2 - t_1)$ are normalized w.r.t. x, whereas φ_2 and φ_3 are not. Hence, let us try to normalize φ_2 and φ_3 w.r.t. x.

Consider $\varphi_2 \equiv (b \cdot x + t_1 \leq a \cdot x + t_1)$. Using Proposition 6, with $b \cdot x + t_1$ in place of t_1, $a \cdot x + t_1$ in place of t_2 and $-t_1$ in place of t_3,

$$\varphi_2 \equiv ite((t_1 \neq 0) \wedge ((t_1 \leq a \cdot x + t_1) \oplus (t_1 \leq b \cdot x + t_1)), (b \cdot x > a \cdot x), (b \cdot x \leq a \cdot x))$$

Using the observations $(\beta \leq \alpha + \beta) \equiv \neg\Theta(\alpha, \beta)$ and $\Theta(\alpha, \beta) \equiv (\beta \neq 0) \wedge (\alpha \geq -\beta)$ for linear terms α and β, and simplifying, $(t_1 \neq 0) \wedge ((t_1 \leq a \cdot x + t_1) \oplus (t_1 \leq b \cdot x + t_1))$ is equivalent to $(t_1 \neq 0) \wedge ((-t_1 \leq a \cdot x) \oplus (-t_1 \leq b \cdot x))$. Hence,

$$\varphi_2 \equiv ite((t_1 \neq 0) \wedge ((-t_1 \leq a \cdot x) \oplus (-t_1 \leq b \cdot x)), (b \cdot x > a \cdot x), (b \cdot x \leq a \cdot x))$$

Similarly, consider $\varphi_3 \equiv (b \cdot x + t_1 \leq b \cdot x + t_2)$. Using Proposition 6, with $b \cdot x + t_1$ in place of t_1, $b \cdot x + t_2$ in place of t_2 and $-b \cdot x$ in place of t_3,

$$\begin{aligned}\varphi_3 &\equiv ite((b \cdot x \neq 0) \wedge ((b \cdot x \leq b \cdot x + t_1) \oplus (b \cdot x \leq b \cdot x + t_2)), (t_1 > t_2), (t_1 \leq t_2)) \\ &\equiv ite((b \cdot x \neq 0) \wedge ((-b \cdot x \leq t_1) \oplus (-b \cdot x \leq t_2)), (t_1 > t_2), (t_1 \leq t_2))\end{aligned}$$

Putting everything together,

$$l_1 \equiv ite(\varphi_1 \wedge (\varphi_2 \oplus \varphi_3), (a \cdot x - b \cdot x > t_2 - t_1), (a \cdot x - b \cdot x \le t_2 - t_1)), \quad where$$
$$\varphi_1 \equiv (b \cdot x + t_1 \neq 0)$$
$$\varphi_2 \equiv ite((t_1 \neq 0) \wedge ((-t_1 \le a \cdot x) \oplus (-t_1 \le b \cdot x)), (b \cdot x > a \cdot x), (b \cdot x \le a \cdot x))$$
$$\varphi_3 \equiv ite((b \cdot x \neq 0) \wedge ((-b \cdot x \le t_1) \oplus (-b \cdot x \le t_2)), (t_1 > t_2), (t_1 \le t_2))$$

Hence l_1 can be equivalently expressed as, $\mathsf{ite}(\varphi, l_2, \neg l_2)$, where $l_2 \equiv (a \cdot x - b \cdot x \le t_2 - t_1)$, and $\varphi \equiv \neg\varphi_1 \vee (\varphi_2 \equiv \varphi_3)$. Note that φ here is a Boolean combination of LMCs normalized w.r.t. x. □

3.5.2 Modified FM for normalized LMIs

We begin by illustrating the primary idea through an example. Consider the problem of computing $\exists x.C$, where $C \equiv (y \le 4x) \wedge (4x \le z)$ with modulus 16. Note that $\exists x.C$ is "the condition under which there exists a multiple of 4 between y and z, where $y \le z$". Note that if x, y, z were reals, then we would have obtained $(y \le z)$ for $\exists x.C$. However, as in the case of integers, this would over-approximate $\exists x.C$ in the case of fixed width bit-vectors.

If $(y \le 12) \wedge (z \ge y + 3)$ holds, then the difference between y and z is ≥ 3. In this case, existence of a multiple of 4 between y and z is guaranteed. Thus $(y \le z) \wedge (y \le 12) \wedge (z \ge y + 3) \Rightarrow \exists x.C$.

It can be seen that if $(y > 12)$, then there does not exist any x such that $(y \le 4x)$. Hence, if $(y > 12)$, then $\exists x.C$ is false. If $(z < y + 3)$, then $\exists x.C$ is true iff one of the following conditions holds: (i) $(y \le z)$ and y is a multiple of 4, i.e., $(y \le z) \wedge (4y = 0)$, (ii) $(y \le z)$ and $(y > z \pmod 4)$, i.e., $(y \le z) \wedge (4y > 4z)$.

Hence $\exists x.C$ is equivalent to $(y \le z) \wedge \varphi$, where φ is the disjunction of the following three formulas: (i) $(z \ge y+3) \wedge (y \le 12)$, (ii) $(z < y+3) \wedge (4y = 0)$, (iii) $(z < y+3) \wedge (4y > 4z)$.

The following Lemma generalizes this idea.

Lemma 6 *Let $l_1 : (t_1 \le a \cdot x)$ and $l_2 : (a \cdot x \le t_2)$ be LMIs in NF1 w.r.t. x. Let k be $\kappa(x, a \cdot x)$. Then, $\exists x.(l_1 \wedge l_2) \equiv (t_1 \le t_2) \wedge \varphi$, where φ is the disjunction of the formulas: (i) $(2^{p-k} \cdot t_1 = 0)$, (ii) $(t_2 \ge t_1 + 2^k - 1) \wedge (t_1 \le 2^p - 2^k)$, and (iii) $(t_2 < t_1 + 2^k - 1) \wedge (2^{p-k} \cdot t_1 > 2^{p-k} \cdot t_2)$.*

Proof Note that $\exists x.(l_1 \wedge l_2) \equiv \exists x.(l_1' \wedge l_2')$, where $l_1' \equiv (t_1 \le 2^k \cdot x)$ and $l_2' \equiv (2^k \cdot x \le t_2)$, since the multiples of 2^k and $2^k \cdot e$ are the same modulo 2^p for any odd number $e \in \{1, \ldots, 2^p - 1\}$.

Now $\exists x.(l_1' \wedge l_2') \equiv \exists x.\psi_1 \vee \exists x.\psi_2 \vee \exists x.\psi_3 \vee \exists x.\psi_4$, where

- $\psi_1 \equiv l_1' \wedge l_2' \wedge (2^{p-k} \cdot t_1 = 0)$
- $\psi_2 \equiv l_1' \wedge l_2' \wedge (2^{p-k} \cdot t_1 \neq 0) \wedge (t_2 \ge t_1 + 2^k - 1) \wedge (t_1 \le 2^p - 2^k)$
- $\psi_3 \equiv l_1' \wedge l_2' \wedge (2^{p-k} \cdot t_1 \neq 0) \wedge (t_2 < t_1 + 2^k - 1)$
- $\psi_4 \equiv l_1' \wedge l_2' \wedge (2^{p-k} \cdot t_1 \neq 0) \wedge (t_1 > 2^p - 2^k)$

Consider $\exists x.\psi_1$. This is equivalent to $\exists x.(\psi_1 \wedge (t_1 \le t_2))$, since $(t_1 \le t_2)$ is an LMI implied by ψ_1. It can be seen that $\exists x.(\psi_1 \wedge (t_1 \le t_2))$ is equivalent to $(2^{p-k} \cdot t_1 = 0) \wedge (t_1 \le t_2)$, since given any solution to $(2^{p-k} \cdot t_1 = 0) \wedge (t_1 \le t_2)$, we can satisfy $l_1' \wedge l_2'$ by setting $2^k \cdot x$ to t_1. Note that setting $2^k \cdot x$ to t_1 is always possible, since $2^{p-k} \cdot t_1 = 0 \Rightarrow \exists x.(2^k \cdot x = t_1)$ (see Proposition 3). Hence, $\exists x.\psi_1 \equiv (2^{p-k} \cdot t_1 = 0) \wedge (t_1 \le t_2)$.

Consider $\exists x.\psi_2$. Note that the difference between t_1 and t_2 here is $\ge 2^k - 1$, which implies that there exists a multiple of 2^k between t_1 and t_2. Hence it can be seen that $(t_1 \le t_2) \wedge (2^{p-k} \cdot$

$t_1 \neq 0) \wedge (t_2 \geq t_1 + 2^k - 1) \wedge (t_1 \leq 2^p - 2^k) \Rightarrow \exists x.\psi_2$. Implication in the other direction is obvious. Hence, $\exists x.\psi_2 \equiv (t_1 \leq t_2) \wedge (2^{p-k} \cdot t_1 \neq 0) \wedge (t_2 \geq t_1 + 2^k - 1) \wedge (t_1 \leq 2^p - 2^k)$.

Consider $\exists x.\psi_3$. This implies $(2^{p-k} \cdot t_1 > 2^{p-k} \cdot t_2)$. Hence $\exists x.\psi_3 \equiv \exists x.(\psi_3 \wedge (2^{p-k} \cdot t_1 > 2^{p-k} \cdot t_2))$. This is equivalent to $(t_1 \leq t_2) \wedge (2^{p-k} \cdot t_1 \neq 0) \wedge (t_2 < t_1 + 2^k - 1) \wedge (2^{p-k} \cdot t_1 > 2^{p-k} \cdot t_2)$, as the existence of a multiple of 2^k between t_1 and t_2 is implied by $(t_1 \leq t_2) \wedge (2^{p-k} \cdot t_1 \neq 0) \wedge (t_2 < t_1 + 2^k - 1) \wedge (2^{p-k} \cdot t_1 > 2^{p-k} \cdot t_2)$.

Consider $\exists x.\psi_4$. This is equivalent to false, since given $(t_1 > 2^p - 2^k)$, there exists no t_2 such that $l'_1 \wedge l'_2$ holds.

Putting everything together, it can be seen that $\exists x.(l_1 \wedge l_2) \equiv (t_1 \leq t_2) \wedge \varphi$, 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 6, where k denotes $\kappa(x, a \cdot x)$. It is easy to see that Lemma 6 can be used to compute $\exists x.I_{1,i}$, 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 $I_{1,i}$ into two sets $\Lambda_{\leq}$ and $\Lambda_{\geq}$, where $\Lambda_{\bowtie} = \{l_{i,j} \mid l_{i,j}$ 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 $\exists x.I_{1,i}$ as $\bigwedge_{(a_i \cdot x \leq t_p) \in \Lambda_{\leq},\ (a_i \cdot x \geq t_q) \in \Lambda_{\geq}} \left(\Delta\left(t_q, t_p, \kappa\left(x, a_i \cdot x\right)\right)\right)$.

Each conjunction of LMIs such as $I_{1,i}$ above, where all LMIs are in *NF1* w.r.t. x, and have the same coefficient of x are said to be "coefficient-matched" w.r.t. x. Similarly, a Boolean combination of LMCs φ is said to be coefficient-matched 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. In the special case when $I_2 \equiv \mathsf{true}$ and $n = 1$, i.e., when I is a conjunction of LMIs coefficient-matched w.r.t. x, $\exists x.I$ reduces to $\exists x.I_{1,1}$.

Unfortunately, converting I to coefficient-matched form w.r.t. a variable is inefficient in general. Hence we propose converting I to coefficient-matched form w.r.t. x only in the following cases, where it can be done without much loss of efficiency: (a) $I_2 \equiv \mathsf{true}$, $n = 2$ and $a_2 = -a_1$, and (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 coefficient-matched w.r.t. x by using the following Proposition.

Proposition 7 $(-t_1 \leq -t_2)$ *is equivalent to* $(t_1 = 0) \vee ((t_2 \neq 0) \wedge (t_1 \geq t_2))$.

Example Consider the problem of computing $\exists x.I$, where $I \equiv (y \leq 2x) \wedge (6x \leq z)$ with modulus 8. Using Proposition 7, $(6x \leq z)$ is equivalent to $(2x = 0) \vee ((z \neq 0) \wedge (2x \geq -z))$. Thus $\exists x.I$ can be equivalently expressed as $\exists x.\varphi$, where φ is the disjunction of $(y \leq 2x) \wedge (2x = 0)$ and $(y \leq 2x) \wedge (z \neq 0) \wedge (2x \geq -z)$. Note that φ is coefficient-matched w.r.t. x.

We explain the idea behind case (b) with an example before considering the general case. 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) $\Theta(x, x) \wedge \Theta(z, z) \wedge (2x \leq 2z)$, (ii) $\neg\Theta(x, x) \wedge \neg\Theta(z, z) \wedge (2x \leq 2z)$, and (iii) $\neg\Theta(x, x) \wedge \Theta(z, z)$. Hence, $\exists x.I$ can be equivalently expressed as $\exists x.\varphi'$, where φ' is the disjunction of (i) $\Theta(x, x) \wedge \Theta(z, z) \wedge (2x \leq 2z) \wedge (y \leq 2x)$, (ii) $\neg\Theta(x, x) \wedge \neg\Theta(z, z) \wedge (2x \leq$

$2z) \wedge (y \leq 2x)$, and (iii) $\neg\Theta(x, x) \wedge \Theta(z, z) \wedge (y \leq 2x)$. Note that $\Theta(x, x)$ and $\Theta(z, z)$ 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 $\Theta(x, x)$ 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 coefficient-matched w.r.t. x.

In general, given $\exists x.I$ such that $I_2 \equiv \mathsf{true}$ and every a_i is of the form $2^{k_i} \cdot e$ (as defined above), we have the following Lemma.

Lemma 7 *Let I_1 be a conjunction of LMIs in* NF1 *w.r.t. x. Let $a_1, \ldots, a_n$ be the distinct non-zero coefficients of x in LMIs in I_1. Let each a_i, for $1 \leq i \leq n$, be of the form $2^{k_i} \cdot e$, where e is an odd number in $\{1, \ldots, 2^p - 1\}$ independent of i. Then, $\exists x.I_1$ can be equivalently expressed as $\exists x.\varphi$, where φ is a Boolean combination of LMCs coefficient-matched w.r.t. x.*

Proof Our proof makes use of the following claims.

Claim 3 *An LMI $a \cdot x \leq t$ in* NF1 *can be equivalently expressed as the disjunction of formulas: (i) $\Theta(a\cdot x, a\cdot x) \wedge \Theta(t, t) \wedge (2a\cdot x \leq 2t)$, (ii) $\neg\Theta(a\cdot x, a\cdot x) \wedge \neg\Theta(t, t) \wedge (2a\cdot x \leq 2t)$, and (iii) $\neg\Theta(a \cdot x, a \cdot x) \wedge \Theta(t, t)$.*

Claim 4 *An LMI $a \cdot x \geq t$ in* NF1 *can be equivalently expressed as the disjunction of formulas: (i) $\Theta(a\cdot x, a\cdot x) \wedge \Theta(t, t) \wedge (2a\cdot x \geq 2t)$, (ii) $\neg\Theta(a\cdot x, a\cdot x) \wedge \neg\Theta(t, t) \wedge (2a\cdot x \geq 2t)$, and (iii) $\Theta(a \cdot x, a \cdot x) \wedge \neg\Theta(t, t)$.*

To see why Claim 3 is true, note that $(a \cdot x \leq t) \equiv \psi_1 \vee \psi_2 \vee \psi_3 \vee \psi_4$, where

- $\psi_1 \equiv (a \cdot x \leq t) \wedge \Theta(a \cdot x, a \cdot x) \wedge \Theta(t, t)$
- $\psi_2 \equiv (a \cdot x \leq t) \wedge \Theta(a \cdot x, a \cdot x) \wedge \neg\Theta(t, t)$
- $\psi_3 \equiv (a \cdot x \leq t) \wedge \neg\Theta(a \cdot x, a \cdot x) \wedge \Theta(t, t)$
- $\psi_4 \equiv (a \cdot x \leq t) \wedge \neg\Theta(a \cdot x, a \cdot x) \wedge \neg\Theta(t, t)$

It can be seen that,

- $\psi_1 \equiv \Theta(a \cdot x, a \cdot x) \wedge \Theta(t, t) \wedge (2a \cdot x \leq 2t)$
- $\psi_2 \equiv \mathsf{false}$, since $\Theta(a \cdot x, a \cdot x) \wedge \neg\Theta(t, t) \Rightarrow (a \cdot x > t)$
- $\psi_3 \equiv \neg\Theta(a \cdot x, a \cdot x) \wedge \Theta(t, t)$, since $\neg\Theta(a \cdot x, a \cdot x) \wedge \Theta(t, t) \Rightarrow (a \cdot x < t)$
- $\psi_4 \equiv \neg\Theta(a \cdot x, a \cdot x) \wedge \neg\Theta(t, t) \wedge (2a \cdot x \leq 2t)$

Claim 4 can be proved in a similar manner.

Without loss of generality, let $a_1 > a_2 > \cdots > a_n$, i.e., $2^{k_1} \cdot e > 2^{k_2} \cdot e > \cdots > 2^{k_n} \cdot e$. This implies that (i) $k_1 > k_2 > \cdots > k_n$, and (ii) $a_1 = 2^{k_1 - k_i} \cdot a_i$ for $2 \leq i \leq n$.

Now consider each LMI $a_i \cdot x \bowtie t_j$ in I_1, where $2 \leq i \leq n$ and $\bowtie \in \{\leq, \geq\}$. It can be seen that the above Claims can be used to express $a_i \cdot x \bowtie t_j$ as an equivalent Boolean combination of LMCs, involving (i) the LMI $(2a_i \cdot x \bowtie 2t_j)$, (ii) $\Theta(a_i \cdot x, a_i \cdot x)$, and (iii) $\Theta(t_j, t_j)$. Moreover, the above claims can be used repeatedly to express $a_i \cdot x \bowtie t_j$ as an equivalent Boolean combination of LMCs, involving (i) the LMI $(2^{k_1 - k_i} a_i \cdot x \bowtie 2^{k_1 - k_i} t_j)$, i.e., $(a_1 \cdot x \bowtie 2^{k_1 - k_i} t_j)$, (ii) $\Theta(a_i \cdot x, a_i \cdot x)$, $\Theta(2a_i \cdot x, 2a_i \cdot x)$,..., $\Theta(2^{k_1 - k_i - 1} a_i \cdot x, 2^{k_1 - k_i - 1} a_i \cdot x)$, and (iii) $\Theta(t_j, t_j)$, $\Theta(2t_j, 2t_j)$,..., $\Theta(2^{k_1 - k_i - 1} t_j, 2^{k_1 - k_i - 1} t_j)$.

It can be seen that $\Theta(a_i \cdot x, a_i \cdot x)$, $\Theta(2a_i \cdot x, 2a_i \cdot x)$,..., $\Theta(2^{k_1 - k_i - 1} a_i \cdot x, 2^{k_1 - k_i - 1} a_i \cdot x)$ can be equivalently expressed as $(a_i \cdot x \geq 2^{p-1})$, $(2a_i \cdot x \geq 2^{p-1})$, ..., $(2^{k_1 - k_i - 1} a_i \cdot x \geq 2^{p-1})$ respectively. Similarly $\Theta(t_j, t_j)$, $\Theta(2t_j, 2t_j)$,..., $\Theta(2^{k_1 - k_i - 1} t_j, 2^{k_1 - k_i - 1} t_j)$ can be equivalently expressed as $(t_j \geq 2^{p-1})$, $(2t_j \geq 2^{p-1})$, ..., $(2^{k_1 - k_i - 1} t_j \geq 2^{p-1})$ respectively.

Hence I_1 can be equivalently expressed as a Boolean combination of LMCs φ', involving (i) LMIs of the form $(a_1 \cdot x \bowtie 2^{k_1-k_i} \cdot t_j)$, (ii) LMIs of the form $(a_i \cdot x \geq 2^{p-1})$, $(2a_i \cdot x \geq 2^{p-1})$, …, $(2^{k_1-k_i-1}a_i \cdot x \geq 2^{p-1})$, and (iii) LMIs of the form $(t_j \geq 2^{p-1})$, $(2t_j \geq 2^{p-1})$, …, $(2^{k_1-k_i-1}t_j \geq 2^{p-1})$.

We can express φ' equivalently as $\bigvee_{\ell=1}^{r} C_\ell$, where each C_ℓ is a conjunction of LMCs. Hence $\exists x.\varphi'$ is equivalent to $\bigvee_{\ell=1}^{r}(\exists x.C_\ell)$. Observe that each C_ℓ involves three kinds of LMIs: (i) LMIs of the form $(a_1 \cdot x \bowtie 2^{k_1-k_i} \cdot t_j)$, (ii) LMIs of the form $(a_i \cdot x \geq 2^{p-1})$, $(2a_i \cdot x \geq 2^{p-1})$, …, $(2^{k_1-k_i-1}a_i \cdot x \geq 2^{p-1})$ and/or their negations, and (iii) LMIs of the form $(t_j \geq 2^{p-1})$, $(2t_j \geq 2^{p-1})$, …, $(2^{k_1-k_i-1}t_j \geq 2^{p-1})$ and/or their negations. Let $C_{\ell,1}$ be the conjunction of the first kind of LMIs in C_ℓ. Similarly, let $C_{\ell,2}$ and $C_{\ell,3}$ respectively be the conjunctions of the second and the third kinds of LMIs in C_ℓ. Hence we have $C_\ell \equiv C_{\ell,1} \wedge C_{\ell,2} \wedge C_{\ell,3}$.

Therefore $\exists x.C_\ell \equiv (\exists x.(C_{\ell,1} \wedge C_{\ell,2})) \wedge C_{\ell,3}$, since $C_{\ell,3}$ is free of x. Moreover, by applying Theorem 1 on $\exists x.(C_{\ell,1} \wedge C_{\ell,2})$, it can be proved that $C_{\ell,2}$ is unconstraining in $\exists x.(C_{\ell,1} \wedge C_{\ell,2})$. Hence $\exists x.C_\ell$ can be equivalently expressed as $\exists x.(C_{\ell,1}) \wedge C_{\ell,3}$. Note that the coefficient of x in $C_{\ell,1}$ is a_1. This implies that $\bigvee_{\ell=1}^{r} C_\ell$ can be equivalently expressed as a Boolean combination of LMCs coefficient-matched w.r.t. x, with coefficient of x as a_1.

□

Note that normalizing a given conjunction of LMIs w.r.t. a variable and then converting it to coefficient-matched form transforms it to a Boolean combination of LMCs in general. We make use of techniques in Sect. 4 for eliminating quantifiers from such Boolean combinations of LMCs.

In cases other than those covered in cases (a) and (b) above, we propose computing $\exists x.I$ using *model enumeration*, i.e., by expressing $\exists x.I$ in the equivalent form $I|_{x\leftarrow 0} \vee \cdots \vee I|_{x\leftarrow 2^p-1}$ 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* (see Algorithm 1). Initially, the LMDs and the single LME in the conjunction are converted to LMIs using the equivalences $(t_1 = t_2) \equiv (t_1 \geq t_2) \wedge (t_1 \leq t_2)$ and $(t_1 \neq t_2) \equiv \neg(t_1 = t_2)$. This in general yields a Boolean combination of LMCs φ_1. If φ_1 is a conjunction of LMIs coefficient-matched w.r.t. x, then $\exists x.\varphi_1$ is computed using the modified FM elimination in Lemma 6. Otherwise, $\exists x.\varphi_1$ is computed either by converting φ_1 to coefficient-matched form w.r.t. x, followed by QE from the resulting Boolean combination of LMCs, or by model enumeration.

Analysis of complexity Consider a conjunction of LMCs with a subset of variables in its support to be eliminated. Let n be the number of LMCs in the conjunction, v be the number of variables in its support, and e be the number of variables to be eliminated. Note that Layer3 resorts to model enumeration in the worst case. Consider the elimination of the first quantified variable, say, x_1 by model enumeration.

Elimination of x_1 by model enumeration involves creating 2^p copies of the conjunction, and then replacing x_1 by a constant in each copy. Replacing x_1 by constant and then simplifying takes $O(n)$ arithmetic operations in the worst-case for each copy. Assuming that each arithmetic operation on p-bit numbers take time $O(Q(p))$ in the worst-case, where $p \leq Q(p) \leq p^3$, elimination of x_1 from each copy hence has a worst-case time complexity of $O(n \cdot Q(p))$. Since there are 2^p such copies, elimination of x_1 has a worst-case time complexity of $O(n \cdot Q(p) \cdot 2^p)$.

Elimination of x_1 generates a formula with 2^p disjuncts, where each disjunct can have n LMCs. In a similar manner as above, it can be seen that elimination of the second quantified variable, say, x_2 has a worst-case time complexity of $O(n \cdot Q(p) \cdot 2^{2\cdot p})$. Proceeding like this,

Algorithm 1: *QE1_Layer3*

```
Input: Conjunction of LMCs $C$, Variable to eliminate $x$
Output: Boolean combination of LMCs $\psi$ equivalent to $\exists x. C$
$\varphi_1$ := convertToLMIs($C$);                  // convert LMEs and LMDs to LMIs
if $\varphi_1$ is a coefficient-matched conjunction w.r.t. $x$ then
    $\psi$ := modifiedFM($\varphi_1, x$);            // Apply modified FM based on Lemma 6
else
    if model enumeration is selected to compute $\exists x. \varphi_1$ then
        $\psi$ := modelEnumerate($\varphi_1, x$);    // Apply model enumeration
    else
        $\varphi_2$ := coefficientMatch($\varphi_1, x$);   // $\varphi_1$ in general is a Boolean combination
        $\psi$ := QEFromBooleanCombination($\varphi_2, x$);
        // Eliminate $x$ from Boolean combination $\varphi_2$; this recursively
        // calls Project (see Section 4 for details)
return $\psi$;
```

it can be seen that elimination of e quantified variables has a worst-case time complexity of $O\left(n \cdot Q(p) \cdot \left(2^p + 2^{2 \cdot p} + \cdots 2^{e \cdot p}\right)\right)$, which reduces to $O(n \cdot Q(p) \cdot 2^{(e+1) \cdot p})$.

After elimination of e variables, we have a formula with $2^{e \cdot p}$ disjuncts, where each disjunct can have n LMCs. Writing each disjunct involving n LMCs takes $O(n \cdot v \cdot p)$ time. Hence writing the result takes $O(n \cdot v \cdot p \cdot 2^{e \cdot p})$ time. Therefore Layer3 has a worst-case time complexity of $O(n \cdot Q(p) \cdot 2^{(e+1) \cdot p} + n \cdot v \cdot p \cdot 2^{e \cdot p})$.

3.6 *Project*: combining layers

Recall that *QE1_Layer1*, *QE1_Layer2*, and *QE1_Layer3* try to eliminate a single quantifier from a conjunction of LMCs. These procedures can be 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.

Algorithm 2: *Project*

```
Input: Conjunction of LMCs $A$, Set of variables to eliminate $X$
Output: Boolean combination of LMCs $\psi$ equivalent to $\exists X. A$
$\varphi_1$ := Layer1($A, X$);                      // for each $x \in X$, Apply QE1_Layer1
if $\varphi_1$ has no quantifiers then
    $\psi$ := $\varphi_1$;
else
    // Let $\varphi_1 \equiv A_1 \wedge \exists Y. B$
    $\varphi_2$ := Layer2($B, Y$);                   // for each $x \in Y$, Apply QE1_Layer2
    if $\varphi_2$ has no quantifiers then
        $\psi$ := $A_1 \wedge \varphi_2$;
    else
        // Let $\varphi_2 \equiv A_2 \wedge \exists Z. C$
        $\varphi_3$ := Layer3($C, Z$);               // for each $x \in Z$, Apply QE1_Layer3
        $\psi$ := $A_1 \wedge A_2 \wedge \varphi_3$;
return $\psi$;
```

We now present the overall QE algorithm *Project* (see Algorithm 2) 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 LMCs φ_1. If all variables in X are eliminated by *Layer1*, then φ_1 is free of quantifiers. In this case, $\exists X.A$ is equivalent to φ_1, and *Project* returns φ_1. Otherwise, φ_1 is equivalent to the conjunction of A_1 and $\exists Y.B$, where A_1, B are conjunctions of LMCs, $Y \subseteq X$, and $X \setminus Y$ is the subset of variables in X that are eliminated by *Layer1*. *Project* then tries to compute $\exists Y.B$ using *Layer2*.

Layer2 reduces $\exists Y.B$ to an equivalent conjunction of LMCs φ_2. If all variables in Y are eliminated by *Layer2*, then φ_2 is free of quantifiers. In this case $\exists X.A$ is equivalent to $A_1 \wedge \varphi_2$, and *Project* returns $A_1 \wedge \varphi_2$. Otherwise, φ_2 is equivalent to the conjunction of A_2 and $\exists Z.C$, where A_2, C are conjunctions of LMCs, $Z \subseteq Y$, and $Y \setminus Z$ is the subset of variables in Y that are eliminated by *Layer2*. *Project* calls *Layer3* to compute $\exists Z.C$. *Layer3* computes φ_3, a Boolean combination of LMCs equivalent to $\exists Z.C$, and *Project* returns $A_1 \wedge A_2 \wedge \varphi_3$.

Let x be the variable being eliminated. Line-8 of *QE1_Layer3* generates a Boolean combination of LMCs φ_2 coefficient-matched w.r.t. x. Line-9 of *QE1_Layer3* then calls *QEFromBooleanCombination* in order to eliminate x from φ_2. This eventually gets reduced to eliminating x from a bunch of conjunctions of LMCs. Eliminating x from each such conjunction of LMCs results in a new recursive *Project* call. Because of this feedback, the control flow inside *Project* is not linear.

Note that each new recursive *Project* call may in turn call *QE1_Layer3*. However it can be observed that this mutual recursion between *QE1_Layer3* and *Project* does not result in infinite recursion. To see this, note that in each of the recursive *Project* calls, all LMIs involving x are coefficient-matched w.r.t. x. Hence x will be certainly eliminated by *Layer1*, *Layer2*, or *modifiedFM* inside these recursive *Project* calls. This guarantees that the recursion terminates.

4 Extending QE to Boolean combinations

Algorithm *Project* described above eliminates a set of variables from a conjunction of LMCs. In this section, we explore approaches for extending *Project* to Boolean combinations of LMCs.

As mentioned in Sect. 2, the problem of extending a QE algorithm for conjunctions of constraints to Boolean combinations of constraints is encountered in other theories such as linear real arithmetic and linear integer arithmetic as well. Among the techniques to solve this problem for these theories (see Sect. 2.1), the work by Chaki et. al. in [12] proposes a decision diagram based algorithm for QE from formulas in the theory of Octagons and the work by Monniaux in [44] proposes an SMT solving based algorithm for extending Fourier–Motzkin to arbitrary Boolean combinations of constraints in linear real arithmetic. The approaches for extending *Project* to Boolean combinations of LMCs described in this section are motivated by the ideas introduced in these works.

4.1 Decision diagram based approach

We introduce a data structure called Linear Modular Decision Diagram (LMDD) that represents Boolean combinations of LMCs. LMDDs are BDDs [10] with nodes labeled with LMEs or LMIs. The problem we wish to solve in this subsection can be formally stated as follows. Given an LMDD f representing a Boolean combination of LMCs over a set of variables V, we wish to compute an LMDD g equivalent to $\exists X.f$, where $X \subseteq V$.

The algorithms presented in this subsection use the following helper functions: a) *Vars*: returns the set of variables in an LMC, b) *getConjunct*: computes the conjunction of LMCs in a given set, c) *isUnsat*: determines if the conjunction of LMCs in a given set is unsatisfiable, d) *createLMDD*: creates an LMDD from a Boolean combination of LMCs, e) *AND*, *OR*, *NOT*, *ITE*: performs the basic operations on LMDDs indicated by their names. We denote a non-terminal LMDD node f as $(P(f), H(f), L(f))$, where $P(f)$ is the LME/LMI labeling the node, and $H(f)$ and $L(f)$ are the high child and low child respectively as defined in [10].

A straightforward algorithm to compute $\exists X.f$ is to apply *Project* to each path originating from the root of f. We call this algorithm *All_Path_QElim* (see Algorithm 3). To compute $\exists X.f$, we call *All_Path_QElim* with arguments f, $\{\}$ and X. *All_Path_QElim* performs a recursive traversal of f collecting the set of LMCs S containing any of the variables in X that it encountered along the path from the root of f. If the path leads to a 1-terminal and if the conjunction C_s of LMCs in S is theory-consistent, then *Project* is called to compute $\exists X.C_s$.

Algorithm 3: *All_Path_QElim*

```
Input: LMDD $f$, Set of LMCs $S$, Set of variables to eliminate $X$
Output: LMDD for $\exists X.(f \wedge C_s)$, where $C_s$ is the conjunction of LMCs in $S$
if $f = 0$ or isUnsat($S$) then
    return 0;
if $f = 1$ then                                    // $f$ is theory-consistent 1-terminal
    $C_s$ := getConjunct($S$);
    $\pi$ := Project($C_s$, $X$);// $\pi \equiv \exists X.C_s$
    return createLMDD($\pi$);// $\pi \equiv \exists X.(f \wedge C_s)$
// traverse down
$c := P(f)$;
if Vars($c$) $\cap X$ == {} then                  // $c$ is free of variables to eliminate
    return ITE($c$, All_Path_QElim($H(f)$, $S$, $X$), All_Path_QElim($L(f)$, $S$, $X$));
else                                               // $c$ contains variables to eliminate
    return OR(All_Path_QElim($H(f)$, $S \cup \{c\}$, $X$), All_Path_QElim($L(f)$, $S \cup \{\neg c\}$, $X$));
```

As observed in [11,12], because of the dependence of the result of a recursive call on the context S, if the same LMDD node is encountered following two different paths, then the results of the calls are not the same in general. Hence *All_Path_QElim* is not amenable to dynamic programming, and the number of recursive calls is linear in the number of paths in f, which can be exponential in the number of nodes in f.

In the following discussion we present a more efficient algorithm *QE_LMDD* to compute $\exists X.f$. *QE_LMDD* makes use of an algorithm called *QE1_LMDD* that eliminates a single variable x from f (see Algorithm 4). To compute $\exists x.f$, we call *QE1_LMDD* with arguments f, $\{\}$ and x. *QE1_LMDD* performs a recursive traversal of the LMDD f collecting the set of LMCs S_x containing x that it encountered along the path from f.

In general, *QE1_LMDD* (f, S_x, x) computes an LMDD for $\exists x.(f \wedge C_{S_x})$, where C_{S_x} denotes the conjunction of LMCs in S_x. Let E_x be the set of LMEs in S_x. Let each LME e_i in E_x be of the form $2^{k_i} \cdot x = t_i$, where $k_i = \kappa(x, e_i)$ and $1 \leq i \leq n$ (recall the definition of κ from Sect. 3.1). Without loss of generality, let k_1 be the minimum of $k_1, \ldots, k_n$. Let g be any internal non-terminal node of f represented as $(P(g), H(g), L(g))$. Let us denote $P(g)$ by c. It can be observed that if c has x in its support, then c can be simplified by replacing the occurrences of $2^{k_1} \cdot x$ in it by t_1. Let c' be the simplified LMC. Note that if $\kappa(x, c) \geq k_1$, then

Algorithm 4: *QE1_LMDD*

```
Input: LMDD $f$, Set of LMCs $S_x$, Variable to eliminate $x$
Output: LMDD for $\exists x.\,(f \wedge C_{S_x})$, where $C_{S_x}$ is the conjunction of LMCs in $S_x$
if $f = 0$ or isUnsat($S_x$) then
    return 0;
if $f = 1$ then                                   // theory-consistent 1-terminal
    $C_{S_x}$ := getConjunct($S_x$);
    $\pi$ := Project($C_{S_x}$, $\{x\}$);// $\pi \equiv \exists x.\, C_{S_x}$
    return createLMDD($\pi$);// $\pi \equiv \exists x.\,(f \wedge C_{S_x})$
// simplification using LMEs
$E_x$ := set of LMEs in $S_x$;
if $E_x \neq \{\}$ then
    $e_1$ := selectLME($E_x$);
    $f'$ := simplifyLMDD($f$, $e_1$, $x$);
    if $f'$ is free of $x$ then
        $C_{S_x}$ := getConjunct($S_x$);
        $\pi$ := Project($C_{S_x}$, $\{x\}$);// $\pi \equiv \exists x.\, C_{S_x}$
        return AND($f'$, createLMDD($\pi$));// $f' \wedge \pi \equiv \exists x.\,(f \wedge C_{S_x})$
else
    $f'$ := $f$;
// traverse down
$c$ := $P(f')$;
if $c$ is free of $x$ then
    return ITE($c$, QE1_LMDD($H(f')$, $S_x$, $x$), QE1_LMDD($L(f')$, $S_x$, $x$));
else
    return OR(QE1_LMDD($H(f')$, $S_x \cup \{c\}$, $x$), QE1_LMDD($L(f')$, $S \cup \{\neg c\}$, $x$));
```

c' we get, is free of x. Thus, if $\kappa(x, c) \geq k_1$, then g can be simplified to $(c', H(g), L(g))$, where c' is free of x.

We call the procedure that performs the selection of LME with the minimum κ among the LMEs in E_x as *selectLME*. The Procedure *simplifyLMDD* (see Algorithm 5) performs simplification of f using the selected LME as described above. The procedure *simplifyLMC* in Algorithm 5 simplifies c to c' using the selected LME.

Algorithm 5: *simplifyLMDD*

```
Input: LMDD $f$, LME $e_1 : 2^{k_1} \cdot x = t_1$, Variable to eliminate $x$
Output: LMDD $f$ simplified using $e_1$
if $f = 0$ or $f = 1$ then
    return $f$;
$c$ := $P(f)$;
if $c$ is free of $x$ then
    return ITE($c$, simplifyLMDD($H(f)$, $x$, $e_1$), simplifyLMDD($L(f)$, $x$, $e_1$));
else
    $c'$ := simplifyLMC($c$, $e_1$, $x$);// if $\kappa(x, c) \geq k_1$, then $c'$ is free of $x$
    return ITE($c'$, simplifyLMDD($H(f)$, $x$, $e_1$), simplifyLMDD($L(f)$, $x$, $e_1$));
```

If *simplifyLMDD* is successful in eliminating all occurrences of variable x using the selected LME, then it returns a simplified LMDD f' such that $\exists x.(f \wedge C_{S_x})$ is equivalent to $f' \wedge \exists x.(C_{S_x})$. Note that $\exists x.(C_{S_x})$ can be computed by *Project*. In this case, *QE1_LMDD*

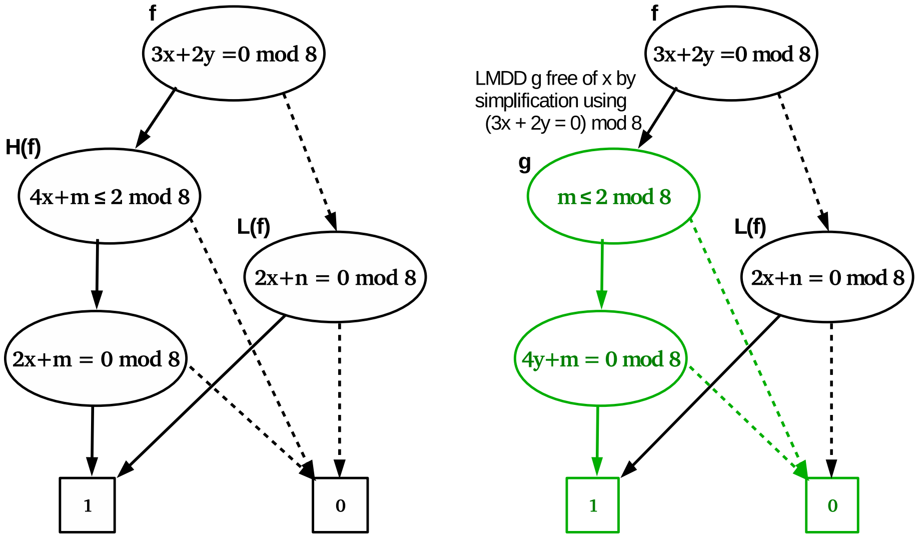


Fig. 3 Example for *QE1_LMDD*

returns without any further recursive calls. If *simplifyLMDD* is unable to eliminate all occurrences of variable x, then *QE1_LMDD* proceeds by recursively traversing the simplified LMDD f'.

Example All LMCs in this example have modulus 8. Let f be the LMDD shown on the left in Fig. 3. Suppose we wish to compute $\exists x.f$ using *QE1_LMDD*. Note that *QE1_LMDD* calls *simplifyLMDD* with arguments $H(f)$, $(3x + 2y = 0)$ and x. The LME $(3x + 2y = 0)$ is equivalent to $(x = 2y)$. *simplifyLMDD* eliminates all occurrences of x in $H(f)$ using $(x = 2y)$, and thus simplifies $H(f)$ as shown on the right in Fig. 3. Let g be the simplified LMDD, which is free of x (shown in different colour in Fig. 3). Notice that $\exists x.(H(f) \wedge (x = 2y))$ is equivalent to $g \wedge \exists x.(x = 2y)$. Since $\exists x.(x = 2y)$ is true, $\exists x.(H(f) \wedge (x = 2y))$ is equivalent to g. However, $L(f)$ cannot be simplified in this manner, as there are no LMEs involving x in its context. *QE1_LMDD* performs traversal of $L(f)$, and calls *Project* to compute $\exists x.((3x + 2y \neq 0) \wedge (2x + n = 0))$. *Project* computes $\exists x.((3x + 2y \neq 0) \wedge (2x + n = 0))$ as $(4n = 0)$. Hence the final result is LMDD for $g \vee (4n = 0)$.

It can be observed that if the same LMDD node is encountered with the same LME following two different paths, then the results of the calls to *simplifyLMDD* must be the same. Hence *simplifyLMDD* can be implemented with dynamic programming. Moreover, although the result of each recursive call to *QE1_LMDD* depends on the context S_x, the number of LMCs in S_x is usually very small, as only the LMCs containing x are collected in S_x. Hence *QE1_LMDD* is still amenable to dynamic programming.

QE1_LMDD can be repeatedly invoked to compute $\exists X.f$. This is implemented in the algorithm *QE_LMDD*. The order in which variables are selected for elimination in *QE_LMDD* has a crucial impact on the sizes of the intermediate and final LMDDs. In our ordering scheme, we selected the variable occurring in the least number of LMDD nodes as the next variable to be eliminated. Intuitively, this ordering scheme usually results in smaller contexts (i.e., smaller S_x's), and more opportunities for dynamic programming.

In practice, the strategy of eliminating one variable at a time and simplification of LMDDs using the LMEs in the context provide significant opportunities for reuse of results through dynamic programming. As a result of these, *QE_LMDD* in practice clearly outperforms *All_Path_QElim*, as also demonstrated by our experiments.

4.2 SMT solving based approach

In this subsection, we present an algorithm *QE_SMT* (see Algorithm 6) which is an extension of the algorithm proposed in [44]. Given a Boolean combination of LMCs φ over a set of variables V, *QE_SMT* computes a Boolean combination of LMCs ψ equivalent to $\exists X.\varphi$, where $X \subseteq V$. Notice that *QE_SMT* involves All-SMT loop with optimizations as suggested in [44].

Algorithm 6: *QE_SMT*

```
Input: Boolean combination of LMCs $\varphi$, Set of variables to eliminate $X$
Output: Boolean combination of LMCs $\psi$ equivalent to $\exists X.\,\varphi$
$H := \varphi$;
$\psi :=$ false;
while $H$ is satisfiable do
    $m :=$ a solution of $H$;// $m \models H$ and $m \models \varphi$
    $C := Generalize1(\varphi, m)$;// $C \Rightarrow \varphi$
    $C' := Generalize2(\varphi, C)$;// $C \Rightarrow C'$ and $C' \Rightarrow \varphi$
    $\pi := Project(C', X)$;// $\pi \equiv \exists X.\,C'$
    $\psi := \psi \vee \pi$;
    $H := H \wedge \neg\pi$;
return $\psi$;// $\psi \equiv \exists X.\,\varphi$
```

Algorithm 7: *Generalize1*

```
Input: Boolean combination of LMCs $\varphi$, A solution $m$ of $\varphi$
Output: A conjunction $C$ of LMCs such that $C \Rightarrow \varphi$
$S :=$ set of LMCs in $\varphi$;
$C :=$ true;
for $c \in S$ do
    if $m \models c$ then
        $C := C \wedge c$;
    else
        $C := C \wedge \neg c$;
return $C$;
```

Each iteration of the All-SMT loop in Algorithm 6 finds a solution m of H. Note that m is also a solution of φ. *Generalize1* (see Algorithm 7, originally proposed in [44]) is then used for generalizing m to a conjunction of LMCs C such that $C \Rightarrow \varphi$. *Generalize1* computes C as follows. First C is initialized to true. Each LMC c in φ is then evaluated with values given to variables in its support as per m. If c evaluates to true under m, i.e., $m \models c$, then c is conjoined with C. Otherwise, if c evaluates to false under m, i.e., $m \models \neg c$, then $\neg c$ is conjoined with C. It is easy to see that the conjunction C returned implies φ.

Generalize2 is used for further generalizing C by dropping unnecessary constraints from C. Hence C' computed by *Generalize2* is such that $C \Rightarrow C'$ and $C' \Rightarrow \varphi$. The implementation of *Generalize2* in [44] works as follows. For each constraint c in C, it is checked to see if

$C \Rightarrow \varphi$ remains valid even after dropping c from C. If $C \Rightarrow \varphi$ remains valid even after dropping c from C, then c is unnecessary and is dropped from C. Otherwise if the implication $C \Rightarrow \varphi$ becomes invalid after dropping c from C, then c is not dropped from C. Checking the validity of $C \Rightarrow \varphi$ involves an SMT solver call. However, in our experiments with LMCs, we have found that this implementation of *Generalize2* is prohibitively time consuming as the number of SMT solver calls is equal to the number of constraints in C. Our implementation of *Generalize2* makes use of a cheaper technique to achieve generalization.

The technique is based on analysis of the Boolean skeleton of the formula φ. Boolean skeleton P of φ is the representation of Boolean structure of φ as a Directed Acyclic Graph (DAG), with leaves representing LMCs in φ and internal nodes as $\neg$, $\wedge$, and $\vee$. As every LMC in φ appears in C in its original or negated form, C effectively gives an assignment of Boolean values to the leaves of P. We now perform a bottom-up traversal of P to evaluate P using the values assigned to the leaves. Let $B(n)$ be the value assigned to a node n in P during the evaluation. For each node n, we find a subset $S(n)$ of LMCs in C that are sufficient to evaluate n to $B(n)$. Table 1 shows how $B(n)$ and $S(n)$ are computed for the different nodes in P under different conditions. Let $S(r)$ be the set of LMCs found in this way for the root r of P. Then C' is computed as the conjunction of LMCs in $S(r)$. It is easy to see that $C \Rightarrow C'$ and $C' \Rightarrow \varphi$.

Example All LMCs in this example have modulus 8. Let φ be $(y = 4x) \wedge ((x \neq z) \vee (x \neq w))$. Suppose we wish to compute $\exists X.\varphi$ using *QE_SMT*, where $X = \{x\}$. Let $m : x = 1$, $y = 4$, $z = 1$, $w = 0$ be the solution of φ from SMT solver in the first iteration of the loop in *QE_SMT*. Note that *Generalize1* generalizes m to the conjunction $C : (y = 4x) \wedge (x = z) \wedge (x \neq w)$. *Generalize2* then generalizes C to $C' : (y = 4x) \wedge (x \neq w)$. To see how *Generalize2* works, observe that the Boolean skeleton P of φ is $n_1 \wedge (n_2 \vee n_3)$, where n_1, n_2, n_3 denote $(y = 4x)$, $(x \neq z)$, $(x \neq w)$ respectively. From Table 1, we have, $B(n_1) = \mathsf{true}$, $B(n_2) = \mathsf{false}$, and $B(n_3) = \mathsf{true}$. Also $S(n_1) = \{n_1\}$, $S(n_2) = \{\neg n_2\}$, and $S(n_3) = \{n_3\}$. Let n_4 be the node $(n_2 \vee n_3)$. Since $B(n_2) = \mathsf{false}$, $B(n_3) = \mathsf{true}$, and n_4 is $(n_2 \vee n_3)$, we have $B(n_4) = \mathsf{true}$. Note that $B(n_3) = \mathsf{true}$ is sufficient to make $B(n_4) = \mathsf{true}$. We have $S(n_4) = S(n_3) = \{n_3\}$ as per Table 1. Let r be the root node of P, i.e., the node $n_1 \wedge n_4$. Since $B(n_1) = \mathsf{true}$, $B(n_4) = \mathsf{true}$, we have $B(r) = \mathsf{true}$. Since r is $n_1 \wedge n_4$, both $B(n_1)$ and

Table 1 Computation of $B(n)$ and $S(n)$ inside *Generalize2*

node n	Condition	$B(n)$	$S(n)$
LMC c	c appears in C	$true$	$\{c\}$
	$\neg c$ appears in C	$false$	$\{\neg c\}$
$\neg n_1$	$B(n_1) = true$	$false$	$S(n_1)$
	$B(n_1) = false$	$true$	$S(n_1)$
$n_1 \wedge n_2$	$B(n_1) = true \wedge B(n_2) = true$	$true$	$S(n_1) \cup S(n_2)$
	$B(n_1) = true \wedge B(n_2) = false$	$false$	$S(n_2)$
	$B(n_1) = false \wedge B(n_2) = true$	$false$	$S(n_1)$
	$B(n_1) = false \wedge B(n_2) = false$	$false$	Smaller among $S(n_1)$ and $S(n_2)$
$n_1 \vee n_2$	$B(n_1) = true \wedge B(n_2) = true$	$true$	Smaller among $S(n_1)$ and $S(n_2)$
	$B(n_1) = true \wedge B(n_2) = false$	$true$	$S(n_1)$
	$B(n_1) = false \wedge B(n_2) = true$	$true$	$S(n_2)$
	$B(n_1) = false \wedge B(n_2) = false$	$false$	$S(n_1) \cup S(n_2)$

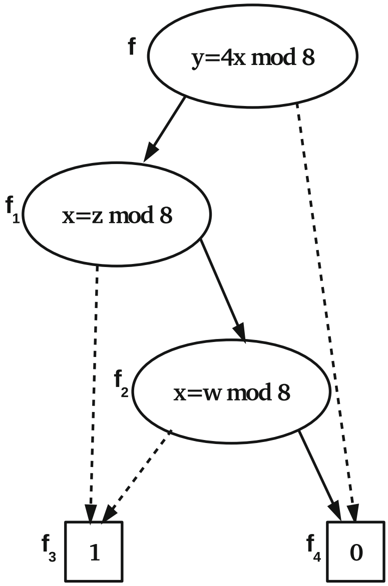


Fig. 4 Example for hybrid approach

$B(n_4)$ should be true for $B(r)$ to be true. We have $S(r) = S(n_1) \cup S(n_4) = \{n_1, n_3\}$. Finally C' is $n_1 \wedge n_3$, i.e., $(y = 4x) \wedge (x \neq w)$. *Project* computes $\exists x.C'$ as $\pi : (2y = 0)$. Note that $\varphi \wedge \neg\pi$ is unsatisfiable, and the algorithm terminates. The result of QE is thus $(2y = 0)$.

4.3 Hybrid approach

The factors that contribute to the success of the LMDD 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 solving 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.

We illustrate the idea with the help of an example. All LMCs in this example have modulus 8. Let f be the LMDD shown in Fig. 4. Let f_1, f_2, f_3, and f_4 be the internal nodes of the LMDD as shown in Fig. 4. Suppose we wish to compute $\exists x.f$. Note that $\exists x.f$ is the disjunction of three sub-problems: (i) $\exists x.(f_3 \wedge (y = 4x) \wedge (x \neq z))$, (ii) $\exists x.(f_2 \wedge (y = 4x) \wedge (x = z))$, and (iii) $\exists x.(f_4 \wedge (y \neq 4x))$. Also, notice that $\exists x.f$ is actually equivalent to $(2y = 0)$, the result of the first sub-problem $\exists x.(f_3 \wedge (y = 4x) \wedge (x \neq z))$. Hence it is not necessary to compute the sub-problems $\exists x.(f_2 \wedge (y = 4x) \wedge (x = z))$ and $\exists x.(f_4 \wedge (y \neq 4x))$. We call such sub-problems whose computation is not necessary as “redundant” sub-problems. We can infer that the sub-problems $\exists x.(f_2 \wedge (y = 4x) \wedge (x = z))$ and $\exists x.(f_4 \wedge (y \neq 4x))$ are redundant, from the fact that $f_2 \wedge (y = 4x) \wedge (x = z) \wedge (2y \neq 0)$ and $f_4 \wedge (y \neq 4x) \wedge (2y \neq 0)$ are unsatisfiable.

In general, suppose we wish to compute $\exists X.f$, where f denotes an LMDD representing a Boolean combination of LMCs over a set of variables V and $X \subseteq V$. We can derive a set of sub-problems of the form $\exists X.(f_i \wedge C_i)$, for $1 \leq i \leq n$, where f_i denotes an LMDD and C_i denotes a conjunction of LMCs, such that $\exists X.f$ is equivalent to $\bigvee_{i=1}^{n} (\exists X.(f_i \wedge C_i))$.

Let g denote $\bigvee_{i=1}^{m}(\exists X.(f_i \wedge C_i))$, where $1 \leq m < n$. A sub-problem $\exists X.(f_j \wedge C_j)$, where $m+1 \leq j \leq n$, is redundant if $f_j \wedge C_j \wedge \neg g$ is unsatisfiable.

Our hybrid algorithm *QE_Combined* (see Algorithm 8) makes use of this idea to identify redundant sub-problems. Initially, *QE_Combined* selects a satisfiable path π in the LMDD f using a function *selectPath*. Subsequently, the algorithm *simplify* (see Algorithm 9) is invoked, which traverses the path π, in order to 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. In Algorithm 9, $(f_i \wedge C_i)$ is represented as a pair $\langle f_i, C_i \rangle$.

Algorithm 8: *QE_Combined*

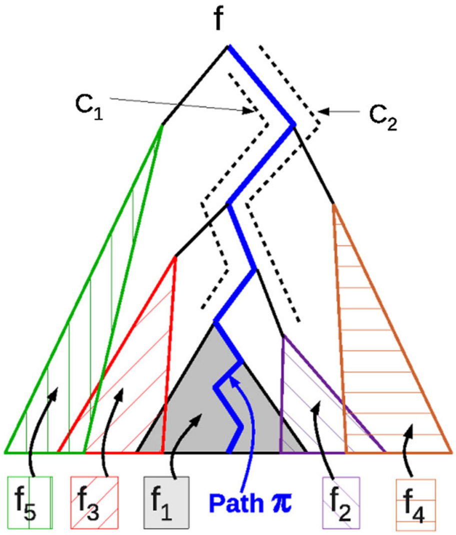
```
Input: LMDD $f$, Set of variables to eliminate $X$
Output: Boolean combination of LMCs $g$ equivalent to $\exists X.\ f$
$\pi$ := selectPath($f$);
$S$ := {};// set of sub-problems
$C$ := true;
simplify($f, \pi, C, S$);
$g$ := false;
for each $\langle f_i, C_i \rangle \in S$ do
    if $f_i \wedge C_i \wedge \neg g$ is satisfiable then
        $h$ := QE_LMDD_Mod($f_i, C_i, X$);
        $g := g \vee h$;
return $g$;
```

In order to split LMDD f, *simplify* is called with arguments f, π, C and S. Note that C is initialized to true and S initialized to $\{\}$. *simplify* collects $(f_i \wedge C_i)$, for $1 \leq i \leq n$ in the set S in the following way. The path π is traversed recursively starting from the root node of f, conjoining with C all LMCs encountered on π. In each recursive call, if f is a terminal, then $\langle f, C \rangle$ is inserted in S. Otherwise if f is a non-terminal and node $H(f)$ appears in π, then $\langle L(f), C \wedge \neg P(f) \rangle$ is inserted in S. Similarly if f is a non-terminal and node $L(f)$ appears in π, then $\langle H(f), C \wedge P(f) \rangle$ is inserted in S. Figure 5 illustrates the splitting scheme followed by *simplify*. For example, in the case of LMDD in Fig. 4, using the path $(y = 4x) \rightarrow (x \neq z) \rightarrow 1$ as π, splits the LMDD into (i) $\langle f_3, (y = 4x) \wedge (x \neq z) \rangle$, (ii) $\langle f_2, (y = 4x) \wedge (x = z) \rangle$, and (iii) $\langle f_4, (y \neq 4x) \rangle$.

Algorithm 9: *Simplify*

```
Input: LMDD $f$, Satisfiable path $\pi$,
       Conjunction $C$ of LMCs encountered along $\pi$
Output: Set of sub-problems $S$
if $f = 1$ then
    $S := S \cup \{\langle f, C \rangle\}$;
else
    if node $H(f)$ is in $\pi$ then
        $S := S \cup \{\langle L(f), C \wedge \neg P(f) \rangle\}$;
        simplify($H(f), \pi, C \wedge P(f)$);
    else
        $S := S \cup \{\langle H(f), C \wedge P(f) \rangle\}$;
        simplify($L(f), \pi, C \wedge \neg P(f)$);
```

Fig. 5 Splitting scheme in *simplify*



The function *selectPath* selects the path π in the following way. First, a solution m of f is generated using an SMT solver call. The root node of f is selected as the first node in π. The LMC $P(f)$ labeling the root node of f is then evaluated with values given to variables in its support as per m. If $P(f)$ evaluates to true under m, then $H(f)$ is selected as the next node in π. Otherwise if $P(f)$ evaluates to false under m, then $L(f)$ is selected as the next node in π. The LMC labeling the child of f thus selected as the next node in π is then evaluated under m. These steps are iteratively repeated until 1-terminal is encountered, each iteration adding a new node to π. Note that encountering 1-terminal is guaranteed since m is a solution of f.

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. In order to compute $\exists X.(f_i \wedge C_i)$, *QE_Combined* makes use of an algorithm *QE_LMDD_Mod*. *QE_LMDD_Mod* is a variant of *QE_LMDD* that eliminates a set of variables from an LMDD conjoined with a set of LMCs. *QE_Combined* initially sets g to false. In the first iteration of the loop, the satisfiability of $f_1 \wedge C_1$ is checked. If $f_1 \wedge C_1$ is satisfiable, then g is set to $\exists X.(f_1 \wedge C_1)$. Otherwise if $f_1 \wedge C_1$ is unsatisfiable, then the sub-problem $\exists X.(f_1 \wedge C_1)$ is redundant and is not computed. In the second iteration, the satisfiability of $f_2 \wedge C_2 \wedge \neg g$ is checked. If $f_2 \wedge C_2 \wedge \neg g$ is satisfiable, then $\exists X.(f_2 \wedge C_2)$ is computed and is disjoined with g. Otherwise if $f_2 \wedge C_2 \wedge \neg g$ is unsatisfiable, then $\exists X.(f_2 \wedge C_2)$ is redundant and is not computed. This loop is repeated until all the sub-problems are considered. It can be observed that g is equivalent to $\bigvee_{j=1}^{i} \big(\exists X.(f_j \wedge C_j)\big)$ after the ith iteration of the loop. Hence g is equivalent to $\bigvee_{j=1}^{n} \big(\exists X.(f_j \wedge C_j)\big)$ when the loop is terminated.

In our example, in the first iteration of the loop, the satisfiability of $f_3 \wedge (y = 4x) \wedge (x \neq z)$ is checked. Since $f_3 \wedge (y = 4x) \wedge (x \neq z)$ is satisfiable, g is set to $(2y = 0)$, the result of $\exists x.(f_3 \wedge (y = 4x) \wedge (x \neq z))$. In the second iteration, the satisfiability of $f_2 \wedge (y = 4x) \wedge (x = z) \wedge (2y \neq 0)$ is checked. $f_2 \wedge (y = 4x) \wedge (x = z) \wedge (2y \neq 0)$ is unsatisfiable, and hence $\exists x.(f_2 \wedge (y = 4x) \wedge (x = z))$ is not computed. Similarly, in the third iteration of the loop, the satisfiability of $f_4 \wedge (y \neq 4x) \wedge (2y \neq 0)$ is checked. $f_4 \wedge (y \neq 4x) \wedge (2y \neq 0)$ is unsatisfiable, and $\exists x.(f_4 \wedge (y \neq 4x))$ is also not computed. The final result of QE is $(2y = 0)$.

Note that 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 avoiding the computation of redundant sub-problems. This enables pruning the solution space of the problem, as achieved in *QE_SMT*.

5 Experimental results

We performed experiments to evaluate the performance and effectiveness of our QE algorithms, compare their performance with alternative QE techniques, and evaluate their utility in formal verification.

5.1 Experimental methodology and benchmarks

All 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 s. We implemented our own LMDD package for carrying out QE experiments involving LMDDs. In LMDDs the following heuristic was used to order the LMCs. We performed depth-first traversal of the DAG representations of formulae from which the LMDDs were created. Each new LMC encountered in the traversal was placed at the end of the current order. A similar variable ordering heuristic was used in the experiments involving BDDs. In *Project*, inside the layers, when there were multiple variables to eliminate, we used a simple lexicographic variable elimination order. Moreover, inside *Layer3*, the variables with constraints in coefficient-matched form were eliminated before the variables which required transformation to Boolean combination. In all experiments, we used simplifyingSTP as the SMT solver. simplifyingSTP was selected, because it has a variable eliminator [27] considered as suitable for solving bit-vector formulas involving LMEs. In experiments involving Omega Test, we used Pugh et al.'s implementation of Omega Test from [52].

The following simplification heuristics were used in the implementation. (i) The LMDs with modulus 2 were converted to equivalent LMEs. For example, the LMD $x + y \neq 1 \pmod 2$ was converted to $x+y = 0 \pmod 2$. We observed that this helps in easy elimination of existentially quantified variables involved in LMCs with modulus 2. (ii) In a non-terminal LMDD node u, if $P(u)$ is an LME, then it is kept in a normal form $2^k \cdot x = t$, where x is the variable appearing first in lexicographical ordering between the names of variables in the support of $P(u)$, and $k = \kappa(x, P(u))$ (recall the definition of κ from Sect. 3.1). This allows identification of equivalent LMEs during LMDD creation and hence more compact LMDDs.

We used a benchmark suite consisting of 198 *lindd* benchmarks [12] and 39 *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 [12] 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 preprocessing 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 designs taken from ITC99 benchmark suite [22], and the remaining are proprietary. We derived the symbolic transition relations of these VHDL designs. The *vhdl* benchmarks were obtained by quantifying out a subset of 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 8 to 50, 2 to 21, and 10 to 672 respectively.

Overview of Experiments We performed experimental evaluation of our QE techniques in three different ways.

1. *Experimental evaluation at the level of conjunctions of LMCs* This involved evaluation of performance and effectiveness of layers in *Project*, and comparison of the performance of *Project* with alternative QE techniques based on bit-blasting and conversion to linear integer arithmetic.
2. *Experimental evaluation at the level of Boolean combinations of LMCs* This involved evaluation of performance of the algorithms *QE_SMT*, *QE_LMDD*, and *QE_Combined* for QE from Boolean combinations of LMCs. We then compared the performance of *QE_SMT* with alternative QE techniques based on bit-blasting and conversion to linear integer arithmetic.
3. *Evaluation of utility of our techniques in verification* We selected a set of word-level VHDL designs, and derived their symbolic transition relations. We used *QE_LMDD* to compute abstract transition relations of these designs by quantifying out a subset of internal variables from the symbolic transition relations. We then compared the performance of bounded model checking using these abstract transition relations with that of bounded model checking using the original transition relations. We also evaluated the utility of our QE techniques in solving conjunctions of LMCs and for computing Craig interpolants for Boolean combinations of LMCs.

All benchmarks, implementations, and experimental data can be accessed from https://github.com/ajithkjohn123/BenchmarksImplementationExpdataForQE.git.

5.2 Evaluation of QE techniques for conjunctions of LMCs

5.2.1 Evaluation of layers in project

We performed QE from the benchmarks using the algorithms *QE_SMT*, *QE_LMDD*, and *QE_Combined*, and analyzed the *Project* calls that were generated during this process. Recall that *Layer3* involves transforming a conjunction of LMCs to a Boolean combination of LMCs and QE from this Boolean combination. As mentioned in Sect. 3.6, this 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) recursive *Project* calls. In our analysis, we focussed only on the initial/original *Project* calls. The recursive *Project* calls were considered as part of *Layer3*. In the subsequent discussion, whenever we mention "*Project* calls", it refers to the initial/original *Project* calls, unless stated otherwise.

The total number of *Project* calls generated from the *lindd* and *vhdl* benchmarks were 52,836 and 8027 respectively. Statistics of these *Project* calls are shown in Table 2. The contribution of a layer is measured as the ratio of the number of quantifiers eliminated by

Table 2 Details of *Project* calls (figures are per *Project* call)

Type	Vars	Qnt	LMIs	LMEs	LMDs	Contr			Time			*Pr*
						L1	L2	L3	L1	L2	L3	
lindd	39.9	38.1	(88, 0, 18.9)	(60, 0, 10.1)	(35, 0, 8.1)	51	44	5	3	5	13,149	674
vhdl	8.6	7.2	(4, 0, 0.3)	(16, 0, 5.8)	(31, 0, 2.0)	95	4.5	0.5	2	6	161	3

Vars: average number of variables, Qnt: 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*, Pr: *Project*, Time: average time spent per quantifier eliminated in milliseconds

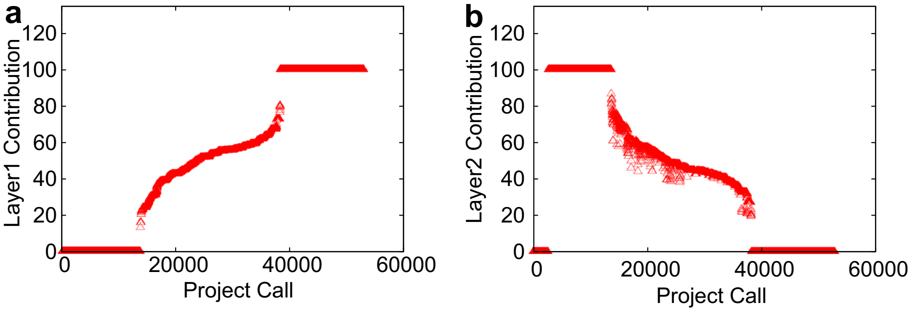
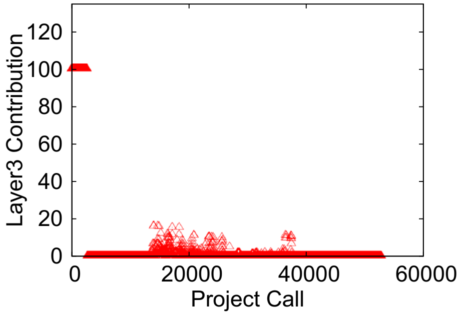


Fig. 6 Contribution of **a** *Layer1* and **b** *Layer2* for *lindd* benchmarks

Fig. 7 Contribution of *Layer3* for *lindd* benchmarks



the layer to the number of quantifiers to be eliminated in the *Project* call multiplied by 100. The time spent per quantifier eliminated for a layer is measured as the ratio of the time spent inside the layer to the number of quantifiers eliminated by the layer. The contributions of the layers and the times spent by the layers per quantifier eliminated for individual *Project* calls from *lindd* benchmarks are shown in Figs. 6, 7 and 10, and those for individual *Project* calls from *vhdl* benchmarks are shown in Figs. 8, 9 and 11. 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*

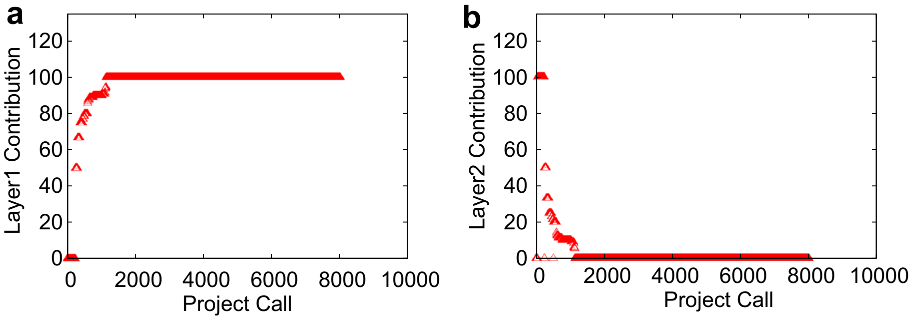


Fig. 8 Contribution of **a** *Layer1* and **b** *Layer2* for *vhdl* benchmarks

Fig. 9 Contribution of *Layer3* for *vhdl* benchmarks

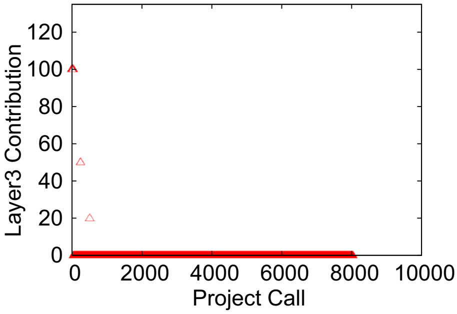
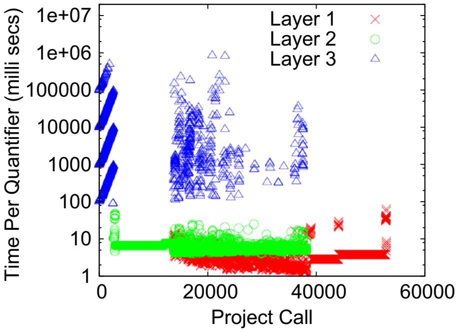


Fig. 10 Cost of layers for *lindd* benchmarks



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

5.2.2 Comparison with alternative QE techniques

We compared the performance of *Project* with QE based on linear integer arithmetic using Omega Test [8,51], and also with QE based on bit-blasting [40,53]. 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 the work in [27]), and then uses bit-blasting and BDD based bit-level QE [53] for the remaining variables.

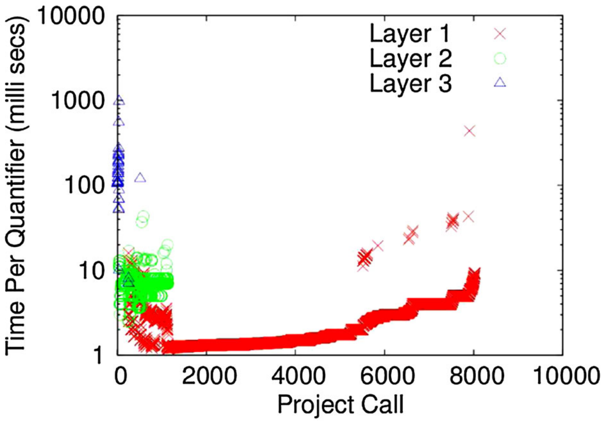


Fig. 11 Cost of layers for *vhdl* benchmarks

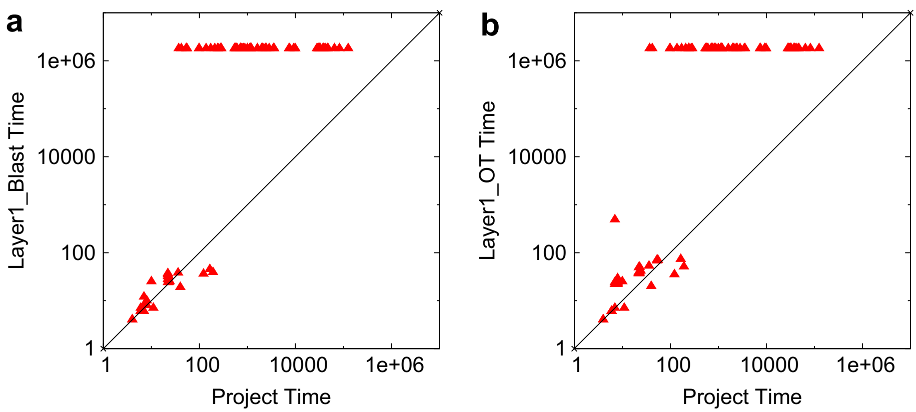


Fig. 12 Plots comparing **a** *Project* and *Layer1_Blast* and **b** *Project* and *Layer1_OT* (all times are in milliseconds)

(ii) *Layer1_OT*, *Layer2_OT*: *Layer1_OT* first quantifies out the variables using *Layer1*, and then uses conversion to linear integer arithmetic and Omega Test for the remaining variables. *Layer2_OT* first quantifies out the variables using *Layer1* followed by *Layer2*, and then uses conversion to linear integer arithmetic and Omega Test for the remaining variables. *Layer2_OT* helps us to compare the performance of *Layer3* with that of Omega Test.

We collected 100 instances of QE problem for conjunctions of LMCs arising from *QE_SMT* when QE is performed on the benchmarks. We performed QE from these conjunction-level problem instances using *Project*, *Layer1_Blast*, *Layer1_OT*, and *Layer2_OT*. Fig. 12a, b compare the QE times taken by *Project* against those taken by *Layer1_Blast* and *Layer1_OT* for each of these conjunction-level problem instances.

Project could successfully eliminate quantifiers in all of the 100 instances. *Layer1_Blast* was unsuccessful in in 68 cases and *Layer1_OT* were unsuccessful in 65 cases. These cases are indicated by the topmost points in Fig. 12a, b respectively. In most cases where *Layer1_Blast* and *Layer1_OT* were successful, the times taken by all the three algorithms were comparable. However there were a few cases where *Layer1_Blast* and *Layer1_OT* performed better than

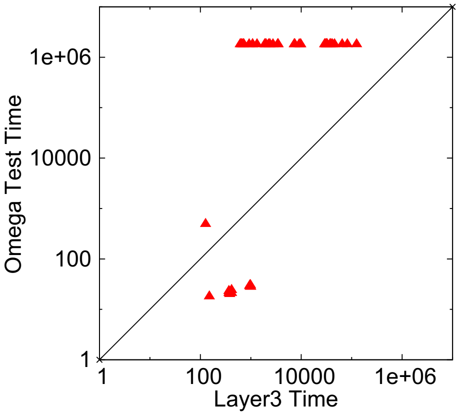


Fig. 13 Plot comparing *Layer3* and Omega Test (all times are in milliseconds)

Project. We found that these cases involved *Layer3*, and most of the time consumed by *Project* was spent inside *Layer3*.

We compared the times consumed by *Layer3* in *Project* with those consumed by Omega Test in *Layer2_OT* (see Fig. 13). There were 51 problem instances which required *Layer3*. Omega Test timed out in 37 of them. In 13 of the remaining 14 cases, Omega Test performed better than *Layer3*. Our analysis revealed that these cases were simpler in terms of number of LMCs and number of variables to be eliminated. However *Layer3* incurred several recursive *Project* calls in these cases.

Recall that given $\exists x.(C \wedge D \wedge I)$, where C is a conjunction of LMCs, D is a conjunction of LMDs and I is a conjunction of LMIs, *Layer2* checks if $\exists x.(C) \equiv \exists x.(C \wedge D \wedge I)$ holds. *Layer2* performs this check by computing an efficiently computable under-approximation of the number of ways in which an arbitrary solution of C can be engineered to satisfy $C \wedge D \wedge I$. We compared the performance of *Layer2* with a BDD based alternative technique to perform this check. We implemented a procedure *BddBasedLayer2* for this purpose. *BddBasedLayer2* computes BDDs for $\exists x.(C)$ and $\exists x.(C \wedge D \wedge I)$, and then checks if these BDDs are the same. Note that $\exists x.(C) \equiv \exists x.(C \wedge D \wedge I)$ holds iff the BDDs for $\exists x.(C)$ and $\exists x.(C \wedge D \wedge I)$ are the same. We then implemented procedure *ProjectWithBddBasedLayer2* which is a variant of *Project* that uses *BddBasedLayer2* in place of *Layer2*.

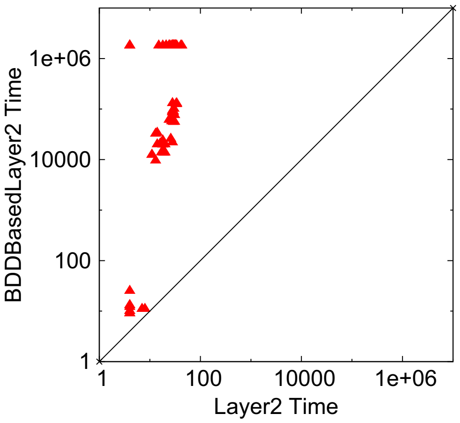
We performed QE from the 100 conjunction-level problem instances using *ProjectWithBddBasedLayer2*. For each problem instance, we then compared the time consumed by *Layer2* in *Project* with that consumed by *BddBasedLayer2* in *ProjectWithBddBasedLayer2* (see Fig. 14). *Layer2* outperformed the BDD based alternative technique in all the 100 problem instances.

5.3 Evaluation of QE techniques for Boolean combinations of LMCs

5.3.1 Evaluation of QE_SMT, QE_LMDD, and QE_Combined

We measured the time taken by *QE_SMT*, *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 Figs. 15, 16). Note that the points in Fig. 16 are scattered, while the points in Fig. 15a, b 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*

Fig. 14 Plot comparing *Layer2* and *BddBasedLayer2* (all times are in milliseconds)



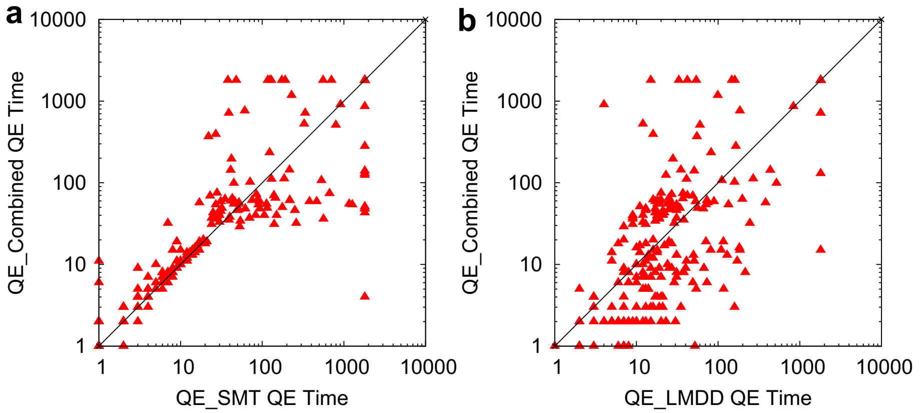
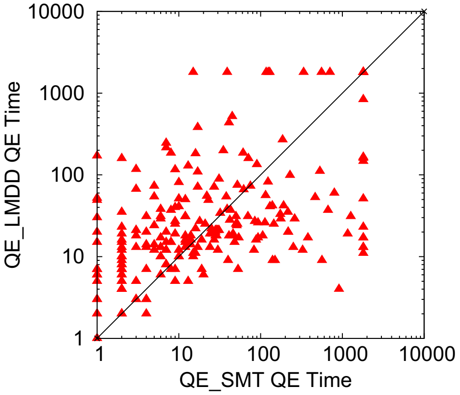


Fig. 15 Plots comparing **a** *QE_SMT* and *QE_Combined* and **b** *QE_LMDD* and *QE_Combined* (all times are in seconds)

Fig. 16 Plot comparing *QE_SMT* and *QE_LMDD* (all times are in seconds)



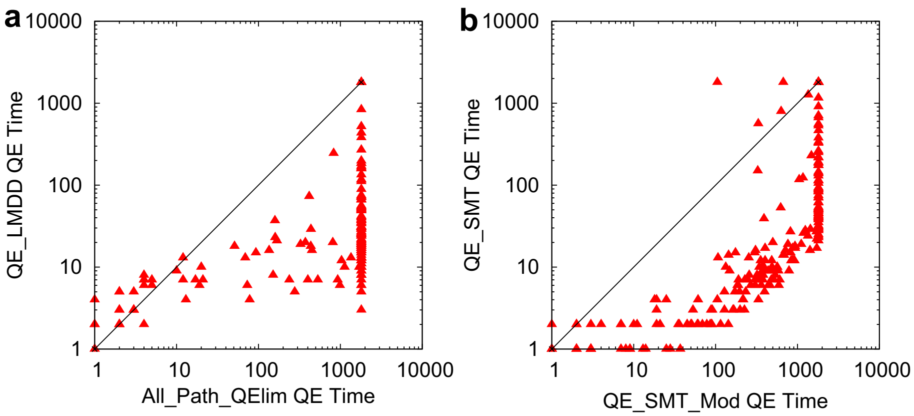


Fig. 17 Plots comparing **a** *All_Path_QElim* and *QE_LMDD* and **b** *QE_SMT* and *QE_SMT_Mod* (all times are in seconds)

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.

Recall that in *QE_Combined*, we converted $\exists X.f$, where f is an LMDD, into an equivalent disjunction of sub-problems, and then gave these sub-problems to *QE_LMDD_Mod* separately. Our analysis revealed that this helped in identifying redundant sub-problems. However, it was observed that splitting $\exists X.f$ into sub-problems and computing the sub-problems separately, reduced scope for reuse of results through dynamic programming when compared to computing $\exists X.f$ directly using *QE_LMDD*. We could also observe that using a more eager strategy for splitting into subproblems (i.e., a strategy that generates more sub-problems) in place of *simplify*, further reduced scope for reuse of results, although it improved opportunity for identifying redundant sub-problems. On the other hand, using a less eager strategy improved reuse of results, but gave less opportunity for identifying redundant sub-problems. Hence, although both reuse of results and splitting into subproblems contribute towards success of the hybrid approach, they act against each other. In our experiments, we found that the splitting scheme in *simplify* achieves a trade-off between them.

In order to evaluate the effectiveness of our simplifications in *QE_LMDD*, we compared the time taken by *QE_LMDD* with that taken by *All_Path_QElim* for QE from each benchmark (see Fig. 17a). *All_Path_QElim* succeeded only in a few cases. This is not surprising, as the LMDDs for the benchmarks contained a huge number of paths. In *QE_LMDD*, the single variable elimination strategy and the simplification of LMDDs using *simplifyLMDD* helped in achieving significant reuse of results through dynamic programming. This helped in avoiding path enumeration, which resulted in considerable performance gains over *All_Path_QElim*.

In order to evaluate the effectiveness of our generalization technique based on analysis of Boolean skeleton of formulae in *Generalize2*, we implemented a variant of *QE_SMT* called *QE_SMT_Mod*. *QE_SMT_Mod* is the same as *QE_SMT* except that it uses the implementation of *Generalize2* as proposed in [44]. Recall from Sect. 4.2 that the implementation of *Generalize2* in [44] makes use of SMT solver calls to identify unnecessary LMCs. We compared the time taken by *QE_SMT* and *QE_SMT_Mod* for QE from each benchmark (see Fig. 17b). *QE_SMT* outperformed *QE_SMT_Mod* except in a few cases. On profiling, we found that most of the time taken by *QE_SMT_Mod* was spent in the SMT solver calls in *Generalize2*. In the few cases where *QE_SMT_Mod* performed better than *QE_SMT*, the

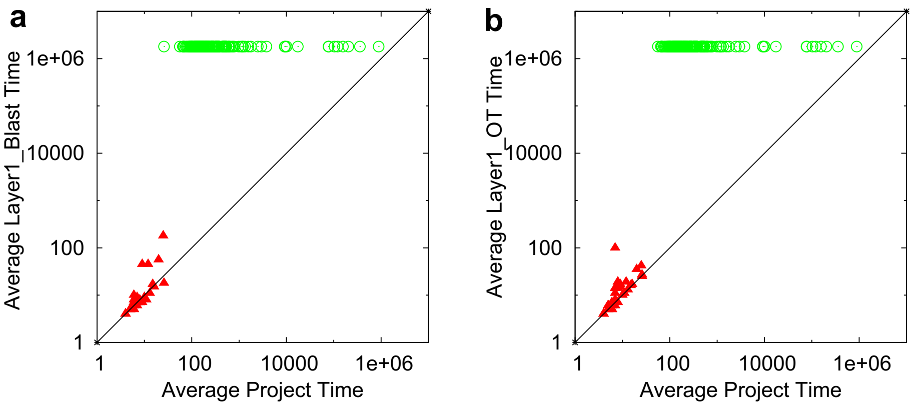


Fig. 18 Plots comparing average times consumed by **a** *Project* and *Layer1_Blast* and **b** *Project* and *Layer1_OT* when used inside *QE_SMT* (all times are in milliseconds). *Topmost green circles* indicate the benchmarks for which *Layer1_Blast* or *Layer1_OT* was unsuccessful. (Color figure online)

SMT solver based generalization in *QE_SMT_Mod* was more effective which helped in faster termination of the All-SMT loop.

5.3.2 Comparison with alternative QE techniques

We wanted to understand how *QE_SMT* would perform if a bit-blasting or linear integer arithmetic based alternative QE algorithm is used in place of *Project*. In order to do this, we first computed the average times taken by *Project* for QE from conjunction-level problem instances arising from *QE_SMT* when QE is performed on each benchmark. We also computed the average times taken by *Layer1_Blast*, *Layer1_OT*, and *Layer2_OT* for QE from these conjunction-level problem instances. For each benchmark, we then compared the average QE times taken by *Project* against those taken by *Layer1_Blast* and *Layer1_OT* (see Fig. 18a, b). 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. 19). For a large number of benchmarks, we observed that the bit-blasting or linear integer arithmetic based alternative QE algorithm was unsuccessful in eliminating quantifiers from the conjunction-level problem instances. These benchmarks are indicated by the topmost green circles in Figs. 18a, b and 19. Note that, for these benchmarks we could not compute the average times consumed by the bit-blasting or linear integer arithmetic based alternative QE algorithm, as the algorithm was unsuccessful in eliminating quantifiers from the conjunction-level problem instances. 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.

We also wanted to understand how *QE_SMT* would perform if the BDD based alternative technique *BddBasedLayer2* is used in place of *Layer2* inside *Project*. In order to do this, for each benchmark, we first computed the average time consumed by *Layer2* when QE is performed using *QE_SMT*. For each benchmark, we then computed the average time consumed by *BddBasedLayer2* when *BddBasedLayer2* is used in place of *Layer2* inside *Project*. Figure 20a compares these times. Many points corresponding to different benchmarks are merged in Fig. 20a, since the average times consumed in *Layer2* were significantly small compared those consumed in *BddBasedLayer2*. We provide a comparison of the total times in

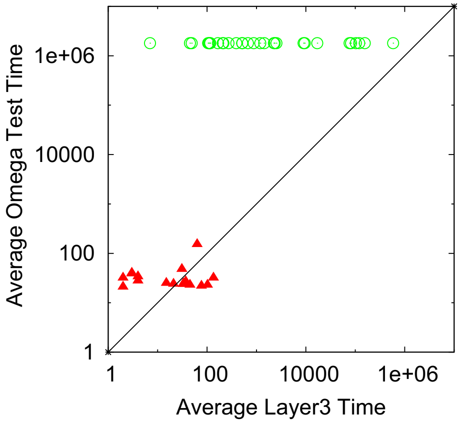


Fig. 19 Plot comparing average times consumed by *Layer3* and Omega Test when used inside *QE_SMT* (all times are in milliseconds). *Topmost green circles* indicate the benchmarks for which Omega Test was unsuccessful. (Color figure online)

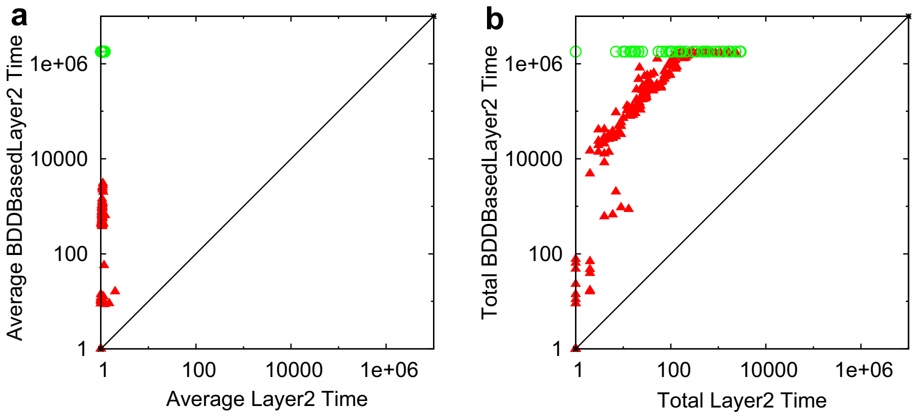


Fig. 20 Plot comparing **a** average times and **b** total times consumed by *Layer2* and *BddBasedLayer2* when used inside *QE_SMT* (all times are in milliseconds). *Topmost green circles* indicate the benchmarks for which *BddBasedLayer2* was unsuccessful. (Color figure online)

Fig. 20b for better exposition. The plots clearly demonstrate that *QE_SMT* performs poorly when the BDD based alternative technique is used in place of *Layer2*. Note that here again, topmost green circles in Fig. 20a, b indicate the benchmarks for which QE was unsuccessful when *BddBasedLayer2* was used in place of *Layer2*.

5.4 Utility of our QE algorithms in verification

5.4.1 Utility in bounded model checking

Recall that the *vhdl* benchmarks were obtained by quantifying out a subset of internal variables from the symbolic transition relations of word-level VHDL designs. The quantifier eliminated formulae give abstract transition relations of the VHDL designs. In order to evaluate the utility of our QE algorithms, we used *QE_LMDD* to compute these abstract transition relations, and then used these abstract transition relations for checking safety properties of the VHDL designs using bounded model checking.

Table 3 Experimental results on VHDL programs

Design	LOC	TR	N = 500	
			NA	QL
machine_1	363	(592, 22, 580)	TO(TO)	52(7, 23)
machine_2	373	(594, 22, 436)	TO(TO)	30(6, 1)
machine_3	383	(620, 25, 439)	TO(TO)	33(6, 3)
machine_4	253	(439, 26, 677)	1471(1441)	24(2, 0)
machine_5	253	(439, 26, 509)	1443(1413)	25(2, 0)
machine_6	363	(406, 17, 64)	78(53)	17(1, 1)
machine_7	379	(440, 22, 69)	221(196)	22(1, 3)
machine_8	251	(286, 20, 157)	193(177)	13(2, 0)
machine_9	251	(286, 20, 485)	331(315)	13(2, 0)
machine_10	363	(406, 17, 420)	TO(TO)	16(0, 1)
machine_11	363	(593, 22, 96)	TO(TO)	40(8, 4)
machine_12	363	(406, 17, 420)	TO(TO)	220(4, 187)
board_1	404	(400, 24, 194)	1442(1424)	21(12, 1)
board_2	373	(420, 24, 194)	TO(TO)	14(5, 1)
board_3	503	(573, 54, 361)	TO(TO)	16(5, 1)
board_4	415	(422, 28, 198)	241(223)	62(9, 2)

All times are in seconds. TO: >1800 s, LOC: lines of code, TR: transition relation details (dag size, number of variables, number of bits), NA: without abstraction: total time (simplifyingSTP time), QL: with *QE_LMDD* for abstraction: total time (*QE_LMDD* time, simplifyingSTP time), N: number of BMC unrollings

In order to check if the safety property holds for the first N cycles of operation, we first unrolled the transition relation N times, and conjoined the unrolled relation with the negation of the property. The resulting formula was then given to an SMT solver for checking satisfiability. Next, we obtained an abstract transition relation R' using *QE_LMDD*. The abstract transition relation was then unrolled N times and was conjoined with the negation of the property to obtain a formula, which was given to the SMT solver to check satisfiability.

All the SMT solver calls were unsatisfiable, which implies that the properties hold for the first N cycles of operation of the designs, and the abstract transition relations are sufficient to prove the properties. Table 3 gives a summary of the results for 16 designs. machine_1 to machine_12 are modified versions of benchmarks from ITC99 benchmark suite [22]. The remaining designs are proprietary. The table clearly shows the significant performance benefit of using abstract transition relations computed by *QE_LMDD* in these verification exercises.

For all the designs except machine_12, all the internal variables were eliminated from the transition relation in order to obtain the abstract transition relation. For machine_12, a manually chosen subset of internal variables were eliminated. It was observed that in all the cases, *Layer1* and *Layer2* were sufficient to eliminate the variables, without any call to *Layer3*. *Layer2* was needed only in five cases: machine_6 through machine_10. In these cases *Layer2* eliminated 12.5% to 40% of the quantified variables.

5.4.2 Utility in other applications

We performed limited preliminary experiments to evaluate the utility of *Layer1* and *Layer2* as preprocessing steps for conjunctions of LMCs before finding satisfying assignments for

Table 4 Experimental results on preprocessing using *Layer1* and *Layer2*

Set	V	E	D	I	NP	PR	AP
set_1	20	14	13	13	1763	1572	2688
set_2	20	0	36	4	3270	251	0
set_3	20	0	4	36	3208	655	3245
set_4	30	20	20	20	8415	4769	9216
set_5	30	0	54	6	7423	533	0
set_6	30	0	6	54	7203	1651	7218
set_7	40	28	26	26	223,880	11,255	171,207
set_8	40	0	72	8	14,115	1150	0
set_9	40	0	8	72	14,343	3561	13,238

All times are in milliseconds. V: number of variables, E: number of LMEs, D: number of LMDs, I: number of LMIs, NP: average time taken by simplifyingSTP for solving the benchmarks in the set without preprocessing, PR: average time for preprocessing the benchmarks in the set, AP: average time taken by simplifyingSTP for solving the benchmarks in the set after preprocessing

the conjunctions using an SMT solver. Towards this end, we generated 9 sets of random benchmarks. Each set included 5 benchmarks that are randomly generated conjunctions of LMCs with the same number of variables, LMEs, LMDs and LMIs. The moduli of all LMCs in all benchmarks was fixed to 2^{24}. The number of variables varied from 20 to 50. The number of LMCs was chosen as twice the number of variables.

In order to properly evaluate the effectiveness of *Layer1* and *Layer2*, we generated three types of benchmarks. Type-1 benchmarks contained an equal mix of LMEs, LMDs, and LMIs. The benchmarks in set_1, set_4, and set_7 in Table 4 were of this type. These benchmarks allowed us to evaluate the effectiveness of *Layer1*. In type-2 benchmarks, 80% of constraints were LMDs and the remaining were LMIs. The benchmarks in set_2, set_5, and set_8 in Table 4 were of this type. Finally, in type-3 benchmarks, 80% of constraints were LMIs and the remaining were LMDs. The benchmarks in set_3, set_6, and set_9 in Table 4 were of type-3. Type-2 and type-3 benchmarks allowed us to evaluate the effectiveness of *Layer2* on different mixes of constraints.

We first measured the time taken by simplifyingSTP to solve each benchmark. We then eliminated variables in the support of each benchmark using *Layer1* and *Layer2*. This yields a potentially simplified benchmark with fewer variables in the support. We then measured the time taken by simplifyingSTP to solve each preprocessed benchmark. Table 4 gives a summary of the results. Preprocessing helped in cases of type-2 benchmark sets set_2, set_5, and set_8. Preprocessing in these cases completely solved the problem instances. In other cases preprocessing either caused additional overhead or was of not much use.

We also performed limited preliminary experiments to evaluate the utility of our QE techniques for computing Craig interpolants for Boolean combinations of LMCs. Towards this end, we generated a set of interpolation benchmarks. Each benchmark is a pair of formulas (φ, ψ), where φ, ψ are Boolean combinations of LMCs which are mutually inconsistent. We denote the set of variables in the support of both φ and ψ as Y. The set of variables in the support of φ but not in the support of ψ is denoted as X. Similarly, the set of variables in the support of ψ but not in the support of φ is denoted as Z.

Note that $\exists X.\varphi$ serves as an interpolant for (φ, ψ). In fact, $\exists X.\varphi$ is the *strongest* interpolant for (φ, ψ). For each interpolation benchmark, we first used *QE_Combined* to compute $\exists X.\varphi$. For each benchmark, we then used Mathsat to compute an interpolant (Mathsat makes use

Table 5 Experimental results on computing interpolants

Benchmark	|X|	|Y|	|Z|	W	MS	QC
benchmark_1	5	16	12	16	12	36
benchmark_2	6	15	8	16	45	44
benchmark_3	8	10	6	8	0	3
benchmark_4	17	23	11	32	7	275
benchmark_5	17	23	10	32	6	142
benchmark_6	21	25	8	32	TO	TO
benchmark_7	12	7	7	22	TO	16
benchmark_8	10	17	8	32	0	29
benchmark_9	3	10	3	32	TO	12
benchmark_10	4	14	3	16	TO	7

All times are in seconds. TO: >1800 s, |X|: number of variables in set X, |Y|: number of variables in set Y, |Z|: number of variables in set Z, W: maximum bit-width of a variable, MS: time taken by Mathsat, QC: time taken by *QE_Combined*

of work in [31] for interpolant computation). We then compared the time taken by Mathsat to compute interpolant with that taken by *QE_Combined* to compute $\exists X.\varphi$. Table 5 gives a summary of the results for 10 benchmarks. Since interpolation is an approximation of QE, it is amenable to simplifications that QE may not be able exploit. Nevertheless our experiments show that the two techniques are incomparable. In some cases, an interpolant can be computed faster than the quantifier-eliminated formula, while in other cases, QE using our techniques can be done much faster than computing an interpolant using the techniques encoded in MathSAT.

Considering the three sets of experiments that we performed for evaluating the utility of our QE techniques, it can be seen that our techniques are convincingly useful for computing abstract transition relations in bounded model checking. Our experiments showed that applying our techniques often translates to a model checking problem being solved within given time constraints, as opposed to timing out. However the other two sets of experiments—applying our techniques in solving conjunctions of LMCs and for computing Craig interpolants for Boolean combinations of LMCs—gave mixed results. Exploring other applications of our techniques is part of future work.

6 Conclusions and future work

We presented a practically efficient and bit-precise algorithm for QE from conjunctions of LMCs. Our algorithm made use of a layered framework—incomplete and cheaper layers are applied first, expensive and complete layers are called only when required. Each of our layers is motivated by QE problem instances that occur in practice. Our studies revealed that using a layered framework allows us to solve such problem instances efficiently using incomplete and cheaper techniques rather than resorting to expensive and complete techniques. Our layers make use of properties of modular arithmetic and keep the quantifier-eliminated formula in modular arithmetic. We extended this algorithm to work with arbitrary Boolean combinations of LMCs. Experiments demonstrated that our techniques significantly outperform alternative QE techniques.

There are several promising directions for future work. Our experiments showed that Layer3 is significantly expensive compared to Layer2. As part of future work, we will explore development of new cheaper layers between Layer2 and Layer3. It is interesting to study how our techniques can be extended to QE from full bit-vector arithmetic. Other than linear modular arithmetic operations, bit-vector arithmetic primarily includes extractions, concatenations, non-linear multiplications and bit-wise operations. Many QE problem instances that arise in practice frequently mix expressions from different theories. It is interesting to understand how our techniques can be extended to work in combined theories such as combination of linear modular arithmetic and equality over uninterpreted functions, combination of linear modular arithmetic and array logic etc. Another interesting direction in future work is to integrate our QE techniques with SMT solvers, which will allow SMT solvers to use these techniques to reason about quantified bit-vector formulas.

We showed the utility of our techniques in computing abstract symbolic transition relations for improving the scalability of bounded model checking of word-level RTL designs. We also presented preliminary experiments that demonstrate the utility of our techniques in solving conjunctions of LMCs and computing Craig interpolants for Boolean combinations of LMCs. There are many other applications that can potentially benefit from our QE techniques. Our techniques can be used for computation of predicate abstractions, computation of strongest post-conditions and image computation in the verification of word-level RTL designs and embedded programs. In a Counterexample-Guided Abstraction Refinement (CEGAR) [14] framework, our techniques can be used to compute Craig interpolants from spurious counterexamples. We plan to explore these applications in future.

References

1. Ax J, Kochen S (1965) Diophantine problems over local fields II. A complete set of axioms for p-adic number theory. Am J Math 87(3):631–648
2. Babic D, Musuvathi M (2005) Modular arithmetic decision procedure. Technical report TR-2005-114, Microsoft Research
3. Bierre A, Cimatti A, Clarke EM, Zhu Y (1999) Symbolic model checking without BDDs. In: Proceedings of international conference on tools and algorithms for the construction and analysis of systems (TACAS), pp 193–207
4. Bjørner N (2010) Linear quantifier elimination as an abstract decision procedure. In: Proceedings of international joint conference on automated reasoning (IJCAR), pp 316–330
5. Bjørner N, Pichora M (1998) Deciding fixed and non-fixed size bit-vectors. In: Proceedings of international conference on tools and algorithms for the construction and analysis of systems (TACAS), pp 376–392
6. Bjørner N, Janota M (2015) Playing with quantified satisfaction. In: Proceedings of international conferences on logic for programming, artificial intelligence and reasoning (LPAR)—short presentations, pp 15–27
7. Bjørner N, Blass A, Gurevich Y, Musuvathi M (2008) Modular difference logic is hard. CoRR abs/0811.0987
8. Brinkmann R, Drechsler R (2002) RTL-datapath verification using integer linear programming. In: Proceedings of IEEE VLSI design conference, pp 741–746
9. Bruttomesso R, Sharygina N (2009) A scalable decision procedure for fixed-width bit-vectors. In: Proceedings of international conference on computer-aided design (ICCAD), pp 13–20
10. Bryant R (1986) Graph-based algorithms for boolean function manipulation. IEEE Trans Comput 35(8):677–691
11. Cavada R, Cimatti A, Franzen A, Kalyanasundaram K, Roveri M, Shyamasundar RK (2007) Computing predicate abstractions by integrating BDDs and SMT solvers. In: Proceedings of international conference on formal methods in computer-aided design (FMCAD), pp 69–76
12. Chaki S, Gurfinkel A, Strichman O (2009) Decision diagrams for linear arithmetic. In: Proceedings of international conference on formal methods in computer-aided design (FMCAD), pp 53–60
13. Clarke EM, Grumberg O, Peled D (1999) Model checking. MIT Press, Cambridge
14. Clarke EM, Grumberg O, Jha S, Lu Y, Veith H (2000) Counterexample-guided abstraction refinement. In: Proceedings of international conference on computer aided verification (CAV), pp 154–169
15. Cohen P (1969) Decision procedures for real and p-adic fields. Commun Pure Appl Logic 25:213–231
16. Cooper D (1972) Theorem proving in arithmetic without multiplication. Mach Intell 7:91–99

17. Craig W (1957) Linear reasoning: a new form of the Herbrand–Gentzen theorem. J Symb Logic 22(3):250–268
18. Cyrluk D, Möller M, Rueß H (1997) An efficient decision procedure for the theory of fixed-sized bit-vectors. In: Proceedings of international conference on computer aided verification (CAV), pp 60–71
19. Damm W, Dierks H, Disch S, Hagemann W, Pigorsch F, Scholl C, Waldmann U, Wirtz B (2012) Exact and fully symbolic verification of linear hybrid automata with large discrete state spaces. Sci Comput Program 77(10–11):1122–1150
20. Dantzig GB, Eaves BC (1973) Fourier–Motzkin elimination and its dual. J Comb Theory Ser A 14(3):288–297
21. Das S (2003) Predicate abstraction. PhD thesis, Stanford University
22. Davidson S (1999) Characteristics of the ITC'99 benchmark circuits. http://cerc.utexas.edu/itc99-benchmarks/bench.html
23. de Moura L, Bjørner N (2007) Relevancy propagation. Technical report TR-2007-140, Microsoft Research
24. de Moura L, Bjørner N (2008) Z3: an efficient SMT solver. In: Proceedings of international conference on tools and algorithms for the construction and analysis of systems (TACAS), pp 337–340
25. Déharbe D, Fontaine P, Berre DL, Mazure B (2013) Computing prime implicants. In: Proceedings of international conference on formal methods in computer-aided design (FMCAD), pp 46–52
26. Ferrante J, Rackoff C (1975) A decision procedure for the first order theory of real addition with order. Soc Ind Appl Math (SIAM) J Comput 4(1):69–76
27. Ganesh V, Dill D (2007) A decision procedure for bit-vectors and arrays. In: Proceedings of international conference on computer aided verification (CAV), pp 519–531
28. Ganesh V, Berezin S, Dill D (2002) Deciding Presburger arithmetic by model checking and comparisons with other methods. In: Proceedings of international conference on formal methods in computer-aided design (FMCAD), pp 171–186
29. Gange G, Søndergaard H, Stuckey P, Schachte P (2013) Solving difference constraints over modular arithmetic. In: Proceedings of international conference on automated deduction (CADE), pp 215–230
30. Gotlieb A, Leconte M, Marre B (2010) Constraint solving on modular integers. In: Proceedings of ninth international workshop on constraint modelling and reformulation (ModRef) co-located with international conference on principles and practice of constraint programming (CP)
31. Griggio A (2011) Effective word-level interpolation for software verification. In: Proceedings of international conference on formal methods in computer-aided design (FMCAD), pp 28–36
32. Hadarean L, Bansal K, Jovanovic D, Barret C, Tinelli C (2014) A tale of two solvers: eager and lazy approaches to bit-vectors. In: Proceedings of international conference on computer aided verification (CAV), pp 680–695
33. Howell JA, Gregory RT (1969) An algorithm for solving linear algebraic equations using residue arithmetic I. BIT Numer Math 9(3):200–224
34. Huang C, Cheng K (2000) Assertion checking by combined word-level ATPG and modular arithmetic constraint-solving techniques. In: Proceedings of ACM/IEEE design automation conference (DAC), pp 118–123
35. Jain H, Clarke EM, Grumberg O (2008) Efficient Craig interpolation for linear diophantine (dis)equations and linear modular equations. In: Proceedings of international conference on computer aided verification (CAV), pp 254–267
36. John A, Chakraborty S (2011) A quantifier elimination algorithm for linear modular equations and disequations. In: Proceedings of international conference on computer aided verification (CAV), pp 486–503
37. John A, Chakraborty S (2013) Extending quantifier elimination to linear inequalities on bit-vectors. In: Proceedings of international conference on tools and algorithms for the construction and analysis of systems (TACAS), pp 78–92
38. Kapur D (2006) A quantifier-elimination based heuristic for automatically generating inductive assertions for programs. J Syst Sci Complex 19(3):307–330
39. Komuravelli A, Gurfinkel A, Chaki S (2014) SMT-based model checking for recursive programs. In: Proceedings of international conference on computer aided verification (CAV), pp 17–34
40. Kroening D, Strichman O (2008) Decision procedures: an algorithmic point of view. Springer, Berlin
41. Lahiri S, Nieuwenhuis R, Oliveras A (2006) SMT techniques for fast predicate abstraction. In: Proceedings of international conference on computer aided verification (CAV), pp 424–437
42. Loos R, Weispfenning V (1993) Applying linear quantifier elimination. Comput J 36(5):450–462
43. Mishchenko A, Chatterjee S, Jiang R, Brayton R (2005) FRAIGs: a unifying representation for logic synthesis and verification. Technical report, EECS Department, UC Berkeley
44. Monniaux D (2008) A quantifier elimination algorithm for linear real arithmetic. In: Proceedings of international conference on logic for programming artificial intelligence and reasoning (LPAR), pp 243–257

45. Monniaux D (2010) Quantifier elimination by lazy model enumeration. In: Proceedings of international conference on computer aided verification (CAV), pp 585–599
46. Müller-Olm M, Seidl H (2007) Analysis of modular arithmetic. ACM Trans Program Lang Syst (TOPLAS) 29(5):29
47. Niemetz A, Preiner M, Biere A (2014) Turbo-charging lemmas on demand with don't care reasoning. In: Proceedings of international conference on formal methods in computer-aided design (FMCAD), pp 179–186
48. Nipkow T (2008) Linear quantifier elimination. In: Proceedings of international joint conference on automated reasoning (IJCAR), pp 18–33
49. Owre S, Rushby J, Shankar N (1992) PVS: A prototype verification system. In: Proceedings of international conference on automated deduction (CADE), pp 748–752
50. Phan A, Bjørner N, Monniaux D (2012) Anatomy of alternating quantifier satisfiability (work in progress). In: Proceedings of SMT workshop at international joint conference on automated reasoning (SMT@IJCAR), pp 120–130
51. Pugh W (1992) The Omega test: a fast and practical integer programming algorithm for dependence analysis. Commun ACM 35(8):102–114
52. Pugh W (2013) The Omega project: frameworks and algorithms for the analysis and transformation of scientific programs. www.cs.umd.edu/projects/omega
53. Somenzi F (2015) CUDD: Colorado university decision diagram package release 3.0.0. http://vlsi.colorado.edu/~fabio/CUDD
54. Szabo N, Tanaka R (1967) Residue arithmetic and its applications to computer technology. McGraw-Hill, New York
55. Tew N, Kalla P, Shekhar N, Gopalakrishnan S (2008) Verification of arithmetic datapaths using polynomial function models and congruence solving. In: Proceedings of international conference on computer-aided design (ICCAD), pp 122–128
56. Veanes M, Bjørner N, Nachmanson L, Bereg S (2014) Monadic decomposition. In: Proceedings of international conference on computer aided verification (CAV), pp 628–645
57. Wintersteiger C, Hamadi Y, de Moura L (2010) Efficiently solving quantified bit-vector formulas. In: Proceedings of international conference on formal methods in computer-aided design (FMCAD), pp 239–246