Abstract Domains of Affine Relations

Matt Elder, Junghee Lim, Tushar Sharma,
Tycho Andersen, Thomas Reps

University of Wisconsin-Madison

Static Analysis Symposium, 2011

Goal

Efficiently find sound linear equalities in programs using machine integers

Goal

Efficiently find sound linear equalities in programs using machine integers

Machine integers = w-bit ints

Analyzing ints

Difficulties:

- Arithmetic overflow: $12 + 7 \equiv_{16} 3$
- All even numbers are zero divisors: $2 \cdot 8 \equiv_{16} 0$
- All odd numbers have inverses: $3 \cdot 11 \equiv_{16} 1$

Analyzing ints

Difficulties:

- Arithmetic overflow: $12 + 7 \equiv_{16} 3$
- All even numbers are zero divisors: $2 \cdot 8 \equiv_{16} 0$
- All odd numbers have inverses: $3 \cdot 11 \equiv_{16} 1$

Advantages:

- Can represent some bit-level properties in linear equations: $8x \equiv_{16} 8$ means x is odd
- Fast operations on native ints
- Int domains are finite lattices
- Soundness: capture real semantics

Int Domains

KS: Conjunction of affine constraints
MOS: Affine set of affine transformers

Int Domains

KS: Conjunction of affine constraints
MOS: Affine set of affine transformers

KS and MOS are **two-vocabulary** domains

Two-vocabulary domain

Definition

If the set of concrete program states is S,
a **two-vocabulary domain** abstracts relations from S to S

"Standard" domains abstract program states,
Two-vocabulary domains abstract program transitions

Also called:

- Sharir-Pneuli-style domain
- Transformer domain
- Transition domain

Question 1

How can we adapt KS to directly model w-bit ints?

Symbolic Functions

Definition

Symbolic abstraction converts logical formulas
to overapproximating domain elements

Symbolic Functions

Definition

Symbolic abstraction converts logical formulas
to overapproximating domain elements

Definition

Symbolic concretization converts domain elements
to overapproximating logical formulas

Question 2

How can we perform symbolic abstraction to MOS?

Question 3

KS and MOS:

Which is more precise?
Which is more efficient?

Outline

- How can we adapt KS to directly model w-bit ints?
- How can we perform symbolic abstraction to MOS?
- Which is more precise, KS or MOS?
- Which is more efficient, KS or MOS?

Outline

- How can we adapt KS to directly model w-bit ints?
- ~~How can we perform symbolic abstraction to MOS?~~
- Which is more precise, KS or MOS?
- Which is more efficient, KS or MOS?

How can we adapt KS to directly model w-bit ints?

KS Definition

KS element: Matrix of *w*-bit ints; each row encodes a constraint

Example

$$\begin{array}{c} \begin{array}{ccccc} x & y & x' & y' & 1 \end{array} \\ \begin{bmatrix} 5 & 7 & 9 & 12 & 6 \\ 5 & 1 & 9 & 2 & 8 \end{bmatrix} \end{array} : \qquad \begin{array}{rl} & 5x + 7y + 9x' + 12y' + 6 = 0 \\ \text{and} & 5x + y + 9x' + 2y' + 8 = 0 \end{array}$$

Generalization of: King and Søndergaard, CAV 2008

KS Definition

KS element: Matrix of *w*-bit ints; each row encodes a constraint

Example

$$\begin{array}{ccccc} x & y & x' & y' & 1 \\ \end{array}$$
$$\left[\begin{array}{ccccc} 5 & 7 & 9 & 12 & 6 \\ 5 & 1 & 9 & 2 & 8 \end{array}\right] : \quad \begin{bmatrix} 5 & 7 & 9 & 12 & 6 \\ 5 & 1 & 9 & 2 & 8 \end{bmatrix} \begin{bmatrix} x \\ y \\ x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Generalization of: King and Søndergaard, CAV 2008

KS Compose

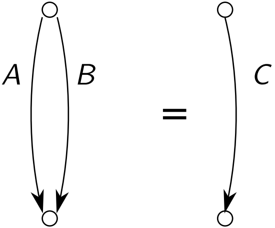
$$\begin{tikzcd} \circ \arrow[d, bend left, "{[A_{\mathrm{pre}}, A_{\mathrm{post}}, A_{\mathrm{c}}]}"] \\ \circ \arrow[d, bend left, "{[B_{\mathrm{pre}}, B_{\mathrm{post}}, B_{\mathrm{c}}]}"] \\ \circ \end{tikzcd} \quad = \quad \begin{tikzcd} \circ \arrow[dd, bend left, "C"] \\ \circ \\ \circ \end{tikzcd}$$

KS Compose

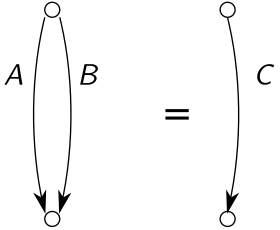
$$
\begin{tikzcd}
\circ \arrow[d, "{[A_{\mathrm{pre}}, A_{\mathrm{post}}, A_{\mathrm{c}}]}"] & & \circ \arrow[dd, bend left, "C"] \\
\circ \arrow[d, "{[B_{\mathrm{pre}}, B_{\mathrm{post}}, B_{\mathrm{c}}]}"] & = & \circ \\
\circ & & \circ
\end{tikzcd}
$$

$$
C = \textcolor{red}{\mathit{Project}}\left(\begin{bmatrix} A_{\mathrm{pre}} & \textcolor{red}{A_{\mathrm{post}}} & 0 & A_{\mathrm{c}} \\ 0 & \textcolor{red}{B_{\mathrm{pre}}} & B_{\mathrm{post}} & B_{\mathrm{c}} \end{bmatrix}\right)
$$

KS Join



KS Join



$$C = \textcolor{red}{Project}\left(\begin{bmatrix} \textcolor{red}{-A} & A \\ \textcolor{red}{B} & 0 \end{bmatrix}\right)$$

Project is the critical operation!

Naive Project

1. Move lost variables to the left
2. Do Gaussian elimination
3. Drop every row constraining a lost variable
4. Drop the lost-variable columns

Example (Project onto x and x')

$$\begin{array}{c} \begin{array}{ccccc} x & y & x' & y' & 1 \end{array} \\ \begin{bmatrix} 9 & 5 & 12 & 7 & 6 \\ 9 & 5 & 2 & 3 & 8 \end{bmatrix} \end{array}$$

Naive Project

1. Move lost variables to the left
2. Do Gaussian elimination
3. Drop every row constraining a lost variable
4. Drop the lost-variable columns

Example (Project onto x and x')

$$\begin{array}{c} \begin{array}{ccccc} x & y & x' & y' & 1 \end{array} \\ \begin{bmatrix} 9 & 5 & 12 & 7 & 6 \\ 9 & 5 & 2 & 3 & 8 \end{bmatrix} \end{array}$$

Naive Project

1. Move lost variables to the left
2. Do Gaussian elimination
3. Drop every row constraining a lost variable
4. Drop the lost-variable columns

Example (Project onto x and x')

$$
\begin{array}{c}
\begin{array}{ccccc} y & y' & x & x' & 1 \end{array} \\
\begin{bmatrix} 5 & 7 & 9 & 12 & 6 \\ 5 & 3 & 9 & 2 & 8 \end{bmatrix}
\end{array}
$$

Naive Project

1. Move lost variables to the left
2. Do Gaussian elimination
3. Drop every row constraining a lost variable
4. Drop the lost-variable columns

Example (Project onto x and x')

$$\begin{array}{c} \begin{array}{ccccc} y & y' & x & x' & 1 \end{array} \\ \begin{bmatrix} 5 & 7 & 9 & 12 & 6 \\ 5 & 3 & 9 & 2 & 8 \end{bmatrix} \end{array}$$

Naive Project

1. Move lost variables to the left
2. Do Gaussian elimination
3. Drop every row constraining a lost variable
4. Drop the lost-variable columns

Example (Project onto x and x')

$$\begin{array}{c} \begin{array}{ccccc} y & y' & x & x' & 1 \end{array} \\ \begin{bmatrix} 5 & 7 & 9 & 12 & 6 \\ 0 & 12 & 0 & 6 & 2 \end{bmatrix} \end{array}$$

Naive Project

1. Move lost variables to the left
2. Do Gaussian elimination
3. Drop every row constraining a lost variable
4. Drop the lost-variable columns

Example (Project onto x and x')

$$
\begin{array}{ccccc}
y & y' & x & x' & 1 \\
\end{array}
$$
$$
\begin{bmatrix}
5 & 7 & 9 & 12 & 6 \\
0 & 12 & 0 & 6 & 2
\end{bmatrix}
$$

Naive Project

1. Move lost variables to the left
2. Do Gaussian elimination
3. Drop every row constraining a lost variable
4. Drop the lost-variable columns

Example (Project onto x and x')

$$\top$$

Naive Project

1. Move lost variables to the left
2. Do Gaussian elimination
3. Drop every row constraining a lost variable
4. Drop the lost-variable columns

Example (Project onto x and x')

$$\top$$

Naive Project

1. Move lost variables to the left
2. Do Gaussian elimination
3. Drop every row constraining a lost variable
4. Drop the lost-variable columns

Example (Project onto x and x')

But x' must be odd!

$$\begin{array}{c} \begin{array}{ccccc} y & y' & x & x' & 1 \end{array} \\ \left[\begin{array}{ccccc} 5 & 7 & 9 & 12 & 6 \\ 0 & 12 & 0 & 6 & 2 \end{array}\right] \end{array} \Rightarrow 12y' + 6x' + 2 = 0$$

Naive Project

1. Move lost variables to the left
2. Do Gaussian elimination
3. Drop every row constraining a lost variable
4. Drop the lost-variable columns

Example (Project onto x and x')

But x' must be odd!

$$\begin{array}{c} \begin{array}{ccccc} y & y' & x & x' & 1 \end{array} \\ \left[\begin{array}{ccccc} 5 & 7 & 9 & 12 & 6 \\ 0 & 12 & 0 & 6 & 2 \end{array}\right] \end{array} \Rightarrow 12y' + 6x' + 2 \equiv_{16} 0$$

Naive Project

1. Move lost variables to the left
2. Do Gaussian elimination
3. Drop every row constraining a lost variable
4. Drop the lost-variable columns

Example (Project onto x and x')

But x' must be odd!

$$\begin{array}{c} \begin{array}{ccccc} y & y' & x & x' & 1 \end{array} \\ \left[\begin{array}{ccccc} 5 & 7 & 9 & 12 & 6 \\ 0 & 12 & 0 & 6 & 2 \end{array}\right] \end{array} \Rightarrow 48y' + 24x' + 8 \equiv_{16} 0$$

Naive Project

1. Move lost variables to the left
2. Do Gaussian elimination
3. Drop every row constraining a lost variable
4. Drop the lost-variable columns

Example (Project onto x and x')

But x' must be odd!

$$\begin{array}{c} \begin{array}{ccccc} y & y' & x & x' & 1 \end{array} \\ \left[\begin{array}{ccccc} 5 & 7 & 9 & 12 & 6 \\ 0 & \color{red}{12} & 0 & \color{red}{6} & \color{red}{2} \end{array}\right] \end{array} \Rightarrow 8x' + 8 \equiv_{16} 0$$

Naive Project

1. Move lost variables to the left
2. Do Gaussian elimination
3. Drop every row constraining a lost variable
4. Drop the lost-variable columns

Example (Project onto x and x')

But x' must be odd!

$$\begin{array}{c} \begin{array}{ccccc} y & y' & x & x' & 1 \end{array} \\ \left[\begin{array}{ccccc} 5 & 7 & 9 & 12 & 6 \\ 0 & \color{red}{12} & 0 & \color{red}{6} & \color{red}{2} \end{array}\right] \end{array} \Rightarrow 8x' \equiv_{16} 8$$

Row Space

Definition

The **row space** of a matrix is the set of linear combinations of its rows

Null Space

Definition

The **null space** of a matrix is the set of values whose product with the matrix is zero

The null space of A is $\{x \mid Ax = 0\}$

Row Operations

Definition

A **row operation** is a matrix transformation
that adds or changes individual rows
without changing the matrix's row space

Row Operations

Some Row Operations:

- Scale a row by an odd number
- Add a multiple of one row to another
- Insert some multiple of a row into the matrix

Row Operations

What if we scale a row by an even number?

Example

Scale first row by 2:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Row Operations

What if we scale a row by an even number?

Example

Scale first row by 2:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

First row space: $\begin{bmatrix} x & y & 0 \end{bmatrix}$ for any x, any y

Second row space: $\begin{bmatrix} x & y & 0 \end{bmatrix}$ for even x, any y

Row-Echelon form

Definition

A matrix is in **Row-Echelon Form** if
each row has fewer leading zeroes than the next row

Example

$$\begin{bmatrix} * & * & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & 0 & 0 & * \end{bmatrix}$$

Howell Form

Howell form is a normal form for int matrices

Howell Form

Howell form is a normal form for int matrices

Like Gaussian Elimination,
Howellization preserves the row space

Properties of Howell Form

- Normal form for row spaces
- Normal form for null spaces
- Normal form for KS

Properties of Howell Form

- Normal form for row spaces
- Normal form for null spaces
- Normal form for KS

Simplifies checking KS equality

Howell Form Definition

The matrix is in Row-Echelon form

Example

$$\begin{bmatrix} 5 & 7 & 9 & 12 & 6 \\ \color{red}{5} & 3 & 9 & 2 & 8 \end{bmatrix}$$

Howell Form Definition

The matrix is in Row-Echelon form

Example

$$\begin{bmatrix} 5 & 7 & 9 & 12 & 6 \\ 0 & 12 & 0 & 6 & 2 \end{bmatrix}$$

Howell Form Definition

Leading values are powers of 2

Example

$$\begin{bmatrix} 5 & 7 & 9 & 12 & 6 \\ 0 & 12 & 0 & 6 & 2 \end{bmatrix}$$

Howell Form Definition

Leading values are powers of 2

Example

$$\begin{bmatrix} 1 & 11 & 5 & 12 & 14 \\ 0 & 4 & 0 & 2 & 6 \end{bmatrix}$$

Howell Form Definition

Leading values are largest in their columns

Example

$$\begin{bmatrix} 1 & \color{red}{11} & 5 & 12 & 14 \\ 0 & 4 & 0 & 2 & 6 \end{bmatrix}$$

Howell Form Definition

Leading values are largest in their columns

Example

$$\begin{bmatrix} 1 & 3 & 5 & 8 & 2 \\ 0 & 4 & 0 & 2 & 6 \end{bmatrix}$$

Howell Form Definition

Every consequence of every row
is a linear combination of the matrix rows
that have at least as many leading zeros as the consequence

Howell Form Definition

Every consequence of every row
is a linear combination of the matrix rows
that have at least as many leading zeros as the consequence

Definition

The vectors $2^k v$ are the **consequences** of v

Example

The consequences of $\begin{bmatrix} 0 & 4 & 0 & 2 & 6 \end{bmatrix}$ are
$\left\{ \begin{bmatrix} 0 & 8 & 0 & 4 & 4 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 8 & 8 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix} \right\}$

Howell Form Definition

Every consequence of every row
is a linear combination of the matrix rows
that have at least as many leading zeros as the consequence

Example

$$\begin{bmatrix} 1 & 3 & 5 & 8 & 2 \\ 0 & 4 & 0 & 2 & 6 \end{bmatrix}$$

Howell Form Definition

Every consequence of every row
is a linear combination of the matrix rows
that have at least as many leading zeros as the consequence

Example

$$\begin{bmatrix} 1 & 3 & 5 & 8 & 2 \\ \color{red}{0} & \color{red}{4} & \color{red}{0} & \color{red}{2} & \color{red}{6} \end{bmatrix}$$

$$\begin{bmatrix} 0 & 8 & 0 & 4 & 4 \end{bmatrix}$$

Howell Form Definition

Every consequence of every row
is a linear combination of the matrix rows
that have at least as many leading zeros as the consequence

Example

$$\begin{bmatrix} 1 & 3 & 5 & 8 & 2 \\ \color{red}{0} & \color{red}{4} & \color{red}{0} & \color{red}{2} & \color{red}{6} \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 8 & 8 \end{bmatrix}$$

Howell Form Definition

Every consequence of every row
is a linear combination of the matrix rows
that have at least as many leading zeros as the consequence

Example

$$\begin{bmatrix} 1 & 3 & 5 & 8 & 2 \\ 0 & 4 & 0 & 2 & 6 \\ 0 & 0 & 0 & 8 & 8 \end{bmatrix}$$

Howell Form Definition

Every consequence of every row
is a linear combination of the matrix rows
that have at least as many leading zeros as the consequence

Example

$$\begin{bmatrix} 1 & 3 & 5 & \color{red}{8} & 2 \\ 0 & 4 & 0 & 2 & 6 \\ 0 & 0 & 0 & 8 & 8 \end{bmatrix}$$

Howell Form Definition

Every consequence of every row
is a linear combination of the matrix rows
that have at least as many leading zeros as the consequence

Example

$$\begin{bmatrix} 1 & 3 & 5 & 0 & 10 \\ 0 & 4 & 0 & 2 & 6 \\ 0 & 0 & 0 & 8 & 8 \end{bmatrix}$$

Precise Projection in KS

1. Move lost variables to the left
2. Howellize the matrix
3. Drop every row constraining a lost variable
4. Drop the lost-variable columns

Example (Project onto x and x')

$$\begin{array}{c} \begin{array}{ccccc} x & y & x' & y' & 1 \end{array} \\ \begin{bmatrix} 9 & 5 & 12 & 7 & 6 \\ 9 & 5 & 2 & 3 & 8 \end{bmatrix} \end{array}$$

Precise Projection in KS

1. Move lost variables to the left
2. Howellize the matrix
3. Drop every row constraining a lost variable
4. Drop the lost-variable columns

Example (Project onto x and x')

$$\begin{array}{c} \begin{array}{ccccc} x & y & x' & y' & 1 \end{array} \\ \begin{bmatrix} 9 & 5 & 12 & 7 & 6 \\ 9 & 5 & 2 & 3 & 8 \end{bmatrix} \end{array}$$

Precise Projection in KS

1. Move lost variables to the left
2. Howellize the matrix
3. Drop every row constraining a lost variable
4. Drop the lost-variable columns

Example (Project onto x and x')

$$\begin{array}{c} \begin{array}{ccccc} \color{red}{y} & \color{red}{y'} & x & x' & 1 \end{array} \\ \begin{bmatrix} 5 & 7 & 9 & 12 & 6 \\ 5 & 3 & 9 & 2 & 8 \end{bmatrix} \end{array}$$

Precise Projection in KS

1. Move lost variables to the left
2. Howellize the matrix
3. Drop every row constraining a lost variable
4. Drop the lost-variable columns

Example (Project onto x and x')

$$\begin{array}{c} \begin{array}{ccccc} y & y' & x & x' & 1 \end{array} \\ \begin{bmatrix} 5 & 7 & 9 & 12 & 6 \\ 5 & 3 & 9 & 2 & 8 \end{bmatrix} \end{array}$$

Precise Projection in KS

1. Move lost variables to the left
2. Howellize the matrix
3. Drop every row constraining a lost variable
4. Drop the lost-variable columns

Example (Project onto x and x')

$$\begin{array}{c} \begin{array}{ccccc} y & y' & x & x' & 1 \end{array} \\ \begin{bmatrix} 1 & 3 & 5 & 0 & 10 \\ 0 & 4 & 0 & 2 & 6 \\ 0 & 0 & 0 & 8 & 8 \end{bmatrix} \end{array}$$

Precise Projection in KS

1. Move lost variables to the left
2. Howellize the matrix
3. Drop every row constraining a lost variable
4. Drop the lost-variable columns

Example (Project onto x and x')

$$\begin{array}{c} \begin{array}{ccccc} y & y' & x & x' & 1 \end{array} \\ \begin{bmatrix} 1 & 3 & 5 & 0 & 10 \\ 0 & 4 & 0 & 2 & 6 \\ 0 & 0 & 0 & 8 & 8 \end{bmatrix} \end{array}$$

Precise Projection in KS

1. Move lost variables to the left
2. Howellize the matrix
3. Drop every row constraining a lost variable
4. Drop the lost-variable columns

Example (Project onto x and x')

$$\begin{array}{c} \begin{array}{ccccc} y & y' & x & x' & 1 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 8 & 8 \end{bmatrix} \end{array}$$

Precise Projection in KS

1. Move lost variables to the left
2. Howellize the matrix
3. Drop every row constraining a lost variable
4. Drop the lost-variable columns

Example (Project onto x and x')

$$\begin{array}{ccccc} y & y' & x & x' & 1 \\ \end{array}$$
$$\begin{bmatrix} 0 & 0 & 0 & 8 & 8 \end{bmatrix}$$

Precise Projection in KS

1. Move lost variables to the left
2. Howellize the matrix
3. Drop every row constraining a lost variable
4. Drop the lost-variable columns

Example (Project onto x and x')

$$\begin{array}{c} \begin{array}{ccc} x & x' & 1 \end{array} \\ \begin{bmatrix} 0 & 8 & 8 \end{bmatrix} \end{array}$$

How can we adapt KS to directly model *w*-bit ints?

Use Howell form for projection!

Which is more precise, KS or MOS?

MOS Definition

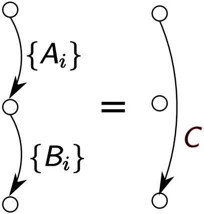
MOS element: a set of matrices of w-bit ints;
every affine combination those matrices
may transform the initial state

Example

$$\left\{ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 6 \\ 0 & 0 & 1 \end{bmatrix} \right\} : \quad \exists k : \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2k \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

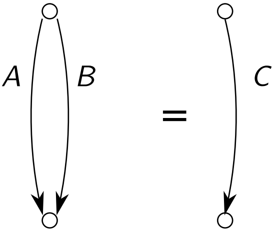
See: Müller-Olm and Seidl, TOPLAS 2007

MOS Compose



$$C = Basis\{B_j A_i\}$$

MOS Join



$$C = \mathit{Basis}\{A \cup B\}$$

Basis via Howellize

Can use Howellize for the *Basis* function

Basis via Howellize

Can use Howellize for the *Basis* function

This allows easy equality checking!

Non-Affine Constraints

MOS can represent non-affine constraints!

Non-Affine Constraints

MOS can represent non-affine constraints!

Example

$$\left\{ \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right\} : \quad \exists k : \begin{bmatrix} k & 1-k & 0 \\ k & 1-k & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

Non-Affine Constraints

MOS can represent non-affine constraints!

Example

$$\left\{ \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right\} : \quad \exists k : x' = y' = y + k(x - y)$$

Assumes

Example

$[\![assume(x = 5)]\!] : x = x' \wedge y = y' \wedge x = 5$

Assumes

Example

$[\![assume(x = 5)]\!] : x = x' \wedge y = y' \wedge x = 5$

One of the best MOS transformers is $\left\{ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right\}$

Assumes

Example

$[\![assume(x = 5)]\!] : x = x' \wedge y = y' \wedge x = 5$

One of the best MOS transformers is $\left\{ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right\}$

MOS cannot represent assumes!

Which is more precise, KS or MOS?

KS and MOS are incomparable

Which is more efficient, KS or MOS?

Naive Algorithms

For *k* variables:

	KS	MOS
Element size	$O(k^2)$	$O(k^4)$
Join	$O(k^3)$	$O(k^6)$
Compose	$O(k^3)$	$O(k^7)$

Fast Algorithms

If ring matrix multiplication is $O(k^\alpha)$, then:

	KS	MOS
Element size	$O(k^2)$	$O(k^4)$
Join	$O(k^\alpha)$	$O(k^{2\alpha})$
Compose	$O(k^\alpha)$	$O(k^{4+\alpha})$

Experimental Setup

For eight small programs (500-4000 instructions):

1. Compute MOS and KS elements on program edges
2. Perform two-phase queries at the beginning of basic blocks that end in branches
3. Compare MOS and KS precision at each query point

Experimental Setup

Symbolic abstraction to build KS elements
Operator reinterpretation to build MOS elements

Experimental Setup

Symbolic abstraction to build KS elements
Operator reinterpretation to build MOS elements

SMT is used only in devising initial KS elements
Not in KS's analysis phases; nowhere in MOS

Experimental Results: Precision

KS (with symbolic abstraction) was at least as precise
as MOS (with operator reinterpretation) at **every** query point

Experimental Results: Precision

KS (with symbolic abstraction) was at least as precise
as MOS (with operator reinterpretation) at **every** query point

If this holds for KS with operator reinterpretation,
then MOS's non-affine constraints don't help for real programs

Experimental Results: Construction Time

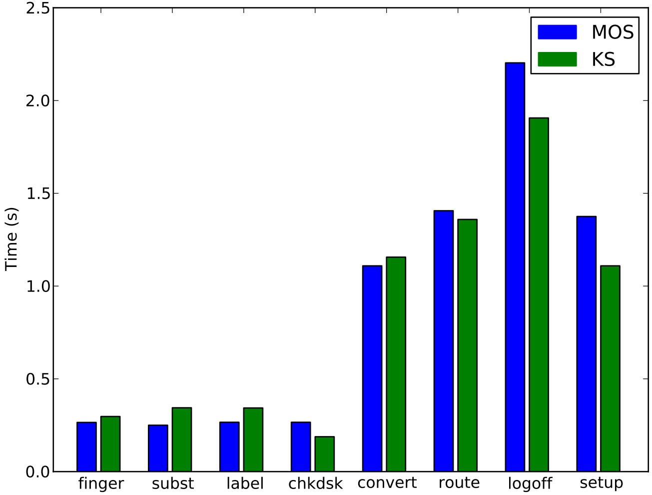
Constructing KS via symbolic abstraction took 325 times longer than constructing MOS via operator reinterpretation

Experimental Results: Construction Time

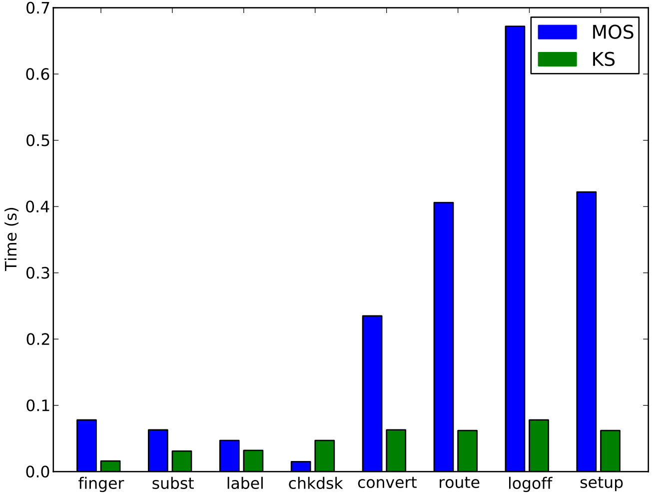
Constructing KS via symbolic abstraction took 325 times longer than constructing MOS via operator reinterpretation

Constructing KS via operator reinterpretation: coming soon

Experimental Results: Phase 1 Time



Experimental Results: Phase 2 Time



Experimental Conclusions

Overall, KS analysis time was 91% of MOS analysis time
Phase 1 time, KS/MOS: 94%
Phase 2 time, KS/MOS: 20%

Seems that KS analysis is somewhat
faster than MOS on real inputs

Technical Highlights

- Howell form allows projection in KS
- Howell form is a normal form for KS and MOS
- MOS can capture non-affine constraints

Conclusions

- KS for *w*-bit ints:
  - Needs no bit blasting
  - Now applies to larger programs
- KS and MOS are mathematically incomparable
- KS analysis is more efficient than MOS,
  in theory and (provisionally) in practice