Word-level Quantifier Elimination

Supratik Chakraborty
IIT Bombay

(Joint work with Ajith John)

VMCAI 2015 (Jan 14, 2015)

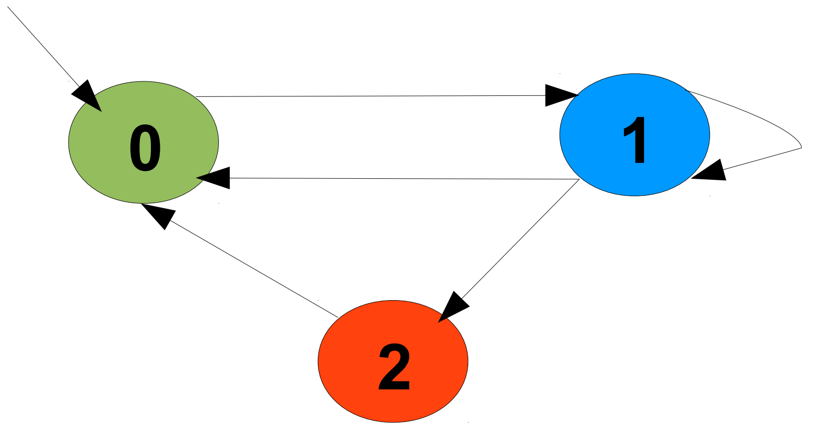
Example Embedded Code

```
...
state = 0; done = 0;
while (more_inputs() || (done != 0)) {
    if (state == 0) {
      a = s1.rd(); b = s2.rd(); x = 0;
      state = 1; done = 0;
    }
    else if (state == 1) {
      if (x+a <= b)  {
          x = x+1;  a = 2*a;
      }
      else if (x == b+1) state = 2;
      else { state = 0; done = 1;}
    }
    else if (state == 2) {
      state = 0; done = 1;
      if (0 < a < x)  RaiseAlarm();
    }
}
```

Repeatedly

- Read a, b from sensors/file
- Iteratively compute smallest

  x s.t. $2^x * a + x > b$
- If smallest x is b+1 and (0 < a < x), raise alarm



Q: Can alarm be raised?

Example Embedded Code

```
...
state = 0; done = 0;
while (more_inputs() || (done != 0)) {
    if (state == 0) {
      a = s1.rd(); b = s2.rd(); x = 0;
      state = 1; done = 0;
    }
    else if (state == 1) {
      if (x+a <= b)  {
          x = x+1;  a = 2*a;
      }
      else if (x == b+1) state = 2;
      else { state = 0; done = 1;}
    }
    else if (state == 2) {
      state = 0; done = 1;
      if (0 < a < x)  RaiseAlarm();
    }
}
```

Repeatedly

- Read a, b from sensors/file
- Iteratively compute smallest
  x s.t. $2^x * a + x > b$
- If smallest x is b+1 and
  $(0 < a < x)$, raise alarm

NO, if a, b, x are unbounded unsigned int **(surely $2^b*a+b > b$)**

YES, if a, b, x are 8-bit unsigned int, all ops are mod 2^8
(consider $a = 2^6$, $b = 2^7+2$)

Example Embedded Code

```
...
state = 0; done = 0;
while (more_inputs() || (done != 0)) {
    i
    }
    e
      if (x+a <= b)  {
          x = x+1;  a = 2*a;
      }
      else if (x == b+1) state = 2;
      else { state = 0; done = 1;}
    }
    else if (state == 2) {
      state = 0; done = 1;
      if (0 < a < x)  RaiseAlarm();
    }
}
```

Repeatedly

Read a, b from sensors/file

Iteratively compute smallest

Need for bit-precise reasoning

NO, if a, b, x are unbounded unsigned int **(surely 2^b*a+b >b)**

YES, if a, b, x are 8-bit unsigned int, all ops are mod 2^8 **(consider a = 2^6, b = 2^7+2)**

Modeling Controller Code

- Transition relation formula of one unfolding of loop

$$state' = ite(state = 0, 1, ite(state = 1, ite(x+a \le b, 1, ite(x = b+1, 2, 0)), 0))$$

$$\wedge$$

$$a' = ite(state = 0, s1\_rd, ite(state = 1, ite(x+a \le b, 2*a, a), a))$$

$$\wedge$$

$$b' = \boxed{ite(state = 0, s2\_rd, b)}$$

$$\wedge$$

$$x' = ite(state = 0, 0, ite(state = 1, ite(x+a \le b, x+1, x), x))$$

Term: **If (state = 0) then s2_rd else b**

state, a, b, x: Values before execution of loop body

state', a', b', x': Values after one execution of loop body

Modeling Controller Code

- Transition relation formula of one unfolding of loop

state' = ite(state = 0, 1, ite(state = 1, ite(x+a ≤ b, 1, ite(x = b+1, 2, 0)), 0))

∧

a' = ite(state = 0, s1_rd, ite(state = 1, ite(x+a ≤ b, 2*a, a), a))

∧

b' = ite(state = 0, s2_rd, b)

∧

x' = ite(state = 0, 0, ite(state = 1, ite(x+a ≤ b, x+1, x), x))

Consider temp = ite(x = b+1, 2, 0)

Sub-formula: If (x=b+1) then (temp = 2) else (temp = 0)

((x = b+1) ∧ (temp= 2)) ∨ ((x ≠ b+1) ∧ (temp= 0))

Linear equality

Modeling Controller Code

- Transition relation formula of one unfolding of loop

$$state' = ite(state = 0, 1, ite(state = 1, ite(x+a \leq b, 1, \boxed{ite(x = b+1, 2, 0)}), 0))$$

$$\wedge$$

$$a' = ite(state = 0, s1\_rd, ite(state = 1, ite(x+a \leq b, 2*a, a), a))$$

$$\wedge$$

$$b' = ite(state = 0, s2\_rd, b)$$

$$\wedge$$

$$x' = ite(state = 0, 0, ite(state = 1, ite(x+a \leq b, x+1, x), x))$$

Consider temp = ite(x = b+1, 2, 0)

Sub-formula: If (x=b+1) then (temp = 2) else (temp = 0)

$$((x = b+1) \wedge (temp = 2)) \vee ((x \neq b+1) \wedge (temp = 0))$$

Linear disequality

Modeling Controller Code

- Transition relation of one unfolding of loop

$$\text{state}' = \text{ite}(\text{state} = 0, 1, \text{ite}(\text{state} = 1, \text{ite}(x+a \le b, 1, \text{ite}(x = b+1, 2, 0)), 0))$$

$$\wedge$$

$$a' = \text{ite}(\text{state} = 0, \text{s1\_rd}, \text{ite}(\text{state} = 1, \boxed{\text{ite}(x+a \le b, 2*a, a)}, a))$$

$$\wedge$$

$$b' = \text{ite}(\text{state} = 0, \text{s2\_rd}, b)$$

$$\wedge$$

$$x' = \text{ite}(\text{state} = 0, 0, \text{ite}(\text{state} = 1, \text{ite}(x+a \le b, x+1, x), x))$$

Linear inequality

$$((x+a \le b) \wedge \ldots) \vee ((x+a > b) \wedge \ldots)$$

Trans relation R($\underbrace{\textbf{state, a, b, x,}}_{Y}$ $\underbrace{\textbf{state', a', b', x'}}_{Y'}$ **) :**

Boolean combination of Linear Arithmetic Formulae

Whither Quantifiers?

- Computing strongest post-condition (SP) of a loop
  - Suppose state at start of loop satisfies $\varphi(Y)$
  - What will state after one loop itern satisfy?
  - $SP(\varphi(Y), \text{loop-body}) = \exists Y. (\varphi(Y) \wedge R(Y, Y'))$

    $= \text{a formula on } Y'$

Whither Quantifiers?

- Bounded model checking
  - Values before iteration satisfy I(Y)
  - Can values satisfy Bad(Z) at the end of k iterations?
  - Check satisfiability of

    $I(Y_0) \land R(Y_0, Y_1) \land \ldots \land R(Y_{k-1}, Y_k) \land Bad(Z_k)$
  - Includes all variables in each unrolling
    - Bottleneck if k is large
  - Can we use an abstract transition relation?
    - $R'(W, W') = \exists(Y \setminus W)\exists(Y' \setminus W').\ R(Y, Y')$
    - $W \subseteq Y$ and $W' \subseteq Y'$

$Z \subseteq Y$

Whither Quantifiers?

- Projections based state abstractions
  - State: Values of all variables in program at given program location
    - e.g. $a = 0, b = 1, x = 0$
  - Set of states:
    - $(a,b,x) \in \{(0,1,0), (2,8,3), (100,5,98), \ldots\}$
  - Symbolic state:
    - Formula on variables
    - Represents set of states that satisfy formula
    - e.g. $(b + x) > a$ … as integers

Whither Quantifiers?

- Projections based state abstractions
  - What if values of only some variables are interesting or relevant?
  - Simply symbolic state formula
    - e.g. symbolic state $\varphi(Y)$, but only values of vars in $W \subseteq Y$ relevant
    - $\text{Abstract}(\varphi(Y)) = \exists (Y \setminus W).\ \varphi(Y)$
- Program Synthesis
  - Several key steps involve existentially quantifying variables from formulae

Why Eliminate Quantifiers?

- Reasoning about quantified formulas more difficult in practice
- Efficient decision procedures for several quantifier-free theories exist
  - Corresponding quantified theories may not have efficient decision procedures
  - E.g. linear arithmetic over reals
- Bounded Model Checking with abstract transition relations
  - Fewer vars in formula if abstract trans relation is quantifier-free

Quantifier Elimination (QE)

- Theory T admits QE if
  - for every quantified formula $\varphi(Y)$ in T, there is a quantifier-free formula $\varphi'(Y)$ s.t. $\varphi(Y) \equiv_T \varphi'(Y)$
- Not every theory admits QE
  - Theory of fixed-width bit-vectors does
  - Theory of monadic predicates does not
- **QE algorithm for T** (that admits QE)
  - Given a quantified formula in T, generates an equivalent quantifier-free formula in T

Motivation

- Verification tools assuming integer / real types for program variables can give incorrect results

- Machine arithmetic not same as integer / real arithmetic

Motivating Word-level QE

- Verification tools assuming integer / real types for program variables can give incorrect results

- Machine arithmetic not same as integer / real arithmetic

$$(x \geq 3) \wedge (x + 1 \leq 2)$$

sat on 2-bit bit-vectors

unsat on integers/reals

Motivation

- Verification tools assuming integer / real types for program variables can give incorrect results

- Machine arithmetic not same as integer / real arithmetic

$(x \geq 3) \wedge (x + 1 \leq 2)$

sat on 2-bit bit-vectors

unsat on integers/reals

$(x \geq 3) \wedge (y > x)$

unsat on 2-bit bit-vectors

sat on integers/reals

Motivation

- Verification tools assuming integer / real types for program variables can give incorrect results

- Machine arithmetic not same as integer / real arithmetic

sat on 2-bit bit-vectors

$(x \geq 3) \wedge (x + 1 \leq 2)$

unsat on integers/reals

unsat on 2-bit bit-vectors

$(x \geq 3) \wedge (y > x)$

sat on integers/reals

- Motivates bit-precise (word-level) reasoning techniques
  - **Focus on linear bit-vector arithmetic constraints**

Notation

- ***LME*** : $c_1.x_1 + \ldots + c_n.x_n = c_0 \pmod{2^p}$
- ***LMD*** : $c_1.x_1 + \ldots + c_n.x_n \neq c_0 \pmod{2^p}$
- ***LMI*** : $c_1.x_1 + \ldots + c_n.x_n + c_0 \leq d_1.x_1 + \ldots + d_n.x_n + d_0 \pmod{2^p}$
- ***LMC*** : LME, LMD or LMI (linear arithmetic modulo congruence)

p : a +ve integer constant
2^p : modulus
$x_1,\ldots,x_n$: p-bit non-negative integer variables
$c_0,\ldots,c_n$, $d_0,\ldots,d_n$: p-bit non-negative integer constants

Assume for now all LMCs have the same modulus

Quick Partial Literature Survey

Classical work

Presburger Arithmetic with congruence relation admits QE [*Presburger 1929*]

QE algorithm results in exponential (in #vars quantified) blowup in all but the simplest cases

Scalability issues in practice

More Efficient Reasoning about LMEs and LMDs

- Reducing LMEs into solved form : *Ganesh & Dill., 2007*

- Interpolation algorithm for LMEs, hardness of satisfiability problem for LMDs/LMIs : *Jain et al 2008, Bjorner et al 2008*

- QE algorithm for LMEs and LMDs : *John & C. 2011, 2013*

Quick Literature Survey

QE from LMCs

- Bit-blasting + QE at bit-level
  - Destroys the word-level structure
  - Does not scale well for LMCs with large modulus
- Conversion to Integer Linear Arithmetic (ILA) + ILA QE

  Brinkmann et al 2002
  - Converting back to modular arithmetic difficult
  - Blow-up in many practical cases

Outline

- QE from conjunctions of LMCs: layered algorithm
- Extending to Boolean combinations
- Experimental results
- Conclusion

Layer 1: Substituion (*Ganesh & Dill 2007)*

$\exists x.((2x+z \neq 0) \wedge (2x+3y = 4) \wedge (x+y \leq 3))$ mod 8

Layer 1: Substituion (*Ganesh & Dill 2007)*

$$\exists x.((2x+z \neq 0) \wedge (2x+3y = 4) \wedge (x+y \leq 3)) \bmod 8$$

$$\Big\downarrow \; 2x + 3y + 5y = 4 + 5y$$

$$\exists x.((2x+z \neq 0) \wedge (2x = 5y+4) \wedge (x+y \leq 3)) \bmod 8$$

Layer 1: Substitution (*Ganesh & Dill 2007*)

$\exists x.((2x+z \neq 0) \wedge (2x+3y = 4) \wedge (x+y \leq 3)) \bmod 8$

$\exists x.((2x+z \neq 0) \wedge (2x = 5y+4) \wedge (x+y \leq 3)) \bmod 8$

$2x = 5y + 4$

$(5y+4+z \neq 0) \wedge \exists x.((2x = 5y+4) \wedge (x+y \leq 3)) \bmod 8$

- Layer1 may not eliminate quantifier

Layer 2: Drop unconstraining LMCs

$$\exists x.((2x = 5y+4) \wedge (x+y \leq 3)) \bmod 8$$

x is a bit-vector of size 3

x_2	x_1	x_0

(2x=5y+4)

x_1	x_0	0	= 5y+4

- x_2 does not affect satisfaction of $(2x = 5y+4)$

Layer 2: Drop unconstraining LMCs

$\exists x.((2x = 5y+4) \wedge (x+y \leq 3)) \bmod 8$

x is a bit-vector of size 3

x_2	x_1	x_0

(2x=5y+4)

x_1	x_0	0	= 5y+4

- x_2 does not affect satisfaction of $(2x = 5y+4)$

$(x+y \leq 3)$

x_2	x_1	x_0	$+y \leq 3$

- Can we "engineer" *every* solution of $(2x = 5y+4)$ to become a solution of $(2x = 5y+4) \wedge (x+y \leq 3)$ *by choosing x_2 appropriately*?

Layer 2: Drop unconstraining LMCs

$\exists x.((2x = 5y+4) \wedge (x+y \leq 3)) \bmod 8$

x is a bit-vector of size 3

if yes, then
$(x+y \leq 3)$ is unconstraining

$(2x=5y+4)$ 5y+4

- x_2 does not affect satisfaction of $(2x = 5y+4)$

$(x+y \leq 3)$

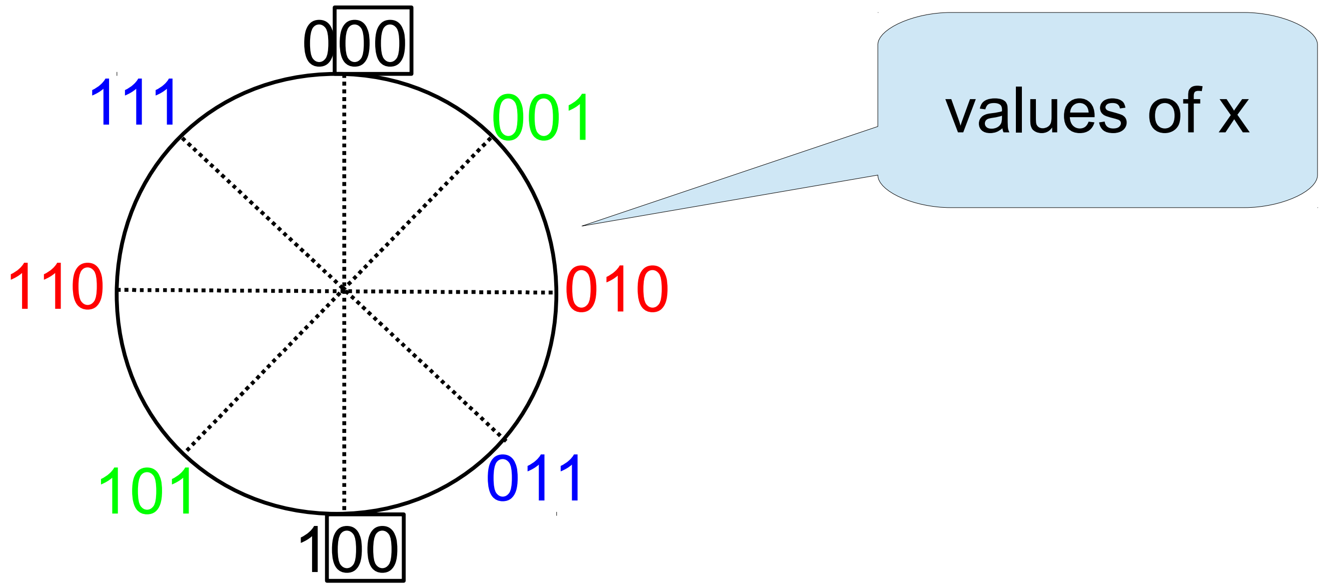
x_2	x_1	x_0	$+y \leq 3$

- Can we "engineer" *every* solution of $(2x = 5y+4)$ to become a solution of $(2x = 5y+4) \wedge (x+y \leq 3)$ *by choosing x_2 appropriately*?

Layer 2: Drop unconstraining LMCs

$$\exists x.((2x = 5y+4) \wedge (x+y \leq 3)) \bmod 8$$

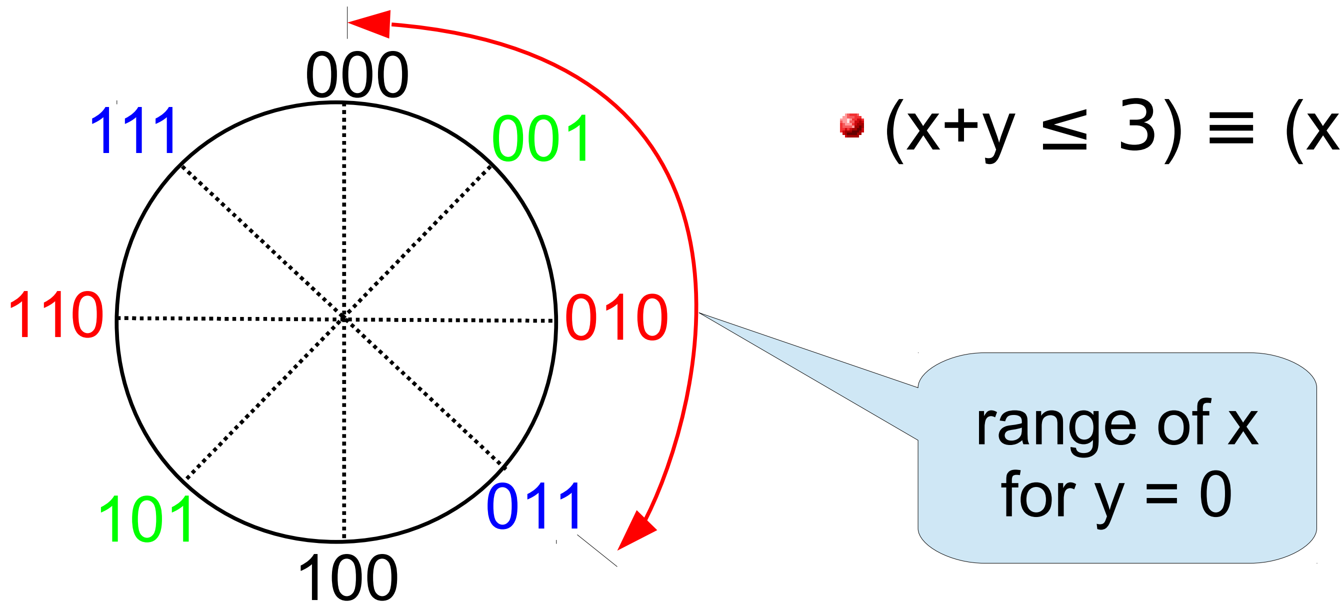
- x_2 does not affect satisfaction of $(2x = 5y+4)$



Layer 2: Drop unconstraining LMCs

$$\exists x.((2x = 5y+4) \wedge (x+y \leq 3)) \bmod 8$$

- x_2 does not affect satisfaction of $(2x = 5y+4)$

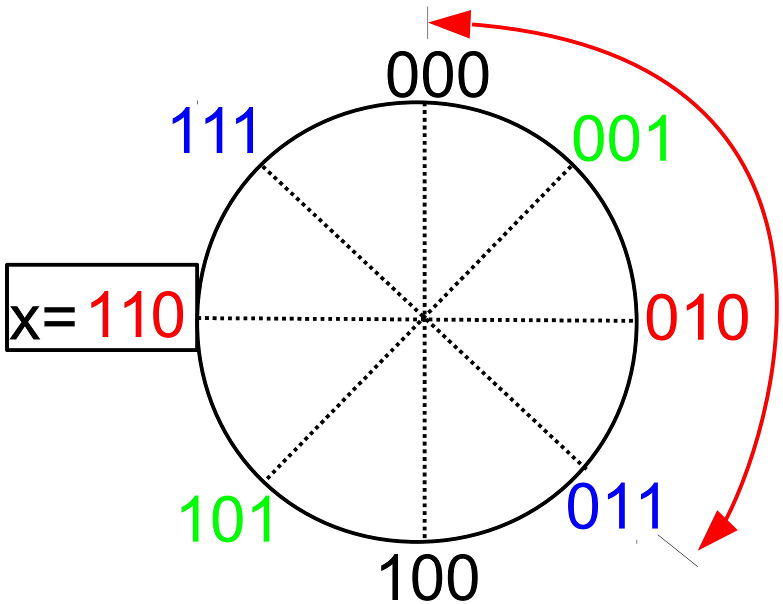


- $(x+y \leq 3) \equiv (x+y \geq 0) \wedge (x+y \leq 3)$

Layer 2: Drop unconstraining LMCs

$$\exists x.((2x = 5y+4) \wedge (x+y \leq 3)) \bmod 8$$

- x_2 does not affect satisfaction of $(2x = 5y+4)$



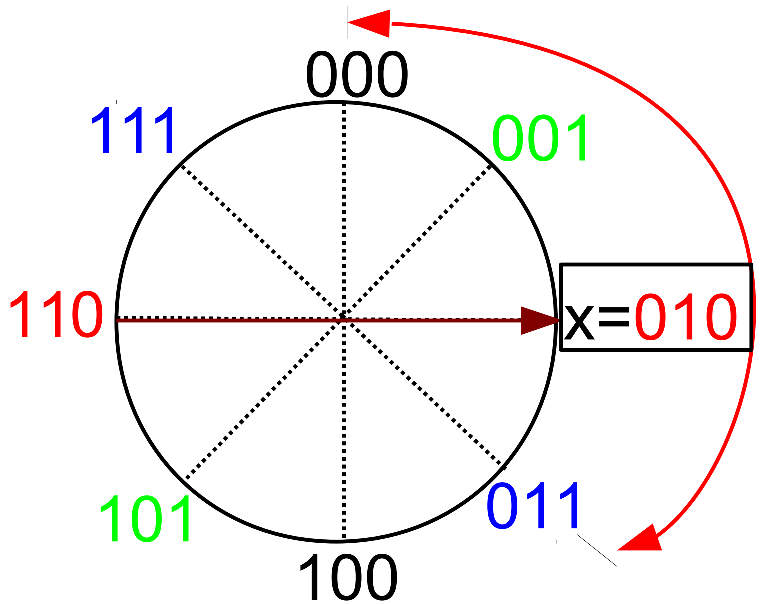
- $(x+y \leq 3) \equiv (x+y \geq 0) \wedge (x+y \leq 3)$

- $y = 0$, $x = 6$: solution of $(2x = 5y+4)$

Layer 2: Drop unconstraining LMCs

$$\exists x.((2x = 5y+4) \wedge (x+y \leq 3)) \bmod 8$$

- x_2 does not affect satisfaction of $(2x = 5y+4)$

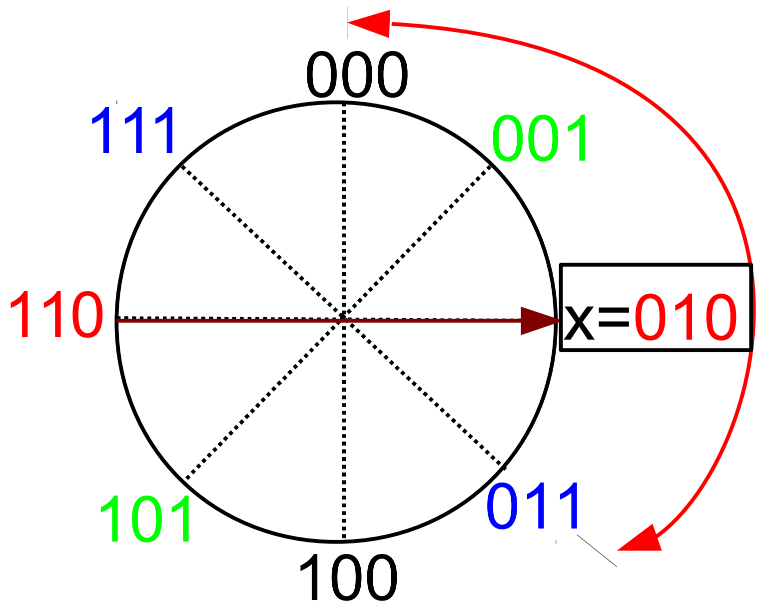


- $(x+y \leq 3) \equiv (x+y \geq 0 \wedge (x+y \leq 3)$
- $y = 0$, $x = 6$: solution of $(2x = 5y+4)$
- setting x_2=0 yields $y = 0$, $x = 2$: solution of $(2x = 5y+4) \wedge (x+y \leq 3)$

Layer 2: Drop unconstraining LMCs

$$\exists x.((2x = 5y+4) \wedge (x+y \leq 3)) \bmod 8$$

- x_2 does not affect satisfaction of $(2x = 5y+4)$



- $(x+y \leq 3) \equiv (x+y \geq 0 \wedge (x+y \leq 3)$
- $y = 0, x = 6$: solution of $(2x = 5y+4)$
- setting $x_2=0$ yields $y = 0, x = 2$: solution of $(2x = 5y+4) \wedge (x+y \leq 3)$

- Can we "engineer" *every* solution of $(2x = 5y+4)$ to become a solution of $(2x = 5y+4) \wedge (x+y \leq 3)$ *by choosing x_2 appropriately*? Yes

Layer 2: Drop unconstraining LMCs

$$\exists x.((2x = 5y+4) \wedge (x+y \leq 3)) \bmod 8$$

- *Number of ways to choose x_2* s.t. we can "engineer" *every* solution of $(2x = 5y+4)$ to become a solution of $(2x = 5y+4) \wedge (x+y \leq 3)$

- Can we "engineer" *every* solution of $(2x = 5y+4)$ to become a solution of $(2x = 5y+4) \wedge (x+y \leq 3)$ *by choosing x_2 appropriately*? Yes

Layer 2: Drop unconstraining LMCs

$$\exists x.((2x = 5y+4) \wedge (x+y \leq 3)) \bmod 8$$

- *Number of ways to choose* x_2 s.t. we can "engineer" *every* solution of $(2x = 5y+4)$ to become a solution of $(2x = 5y+4) \wedge (x+y \leq 3)$

we find efficiently computable under-approximation (η)

Layer 2: Drop unconstraining LMCs

$$\exists x.((2x = 5y+4) \wedge (x+y \leq 3)) \bmod 8$$

- *Number of ways to choose* x_2 s.t. we can "engineer" *every* solution of $(2x = 5y+4)$ to become a solution of $(2x = 5y+4) \wedge (x+y \leq 3)$

we find efficiently computable under-approximation (η)

If $\eta \geq 1$ then

$$\exists x.((2x = 5y+4)) \bmod 8 \Rightarrow \exists x.((2x = 5y+4) \wedge (x+y \leq 3)) \bmod 8$$

Layer 2: Drop unconstraining LMCs

$$\exists x.((2x = 5y+4) \wedge (x+y \leq 3)) \bmod 8$$

- *Number of ways to choose x_2* s.t. we can "engineer" *every* solution of $(2x = 5y+4)$ to become a solution of $(2x = 5y+4) \wedge (x+y \leq 3)$

we find efficiently computable under-approximation (η)

If $\eta \geq 1$ then

$$\exists x.((2x = 5y+4)) \bmod 8 \Rightarrow \exists x.((2x = 5y+4) \wedge (x+y \leq 3)) \bmod 8$$

$$\exists x.((2x = 5y+4) \wedge (x+y \leq 3)) \bmod 8 \equiv \exists x.((2x = 5y+4)) \bmod 8$$

Layer 2: Drop unconstraining LMCs

$\exists x.((2x = 5y+4) \wedge (x+y \leq 3)) \bmod 8$

- *Number of ways to choose x_2* s.t. we can "engineer" *every* solution of $(2x = 5y+4)$ to become a solution of $(2x = 5y+4) \wedge (x+y \leq 3)$

we find efficiently computable under-approximation (η)

If $\eta \geq 1$ then

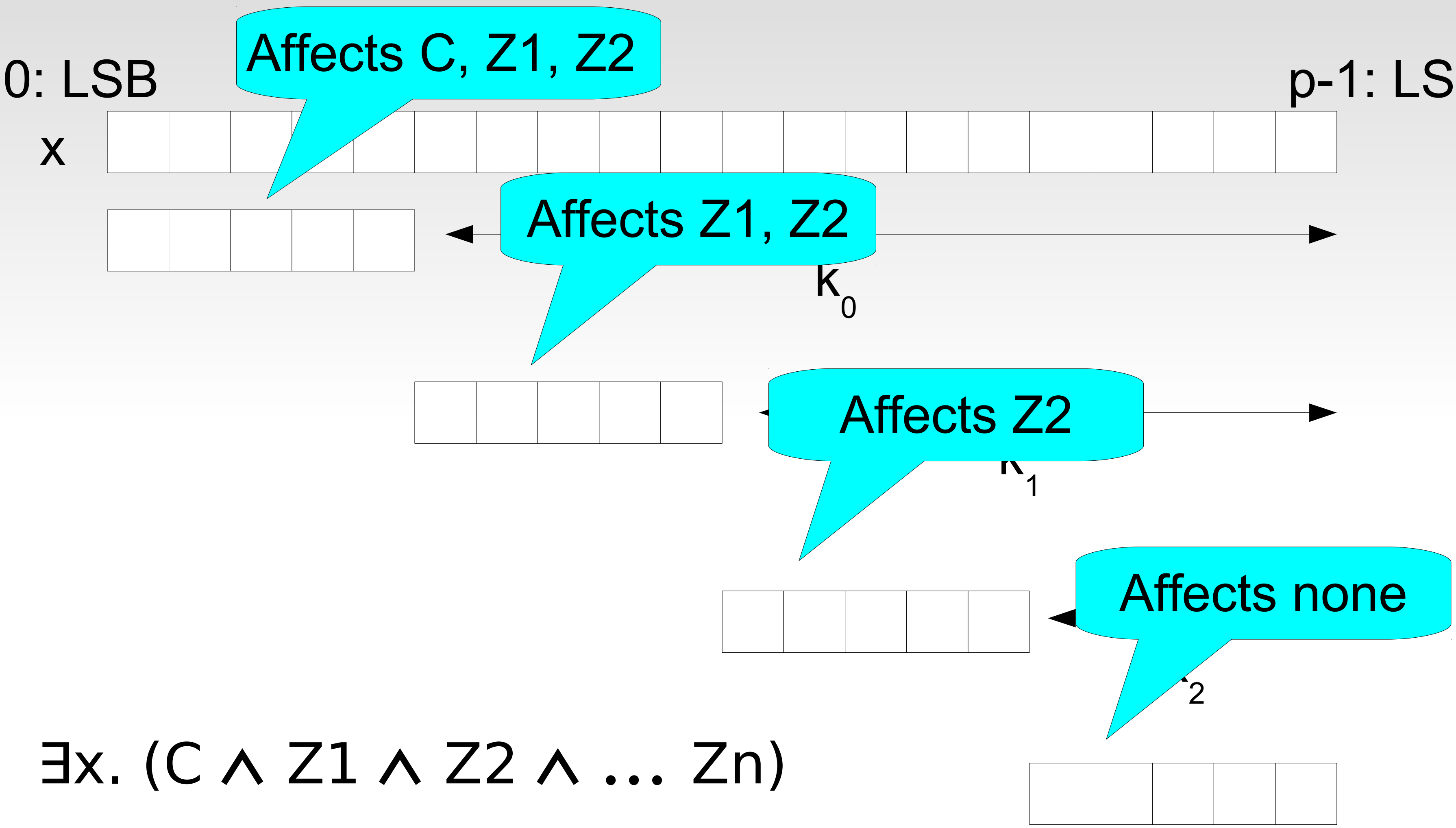
$\exists x.((2x = 5y+4)) \bmod 8 \Rightarrow \exists x.((2x = 5y+4) \wedge (x+y \leq 3)) \bmod 8$

$\exists x.((2x = 5y+4) \wedge (x+y \leq 3)) \bmod 8 \equiv \exists x.((2x = 5y+4)) \bmod 8$

$0 = (5y + 4) \bmod 2$

$\equiv (4y = 0) \bmod 8$

Layer 2: Intuition



$\exists x. (C \wedge Z1 \wedge Z2 \wedge \ldots Zn)$

Layer 2: Intuition

- Take an arbitrary solution of C
  - In how many ways can it be "engineered" to satisfy Z1 without affecting bit-slice that affects C?
- Take an arbitrary solution of $C \wedge Z1$
  - In how many ways … to satisfy Z2 without affecting bit-slices that affect C or Z1?
  - If answer > 1, then $\exists x.\ C \Rightarrow \exists x.\ (C \wedge Z1 \wedge Z2)$
- Closed form, efficiently computable, conservative formula for answer

Layer 3: Fourier-Motzkin style QE

- Fourier-Motzkin: QE from linear inequalities on reals, rationals
- Normalization:
  - Preservation of inequalities under addition and multiplication by positive terms
  - Existence of multiplicative, additive inverses

$$\exists x.((4x+4 \leq 8y) \wedge (x \geq z))$$

$$\downarrow$$

$$\exists x.((4x \leq 8y-4) \wedge (x \geq z))$$

$$\downarrow$$

$$\exists x.((x \leq 2y-1) \wedge (x \geq z))$$

Layer 3: Fourier-Motzkin style QE

- Fourier-Motzkin: QE from linear inequalities on reals

- Elimination:

$$\exists x.((2x \leq y) \wedge (2x \geq z))$$

$$\downarrow$$

$$(z \leq y)$$

Layer 3: Fourier-Motzkin style QE

- Fourier-Motzkin: QE from linear inequalities on reals

- Elimination: Density of reals

$$\exists x.((2x \leq y) \wedge (2x \geq z))$$

$$\downarrow$$

$$(z \leq y)$$

Layer 3: Fourier-Motzkin style QE

- Fourier-Motzkin: QE from linear inequalities on reals

- Normalization: Preservation of inequalities under addition and multiplication by positive terms

- Elimination: Density of reals

$$(4x+4 \leq 8y) \equiv (x \leq 2y-1)$$

$$\exists x.((2x \leq y) \wedge (2x \geq z)) \equiv (z \leq y)$$

Layer 3: Fourier-Motzkin style QE

- Fourier-Motzkin: QE from linear inequalities on reals

- Normalization: Preservation of inequalities under addition and multiplication by positive terms

- Elimination: Density of reals

$(4x+4 \leq 8y) \not\equiv (x \leq 2y-1)$

$\exists x.((2x \leq y) \wedge (2x \geq z)) \not\equiv (z \leq y)$

both fail for modular arithmetic

Need to adapt Fourier-Motzkin

Layer 3: Fourier-Motzkin style QE

- Weak normal form for LMIs: $(ax \leq t)$ and $(ax \leq bx)$

$$(4x+2 \leq y) \mod 8$$

Layer 3: Fourier-Motzkin style QE

- Weak normal form for LMIs: $(ax \leq t)$ and $(ax \leq bx)$

+6 +6

$(4x+2 \leq y) \bmod 8$

if ***overflow***$(4x+2, 6) \equiv$ ***overflow***$(y, 6)$ then $4x \leq y+6$ else $4x > y+6$

condition under which $(4x+2)+6$ overflows 3 bits

Layer 3: Fourier-Motzkin style QE

- Weak normal form for LMIs: $(ax \leq t)$ and $(ax \leq bx)$

$$\left(\overset{+6}{\downarrow}4x+2 \leq \overset{+6}{\downarrow}y\right) \bmod 8$$

$$\downarrow$$

if $\boxed{\textit{overflow}(4x+2, 6)} \equiv \boxed{\textit{overflow}(y, 6)}$ then $4x \leq y+6$ else $4x > y+6$

$$\downarrow$$

if $\boxed{4x \leq 5} \equiv \boxed{y \geq 2}$ then $4x \leq y+6$ else $4x > y+6$

$$\downarrow$$

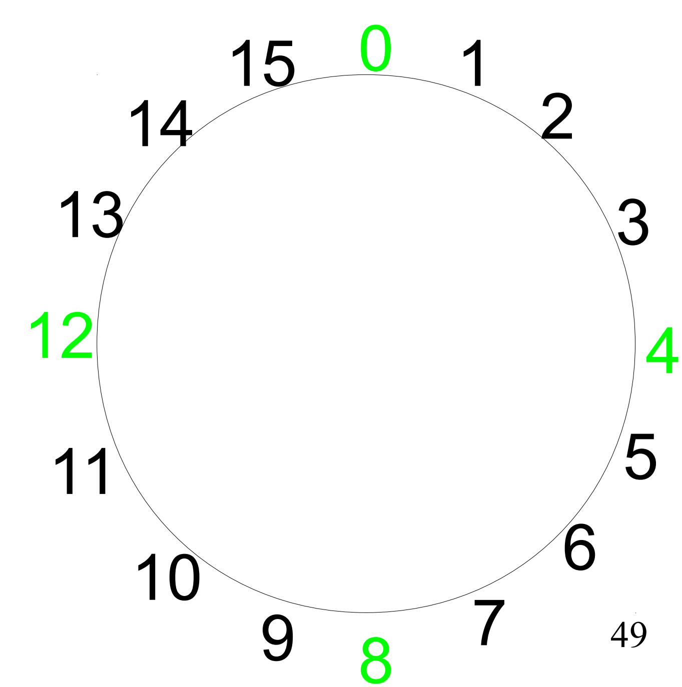
ite ($\boxed{4x \leq 5} \equiv \boxed{y \geq 2}$, $4x \leq y+6$, $4x > y+6$)

Layer 3: Fourier-Motzkin style QE

- Elimination in modular arithmetic: $\exists x.((y \leq 4x) \wedge (4x \leq z)) \bmod 16$

- Existence of multiple of 4 between y and z

- Case analysis: Disjunction of following formulas



Layer 3: Fourier-Motzkin style QE

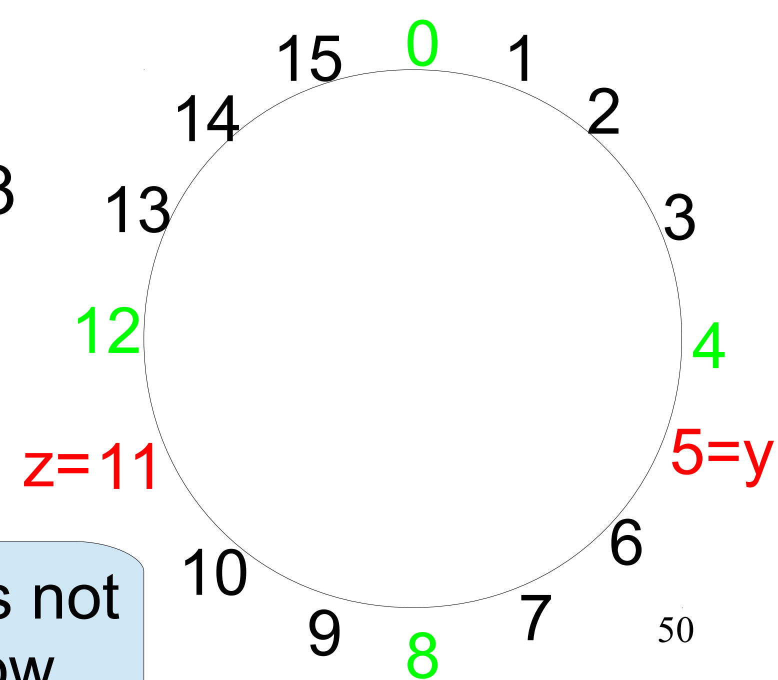
- Elimination in modular arithmetic: $\exists x.((y \leq 4x) \wedge (4x \leq z)) \bmod 16$

- Existence of multiple of 4 between y and z

- Case analysis: Disjunction of following formulas

- Case 1: (difference between y and z) ≥ 3

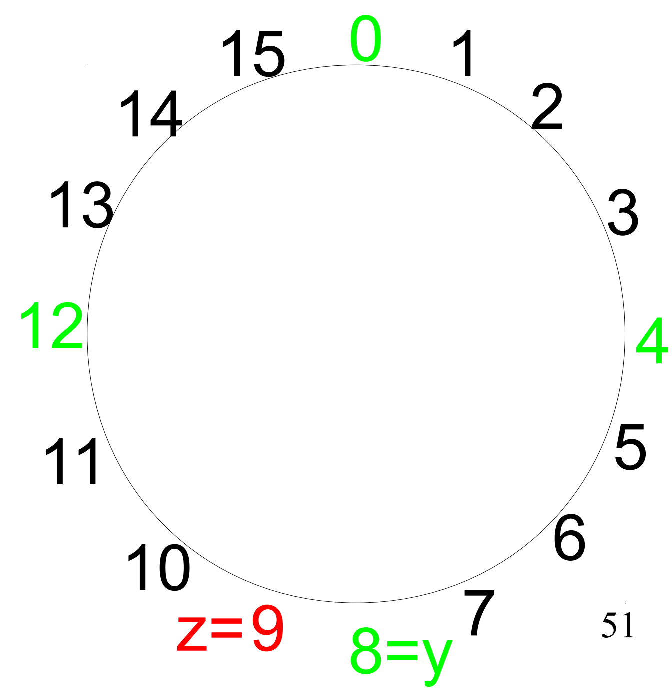
i.e. $(y \leq z) \wedge (z \geq y+3) \wedge \underbrace{(y \leq 12)}_{\text{y+3 does not overflow}}$



Layer 3: Fourier-Motzkin style QE

- Elimination in modular arithmetic: $\exists x.((y \leq 4x) \wedge (4x \leq z)) \bmod 16$
- Existence of multiple of 4 between y and z
- Case analysis: Disjunction of following formulas
- Case 2: y is a multiple of 4

i.e. $(y \leq z) \wedge (4y = 0)$



Layer 3: Fourier-Motzkin style QE

- Elimination in modular arithmetic: $\exists x.((y \leq 4x) \wedge (4x \leq z)) \bmod 16$

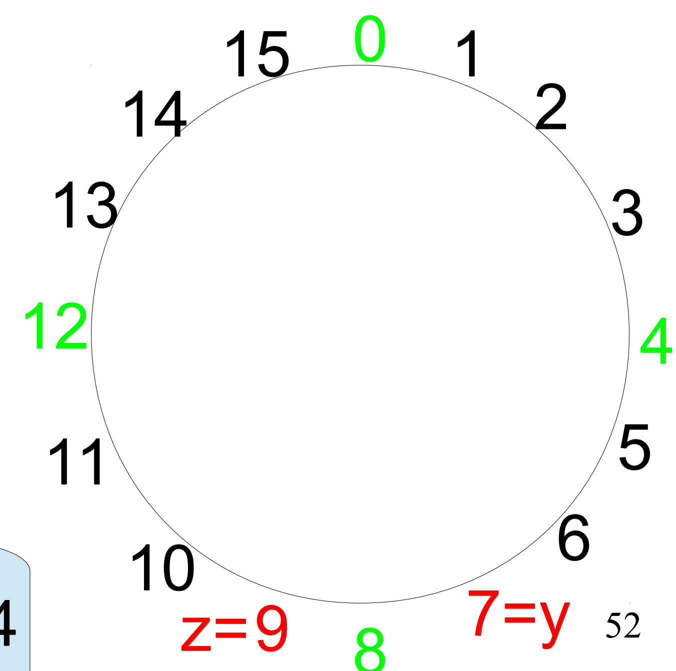
- Existence of multiple of 4 between y and z

- Case analysis: Disjunction of following formulas

- Case 3: (difference between y and z) < 3, but there exists a multiple of 4 in between

i.e. $(y \leq z) \wedge (z < y+3) \wedge (4y > 4z)$

y > z mod 4



Layer 3: Fourier-Motzkin style QE

- Elimination in modular arithmetic: $\exists x.((y \leq 4x) \wedge (4x \leq z)) \bmod 16$
- Existence of multiple of 4 between y and z
- Case analysis: Disjunction of following formulas
  - $(y \leq z) \wedge (z \geq y+3) \wedge (y \leq 12)$
  - $(y \leq z) \wedge (4y = 0)$
  - $(y \leq z) \wedge (z < y+3) \wedge (4y > 4z)$

Layer 3: Model enumeration

$$\exists x.((y \leq 2x) \wedge (3x \leq z)) \bmod 8$$

- Last resort: model enumeration

- $\bigvee_{i=0..7}[(y \leq 2x) \wedge (3x \leq z)]_{|x=i}$

Eliminating multiple quantifiers

- Eliminate each quantifier using Layer 1 to Layer 3

- Procedure to eliminate multiple quantifiers called ***Project***

QE for Boolean combinations of LMCs

- Decision diagram based approach (extending *Chaki et al., 2009, John et al. 2011*)

- SMT-solver based approach (extending *Monniaux, 2008, John et al. 2011*)

- Hybrid approach

QE using Decision Diagrams (DD)

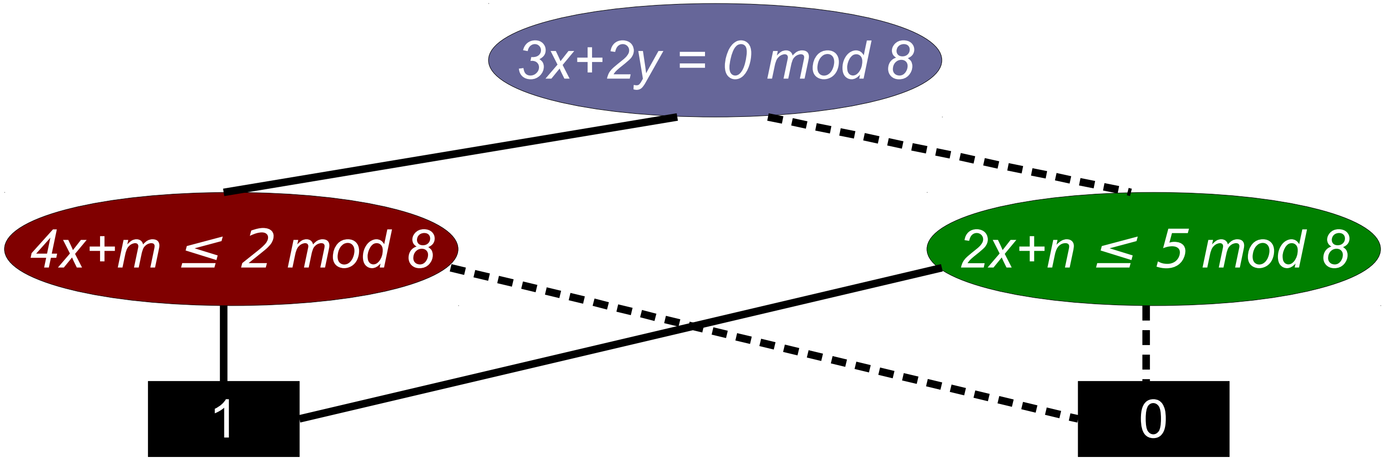
- Represents formula as a DD: BDD with nodes labeled as LMEs and LMIs

- Our procedure ***QE_LMDD*** eliminates quantifiers from DD by applying ***Project*** to each path

- <u>Simplifications</u>
  - *Eliminates single variable at a time*
  - *Simplifies the DD using the LMEs*

QE using Decision Diagrams

- To compute $\exists x.\exists y.\varphi$ ***QE_LMDD***
- Apply ***Project*** to each path
  - *Eliminate single variable at a time*
  - *Simplify the DD using the LMEs*



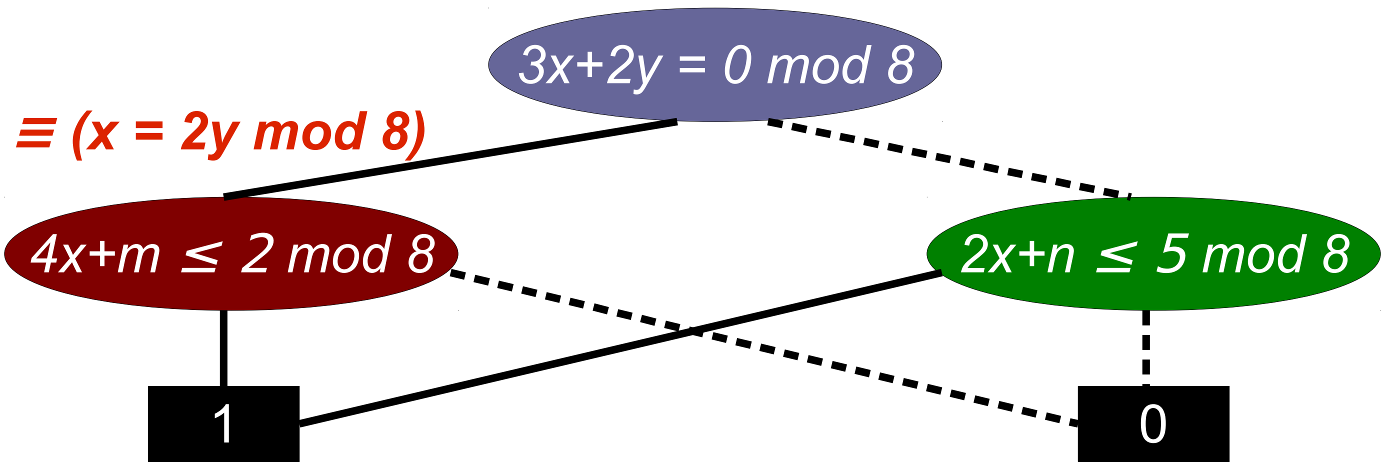
QE using Decision Diagrams

- To compute $\exists x. \exists y. \varphi$ ***QE_LMDD***

- Apply ***Project*** to each path

  - *Eliminate single variable at a time*
  - *Simplify the DD using the LMEs*

(3x+2y = 0 mod 8) ≡ (x = 2y mod 8)



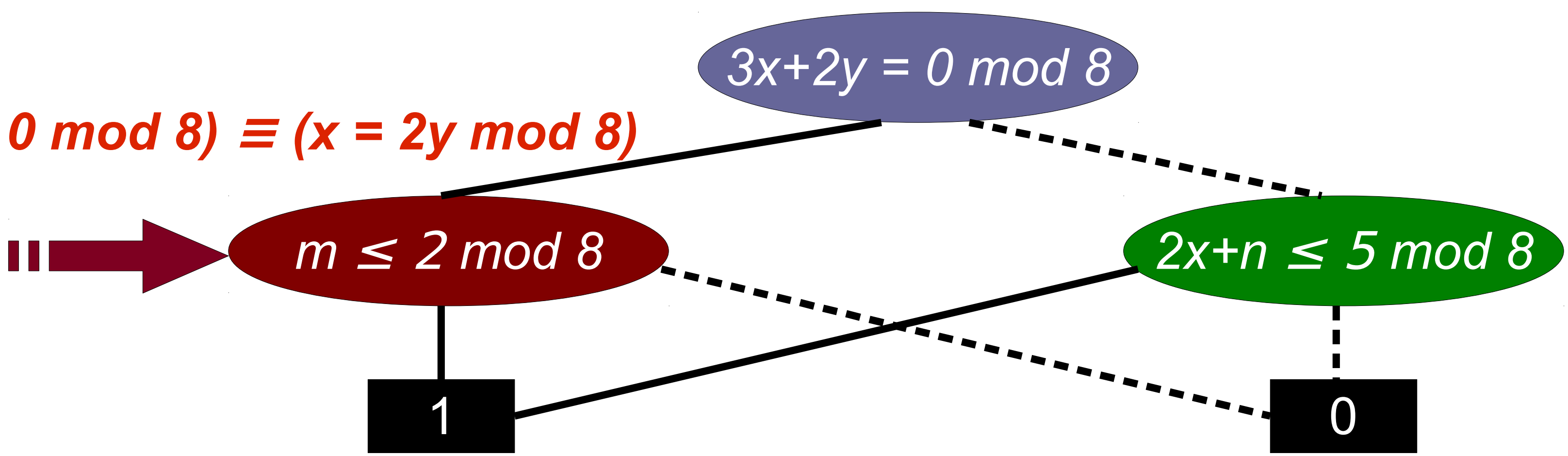
Compute DD for $\exists x. \varphi$

QE using Decision Diagrams

- To compute $\exists x.\exists y.\varphi$ ***QE_LMDD***
- Apply ***Project*** to each path
  - *Eliminate single variable at a time*
  - *Simplify the DD using the LMEs*

(3x+2y = 0 mod 8) ≡ (x = 2y mod 8)

DD becomes free of x



Compute DD for $\exists x.\varphi$

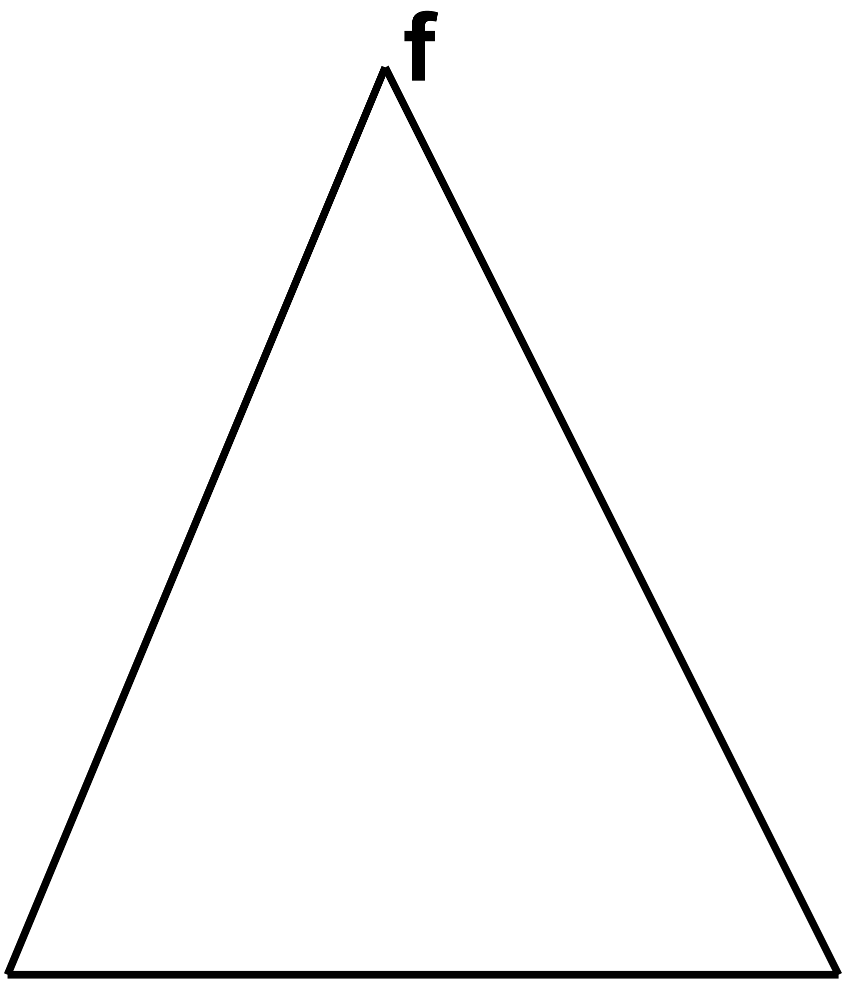
QE using Satisfiability Modulo Theories (SMT) solver

- *Monniaux et al. 2008*: Algorithm to extend Fourier-Motzkin to Boolean combinations of Linear Inequalities over Reals

- Our procedure extends Monniaux's approach
  - *Predicates are LMCs, not Linear Inequalities over Reals*
  - ***Project** in place of Fourier-Motzkin*

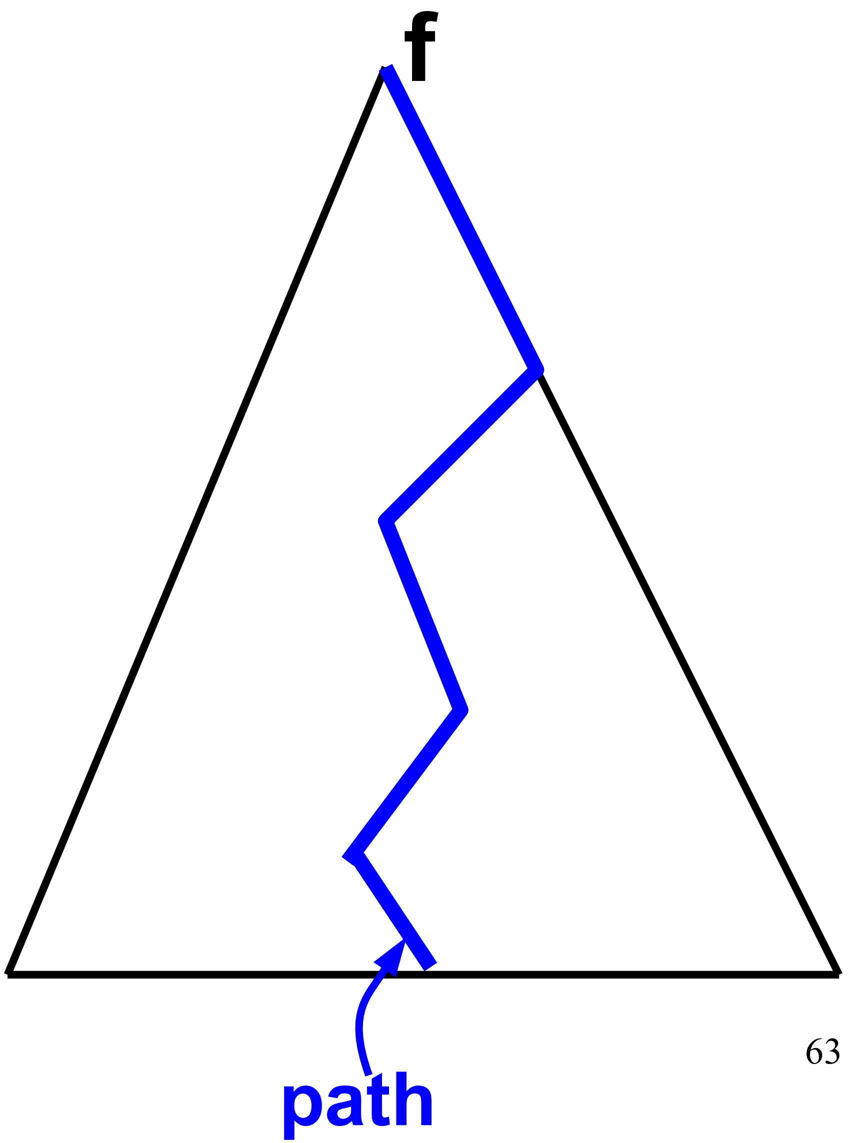
Hybrid approach for QE

- Tries to combine strengths of DD and SMT based approaches
- Given ∃X.f, where f is a DD



Hybrid approach for QE

- Tries to combine strengths of DD and SMT based approaches
- Traverse a satisfiable path in f



Hybrid approach for QE

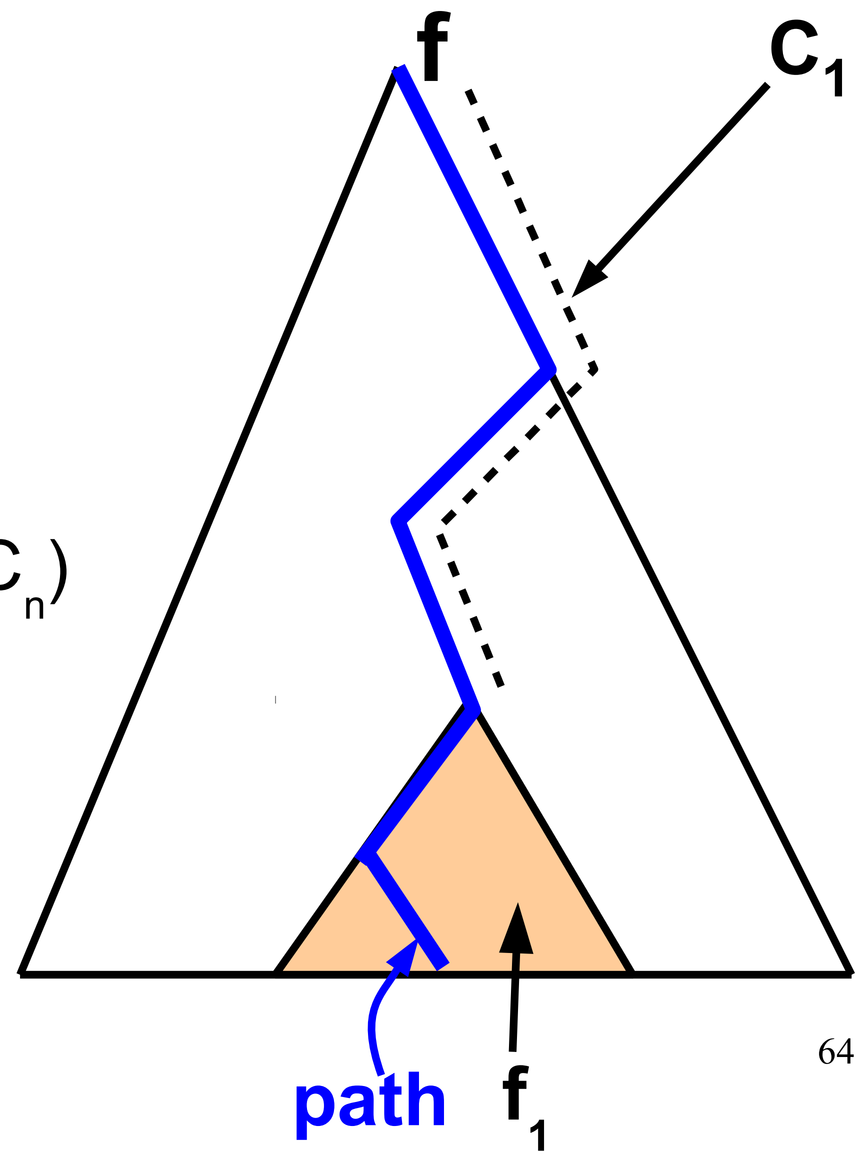
- Tries to combine strengths of DD and SMT based approaches

- Traverse a satisfiable path in f

- Convert $\exists X.f$ into a disjunction of

$\exists X.(f_1 \wedge C_1)$, $\exists X.(f_2 \wedge C_2)$, …., $\exists X.(f_n \wedge C_n)$

f_i : DD
C_i: conjunction of LMCs along the path



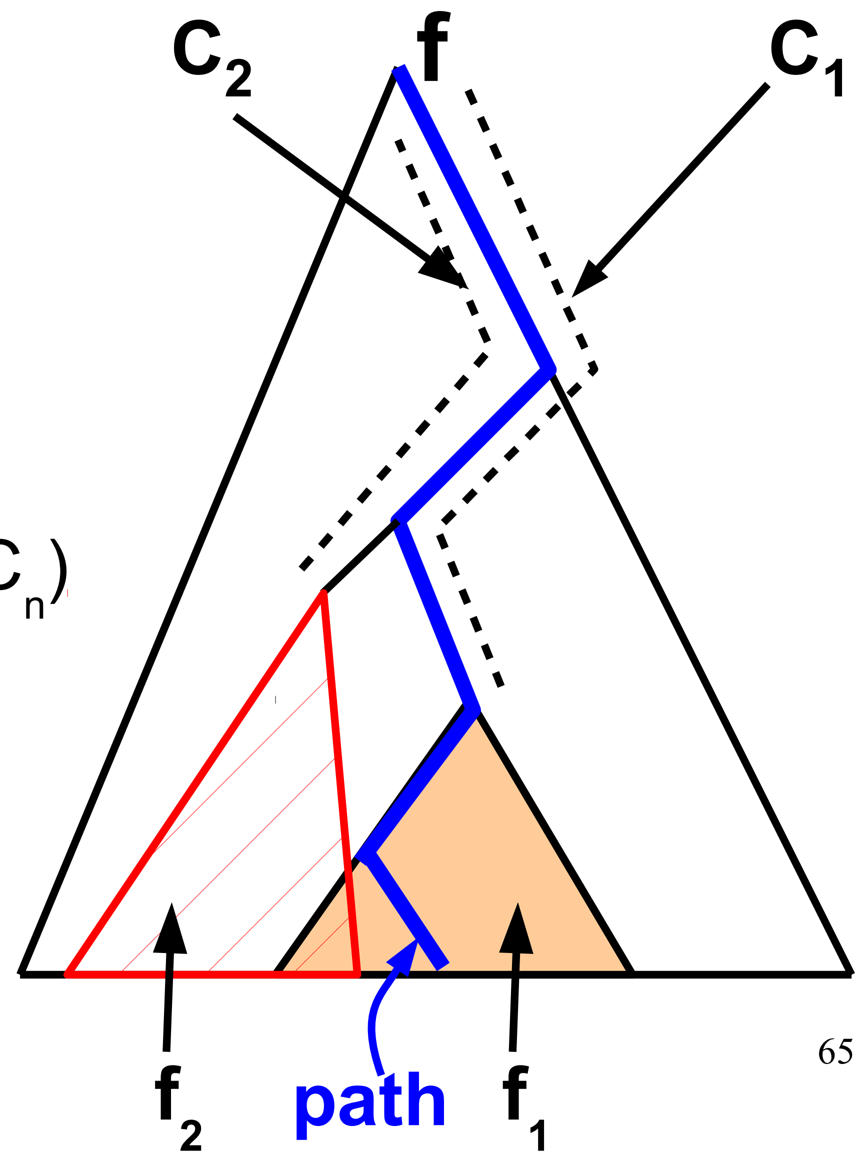
Hybrid approach for QE

- Tries to combine strengths of DD and SMT based approaches
- Traverse a satisfiable path in f
- Convert $\exists X.f$ into a disjunction of

$\exists X.(f_1 \wedge C_1)$, $\exists X.(f_2 \wedge C_2)$, …., $\exists X.(f_n \wedge C_n)$

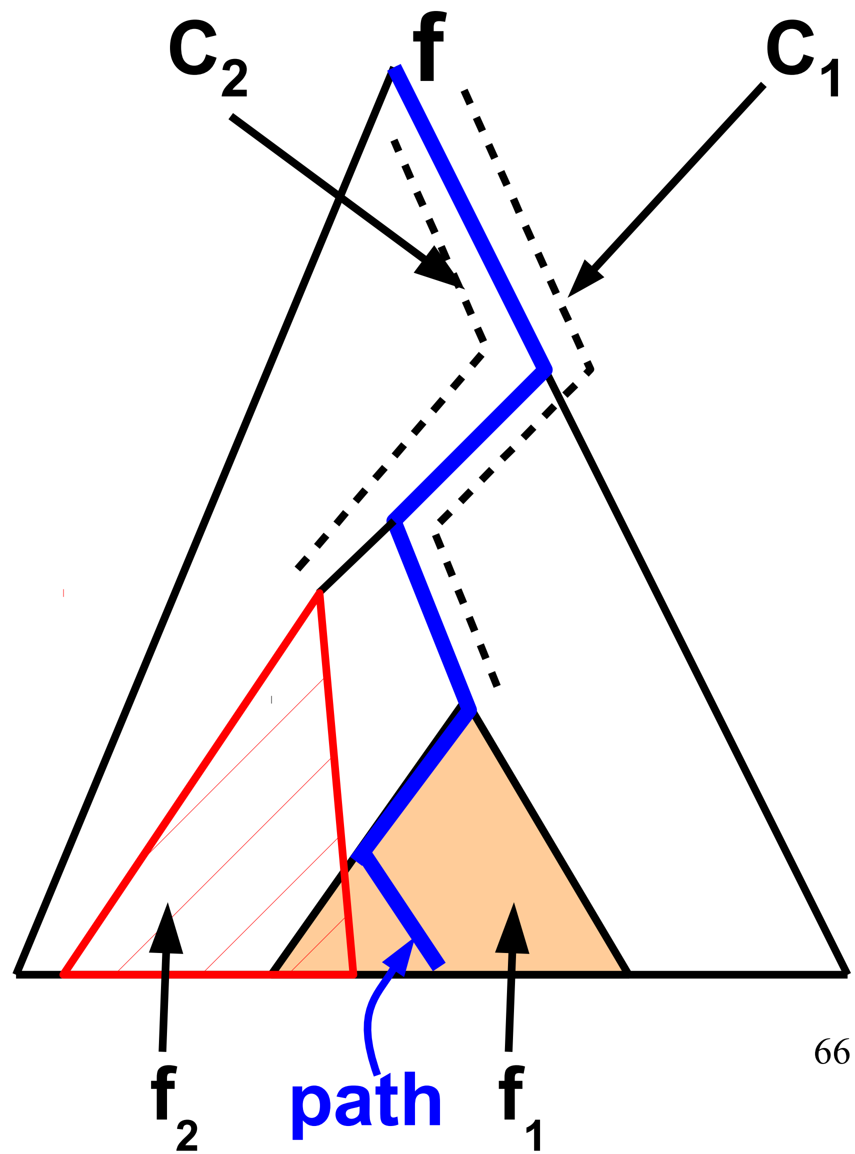
f_i: DD

C_i: conjunction of LMCs along the path



Hybrid approach for QE

- Tries to combine strengths of DD and SMT based approaches
- Traverse a satisfiable path in f
- Each $\exists X.(f_i \wedge C_i)$ computed by DD based approach
- $\vee_{i=1..n}[\exists X.(f_i \wedge C_i)]$ computed by Monniaux style loop

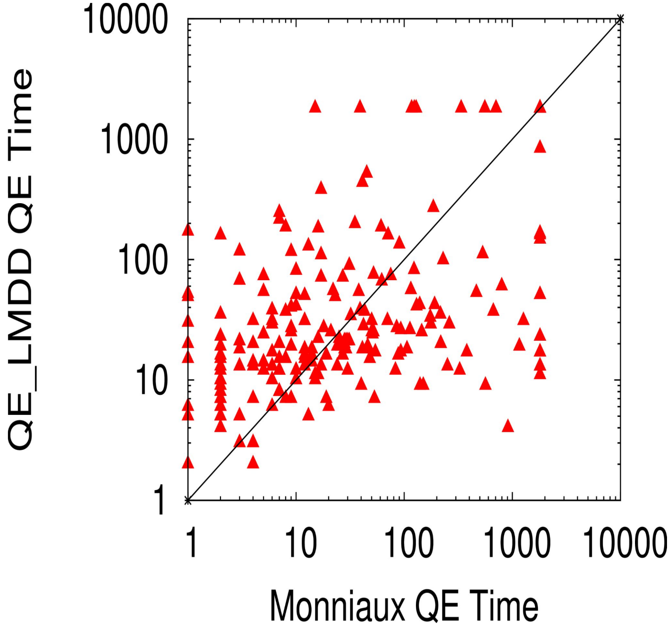


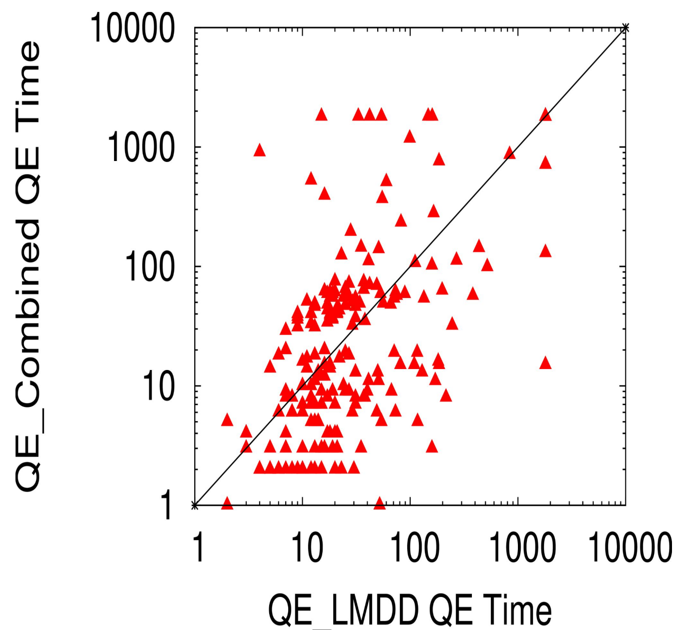
Experimental Results

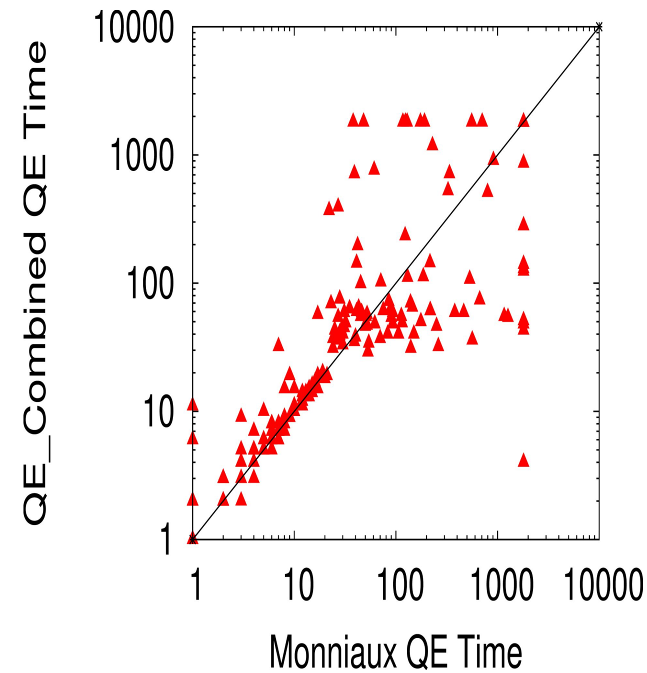
Benchmarks

- Existentially quantified Boolean combinations of LMCs
- 198 LinDD benchmarks (from *Chaki et al. 2009)*
  - *$ax+by \leq k$ over integers, a, b $\in$ {-1, 1}*
  - *Converted to LMCs assuming 16-bits for integers*
- 23 VHDL benchmarks (from transition relation abstraction)

QE_LMDD vs *Monniaux* vs *QE_Combined*







- DD and SMT based approaches incomparable
- Hybrid approach inherits strengths of both

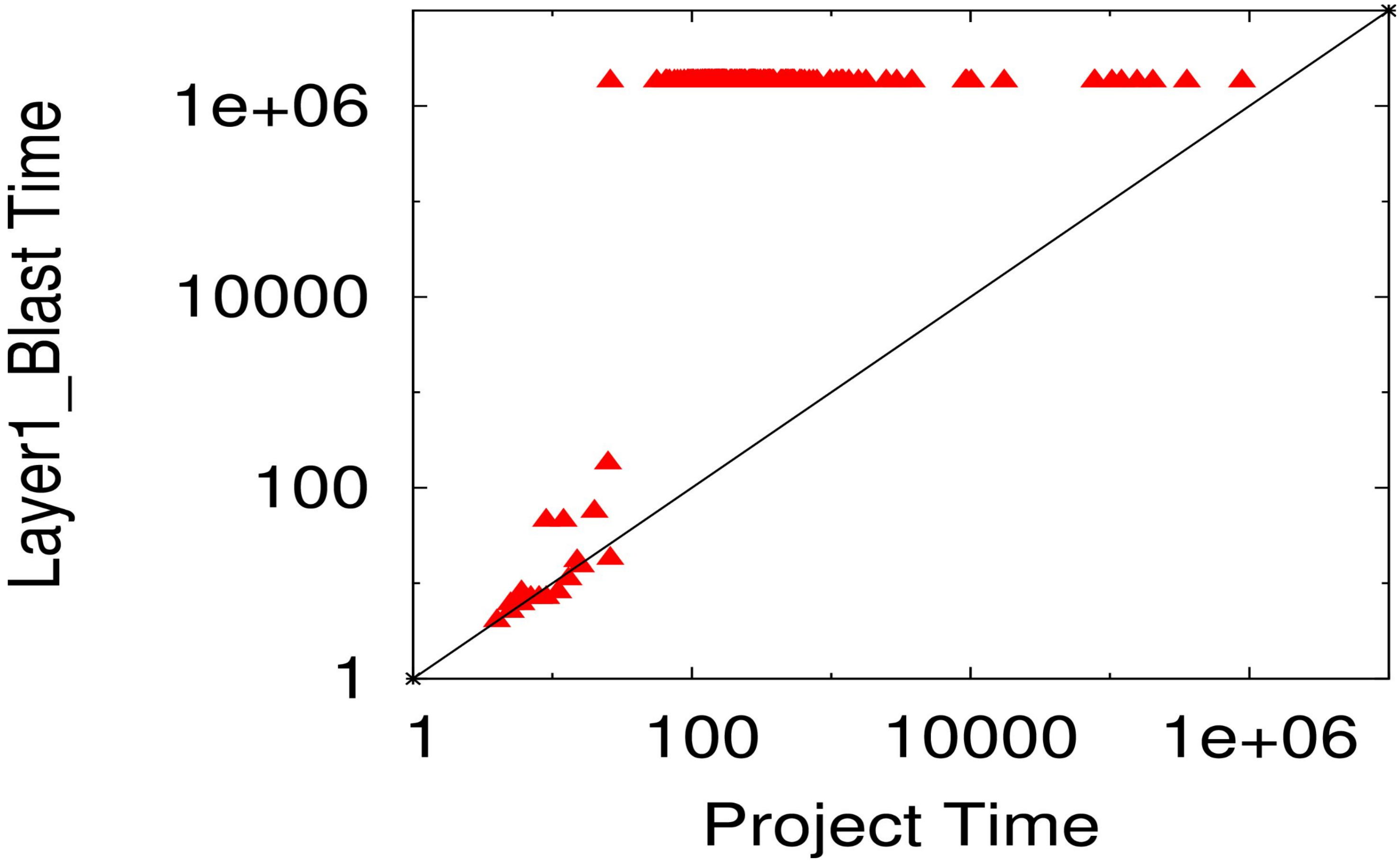
Project details

	Avg	Avg				Avg per $\exists$ eliminated			
Type	*$\exists$'s*	*LMCs*	*%Contribution*			*Time in milliseconds*			
			L1	*L2*	*L3*	*L1*	*L2*	*L3*	*Project*
LinDD	38	37	51	44	5	3	5	13149	674
VHDL	8	15	95	4.5	0.5	1	6	161	3

Layer1 and 2 more effective

Layer3 expensive than Layer 1 and 2

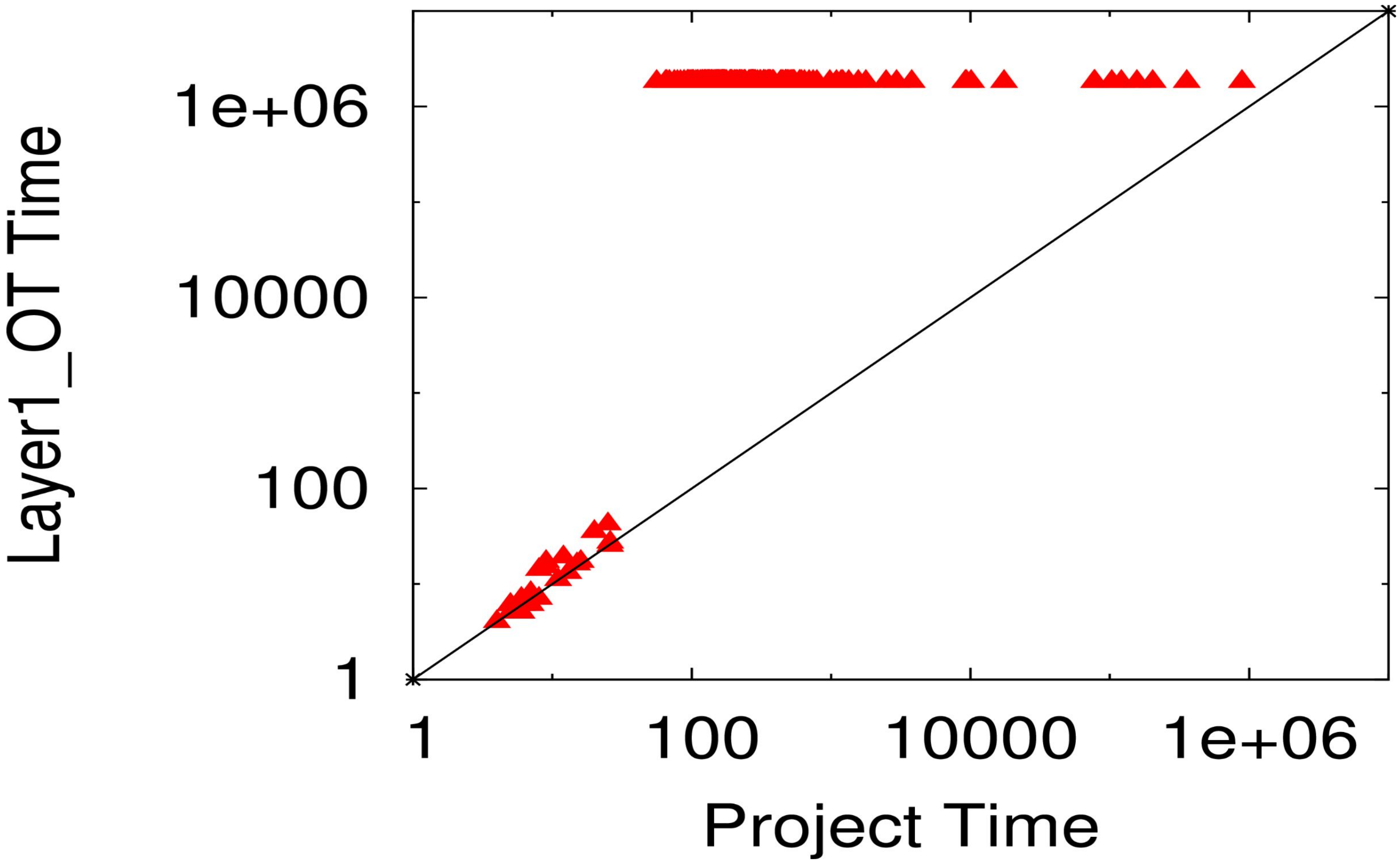
Project Vs Layer1 + Bit-level QE



- ***Project*** compared with Layer1 followed by blasting + QE using BDDs

- ***Project*** outperforms

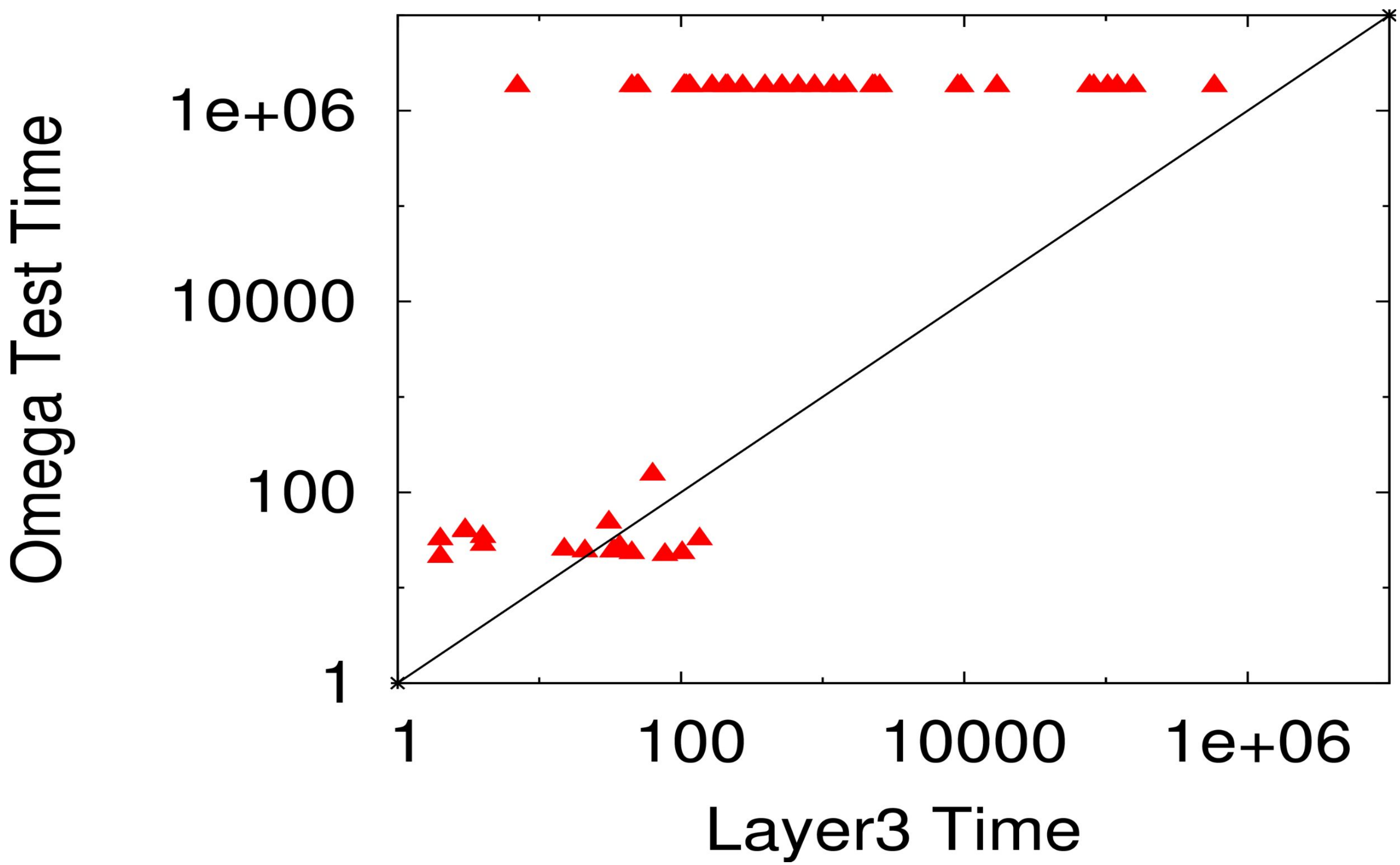
Project Vs Layer1 + Omega Test



- ***Project*** compared with Layer1 followed by conversion to ILA + QE using Omega Test

- ***Project*** outperforms

Layer3 Vs Omega Test



- Layer3 compared with conversion to ILA + QE using Omega Test
- Layer3 outperforms

Conclusions

- Modular arithmetic based techniques exist for QE from LMEs, LMDs, and LMIs
    - *Keep the final result in modular arithmetic*
    - *Outperform Integer Linear Arithmetic and bit-blasting based techniques*
- Further work needed on other non-linear constraints

Questions ?

QE using SMT-solver

Monniaux's algorithm : $\exists x_1,\ldots,\exists x_t.\varphi$

$H \leftarrow F;$

$O \leftarrow false;$

while(H is sat)

{

$a \leftarrow$ a model of H; $\{a \models H\}$

$M_1 \leftarrow$ Generalize1(φ, a); $\{M_1 \models F\}$

$M_2 \leftarrow$ Generalize2(φ, M_1); $\{M_2 \models F\}$

$\pi \leftarrow \exists x_1,\ldots,\exists x_t.\ M_2;$

$O \leftarrow O \vee \pi;$

$H \leftarrow H \wedge \neg\pi;$

}

Ensure : $O \equiv \exists x_1,\ldots,\exists x_t.\varphi$

LIR Formula

LIR Solver

Solver-based Strategy

Fourier-Motzkin

QE using SMT-solver

Monniaux: $\exists x_1,\ldots,\exists x_t.\varphi$

H ← F;
O ← false;
while(H is sat)
{
a ← a model of H; {a ⊨ H}
M_1 ← Generalize1(φ, a); {M_1 ⊨ F}
M_2 ← Generalize2(φ, M_1); {M_2 ⊨ F}
π ← $\exists x_1,\ldots,\exists x_t.$ M_2;
O ← O ∨ π;
H ← H ∧ ¬π;
}
Ensure : $O \equiv \exists x_1,\ldots,\exists x_t.\varphi$

LMC Formula

BV Solver

Evaluation Based Strategy

Project