A New Abstraction Framework for Affine Transformers

THE UNIVERSITY of WISCONSIN MADISON

Tushar Sharma and Thomas Reps

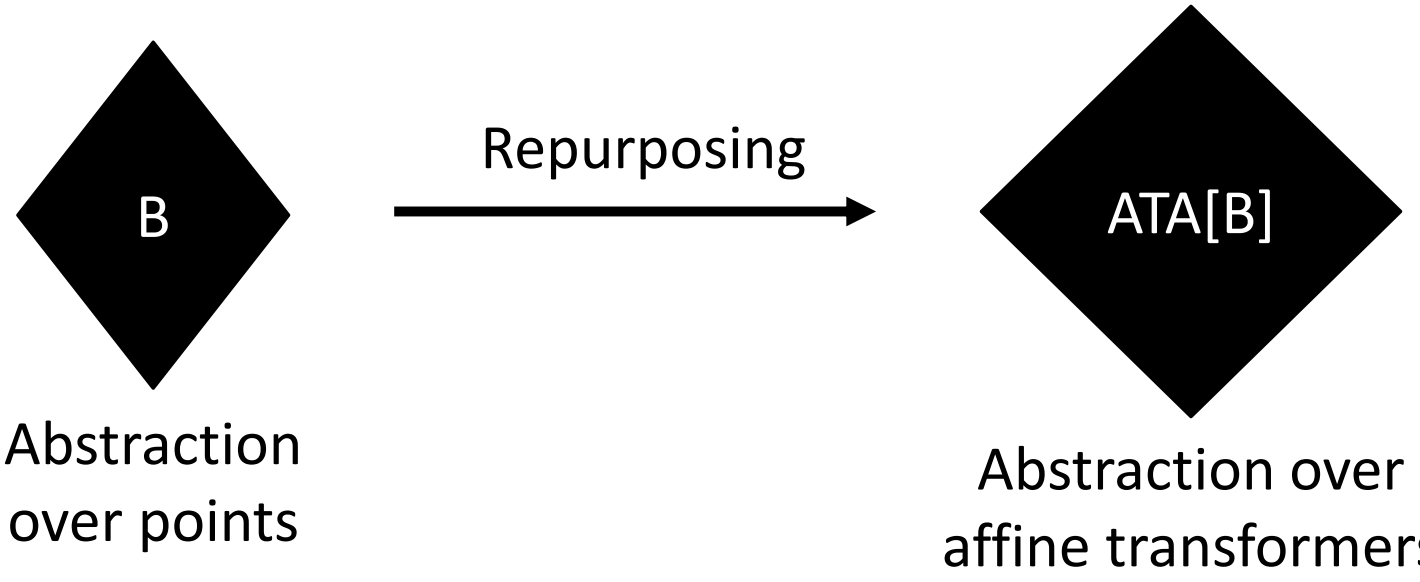
SAS'17

Motivations

- Prove Program Assertions
- Function and loop summaries
- Sound with respect to *bitvectors*

Affine Transformers Abstraction (ATA)

- Affine Transformer Abstraction Framework: ATA[B]
- Family of abstract domains
- Parametrized over a base domain ‘B’ for bitvectors



Affine Transformers Abstraction (ATA)

- New Abstract Domains not discussed previously in literature
- Can express interesting class of disjunctions over affine transformers over bitvectors:
  - *E.g.:* Interval Affine Maps

    $v_1' = [1,7]\, v_1 + [0,2]\, v_2 + [3,4]$

    v_j and v_j' represent pre-transformation and post-transformation variables, respectively.
  - *E.g.:* Octagon Constrained Affine Maps

    $v_1' = i_1 \,.\, v_1 + i_2 \,.\, v_2,\ 0 \leq i_1 + i_2 \leq 5$

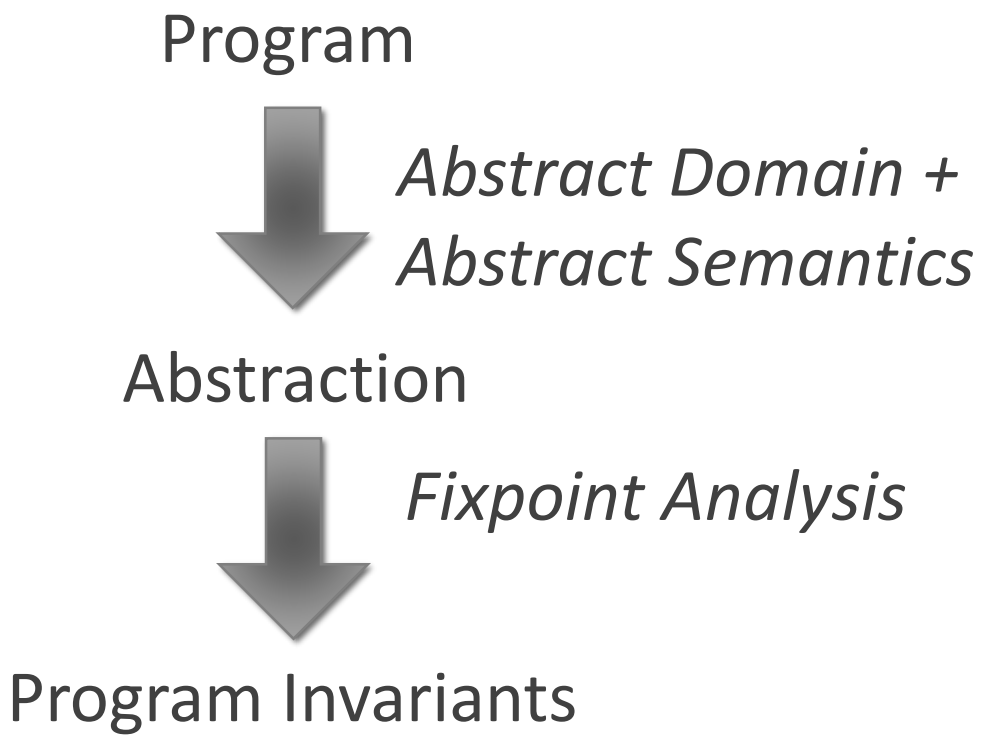
Affine Transformer

- Affine Transformer: $\vec{v}' = \vec{v} \cdot C + \vec{d}$
  - $[v_1' \quad v_2'] = [v_1 \quad v_2]\begin{bmatrix} 1 & 0 \\ 2 & 0 \end{bmatrix} + [10 \quad 0]$, represents ($v_1'$ =
- $T = \begin{bmatrix} 1 & \vec{d} \\ 0 & C \end{bmatrix}$, $[1 \quad \vec{v}'] = [1 \quad \vec{v}] \cdot T$
  - Example: $[1 \quad v_1' \quad v_2'] = [1 \quad v_1 \quad v_2]\left[\begin{array}{c|cc} 1 & 10 & 0 \\ \hline 0 & 1 & 0 \\ 0 & 2 & 0 \end{array}\right]$

    If n = $|\vec{v}|$, then T is a n(n+1) matrix.

All variables and coefficients are equal-width bitvectors (8,16,32,64)

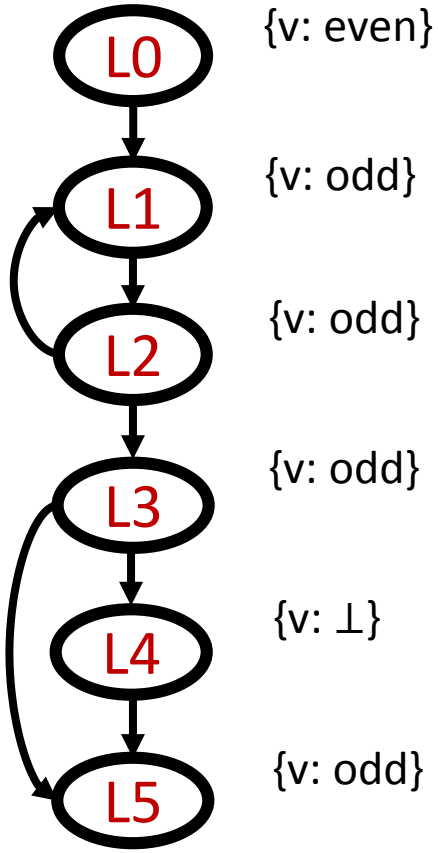
Background: Abstract Interpretation



Background: Abstract Interpretation

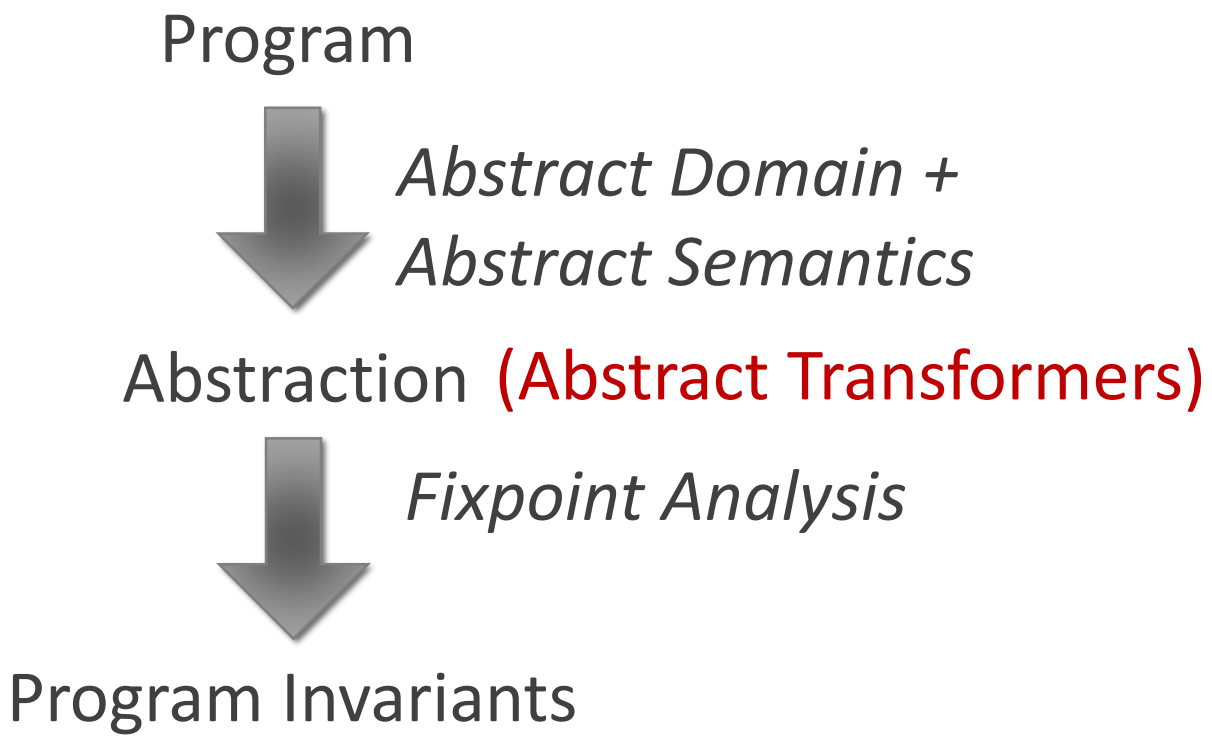
Simple example program with **Parity** Domain.

```
L0:    v=v+1
L1:    while(*) {
L2:       v=v+2
       }
L3:    if(v%2==0) {
L4:       v=v+1
       }
L5:    print(1/v) // assert(v!=0)
```



Abstraction at each node.

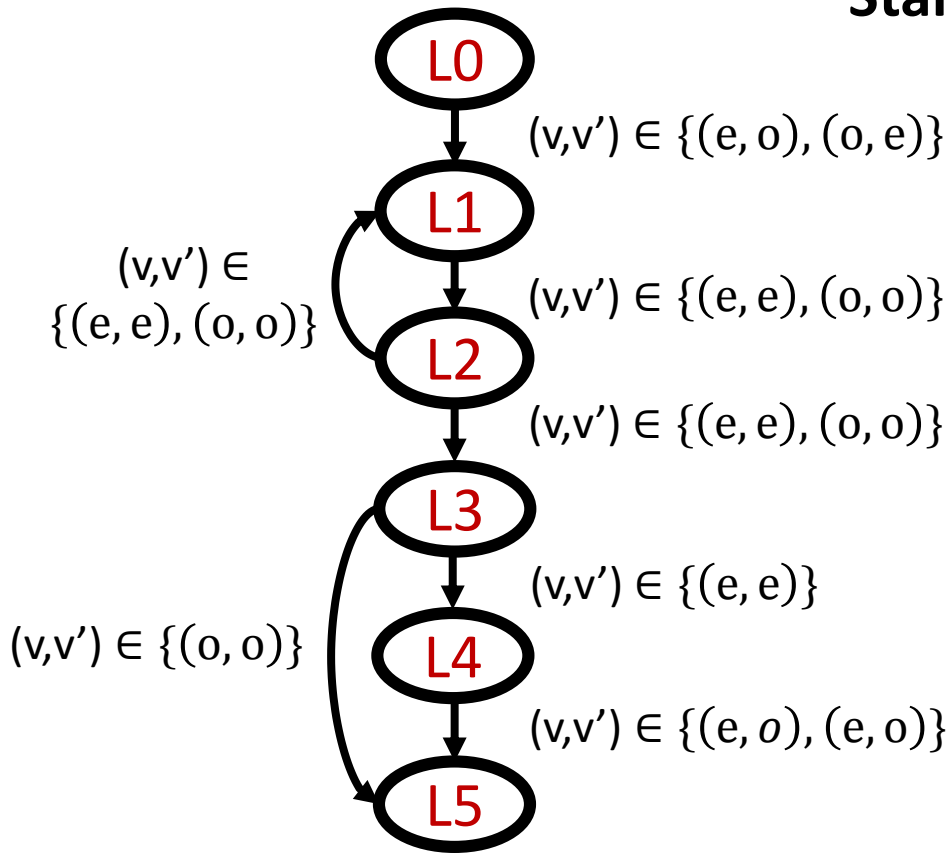
Background: Abstract Interpretation



Abstraction at each node.

Abstraction at each edge

```
L0:     v=v+1
L1:     while(*) {
L2:         v=v+2
        }
L3:     if(v%2==0) {
L4:         v=v+1
        }
L5:     print(1/v)
```



Start (Identity Transformation):

$(v,v') \in \{(e,e),(o,o)\}$

Summary:

$(v,v') \in \{(e,o),(o,o),(o,e)\}$

Background: Abstract Transformers

Type	Operation	Description
$\mathcal{A}$	$\bot$	Bottom element
$bool$	$(a_1 == a_2)$	Equality
$\mathcal{A}$	$(a_1 \sqcup a_2)$	Join
$\mathcal{A}$	$(a_1 \triangledown a_2)$	Widen
$\mathcal{A}$	Id	Identity Transformation
$\mathcal{A}$	$(a_1 \circ a_2)$	Compose

$\bot$ = {} (Representing empty set of points)

$\sqcup$ = least upper bound (Set union for parity domain)

Start (Identity Transformation): (v,v') $\in \{(e,e),(o,o)\}$

$\{(e,o),(o,e)\} \circ \{(e,o),(o,e)\} = \{(e,e),(o,o)\}$

Background: Past Bit-Precise Equality Domains

- KS
- MOS
- Both KS and MOS elements can be used as abstract transformers

KS Definition

A. **K**ing and H. **S**øndergaard, CAV 2008

A matrix, where each row encodes a constraint

Example:

$$\left[\begin{array}{cc|c} 2^{31} & -2^{31} & 0 \end{array}\right] \begin{bmatrix} v \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \end{bmatrix}$$

where, v and v’ are 32-bit values.

In other words, (v-v’) is even.

MOS Definition

M. **M**uller-**O**hm and H. **S**eidl: Set of affine transformers

A set of matrices, every affine combination those matrices may transform the initial state

Example: $\left\{ M_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, M_2 = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \right\}$

$\exists i: \begin{bmatrix} 1 & v' \end{bmatrix} = \begin{bmatrix} 1 & v \end{bmatrix} M \quad M = \begin{bmatrix} 1 & 2p \\ 0 & 1 \end{bmatrix}$

v'= v+2p for some bitvector p. In other words, (v-v') is even.

Bit-Vector Equality Domains

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

Affine-Closed Set

- Affine-closed set = set of affine relations.
- An affine relation is a linear-equality constraint over bitvectors. Example: $2v_1 + 7v_2 + 3 = 0$
- S is an affine-closed set
  - If $p_1 \in S$, $p_2 \in S$ and $k_1 + k_2 = 1$
  - Then $k_1 p_1 + k_2 p_2 \in S$.

KS versus MOS

- Incomparable
- KS can represent pre-condition guard, but MOS cannot:
  - v = 2

$$\left[\begin{array}{cc|c} 1 & 0 & -2 \end{array}\right]\begin{bmatrix} v \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \end{bmatrix}$$

  - MOS cannot express v=2: no affine transformer exists

KS versus MOS

- MOS can encode non-affine-closed relations, but KS cannot
  - Consider MOS element M representing:

    $\exists p.\ v_1' = v_2' = v_1 + p(v_2 - v_1)$
  - $M = (v_1' = v_2' = v_1) \sqcup (v_1' = v_2' = v_2)$

Elements a and b are in M, but their affine combination c is not.

a = $\begin{bmatrix} v_1 & v_2 & v_1' & v_2' \\ 1 & -1 & 1 & 1 \end{bmatrix}$

b = [2 -2 6 6]

c = [0 0 -4 -4] (2a-b)

Why are KS and MOS incomparable?

- KS is affine-closed-set of ‘concrete states’.
- MOS is affine-closed-set of ‘concrete affine transformers’.
- KS defines constraints on the variables of a program, i.e. $\vec{v}'$ and $\vec{v}$ (2n variables: n = $|\vec{v}|$).
- However, MOS defines constraints on the elements of affine transformers T (n(n+1) coefficients).
- Generalize this behavior to create new abstract domains like MOS:

  ATA[KS] = MOS, ATA[$I_{z_2 w}$] = ?

ATA Contributions

- Affine Transformer Abstraction Framework (ATA[B])
- Parameter B allows control over precision/performance tradeoff
- Provide abstract-domain operations for ATA, such as ‘Join’ and ‘Abstract Composition’

Program Analysis using ATA[KS]

```
ENT:  int f(int x) {
L0:       int i = 0, r = 0;
L1:       while(i <= 10) {
L2:          if(*)
L3:             r = r + 2*x;
L4:          i = i + 1;
          }
L5:       return r;
       }
```

Function Summary for ‘f’:

$$\exists i : r' = 2ix$$

Edge	Transformer
L0 → L1	$\left\{ \left[\begin{array}{c\|ccc} 1 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right] \right\}$
L3 → L4	$\left\{ \left[\begin{array}{c\|ccc} 1 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right] \right\}$
L4 → L1	$\left\{ \left[\begin{array}{c\|ccc} 1 & 0 & 1 & 0 \\ \hline 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right] \right\}$

Abstract Transformers for ATA[KS]

Program Analysis using ATA[$I_{z_2 w}$]

```
ENT:  int f(int x) {
L0:       int i = 0, r = 0;
L1:       while(i <= 10) {
L2:          if(*)
L3:             r = r + 2*x;
L4:          i = i + 1;
          }
L5:       return r;
       }
```

Function Summary for ‘f’:

r′ = [0,20]x

Edge	Transformer
L0 → L1	$\begin{bmatrix} 1 & [0,0] & [0,0] & [0,0] \\ \hline 0 & [1,1] & [0,0] & [0,0] \\ 0 & [0,0] & [0,0] & [0,0] \\ 0 & [0,0] & [0,0] & [0,0] \end{bmatrix}$
L1 → L2	$\begin{bmatrix} 1 & [0,0] & [0,10] & [0,0] \\ \hline 0 & [1,1] & [0,0] & [0,0] \\ 0 & [0,0] & [0,0] & [0,0] \\ 0 & [0,0] & [0,0] & [1,1] \end{bmatrix}$
L3 → L4	$\begin{bmatrix} 1 & [0,0] & [0,0] & [0,0] \\ \hline 0 & [1,1] & [0,0] & [2,2] \\ 0 & [0,0] & [1,1] & [0,0] \\ 0 & [0,0] & [0,0] & [1,1] \end{bmatrix}$
L4 → L1	$\begin{bmatrix} 1 & [0,0] & [1,1] & [0,0] \\ \hline 0 & [1,1] & [0,0] & [0,0] \\ 0 & [0,0] & [1,1] & [0,0] \\ 0 & [0,0] & [0,0] & [1,1] \end{bmatrix}$

Abstract Transformers for ATA[$I_{z_2 w}$]

$\mathcal{A}$ (ATA[B]) Abstract-Domain Operations

Each element a ∈ $\mathcal{A}$ contains an element base(a) ∈ B.

Type	Operation
$\mathcal{A}$	⊥
$\mathcal{bool}$	$(a_1 \cong$
$\mathcal{A}$	$(a_1 \widetilde{\sqcup} a_2)$
$\mathcal{A}$	Id
$\mathcal{A}$	$(a_1 \circ a_2)$

$$\left[\begin{array}{l|ll} [1,1] & [0,10] & [0,0] \\ \hline [0,0] & [1,1] & [0,0] \\ [0,0] & [0,0] & [1,1] \end{array}\right] \widetilde{\sqcup} \left[\begin{array}{l|ll} [1,1] & [5,15] & [0,1] \\ \hline [0,0] & [2,2] & [0,0] \\ [0,0] & [1,3] & [1,2] \end{array}\right]$$

$$= \left[\begin{array}{l|ll} [1,1] & [0,15] & [0,1] \\ \hline [0,0] & [1,2] & [0,0] \\ [0,0] & [0,3] & [1,2] \end{array}\right]$$

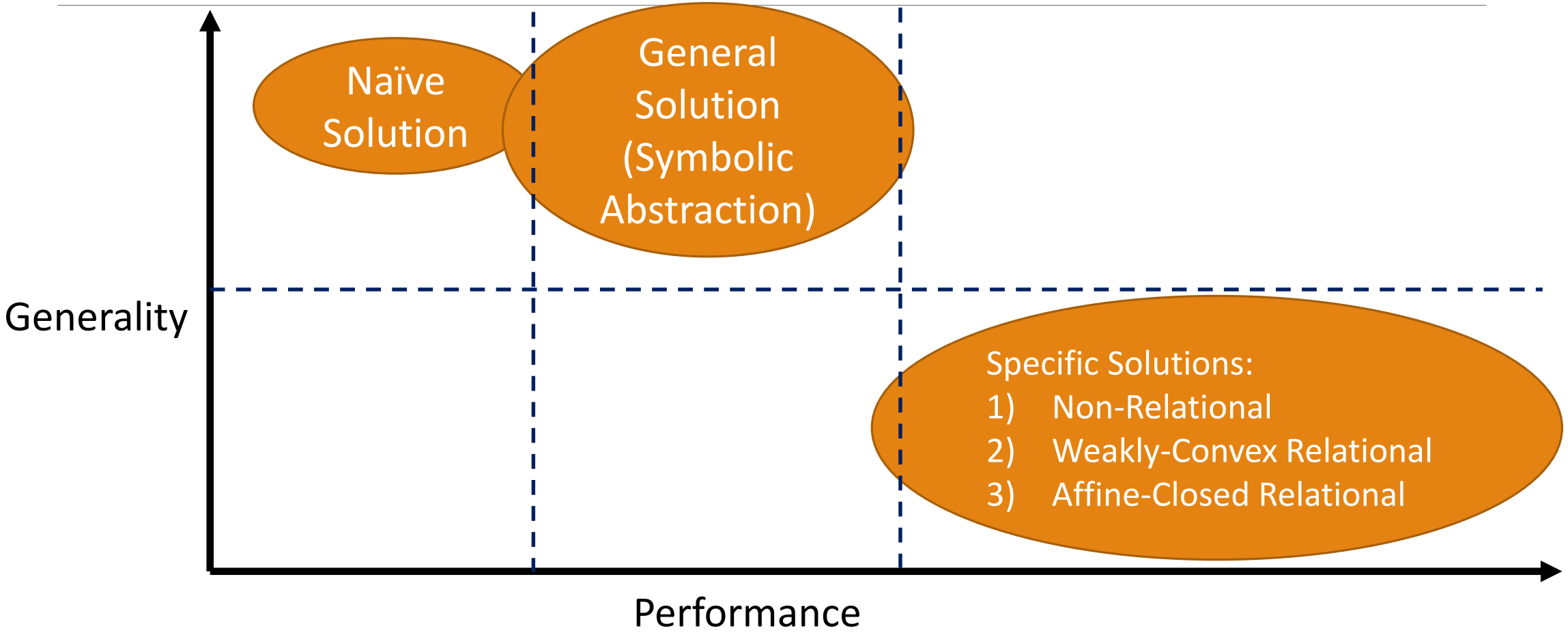
Abstract Composition

- $a_3 = a_2 \circ a_1$
- If *affine transformer* $t_1 \in \gamma(a_1)$ and affine transformer $t_2 \in \gamma(a_2)$, then $\boldsymbol{(t_1 \times t_2) \in \gamma(a_3)}$.
- t_1 and t_2 are (n+1)×(n+1) matrices.

Best (but Naïve) Solution

- Enumerate all concrete affine transformers $t_1 \in \gamma(a_1)$, $t_2 \in \gamma(a_2)$
- Perform matrix multiplication $(t_1 \times t_2)$ of each such pair
- Join over all $(t_1 \times t_2)$
- Infeasible
- Better Solution: Represent $(t_1 \times t_2)$ symbolically
- Non-linear components: $t_1 \times t_2$

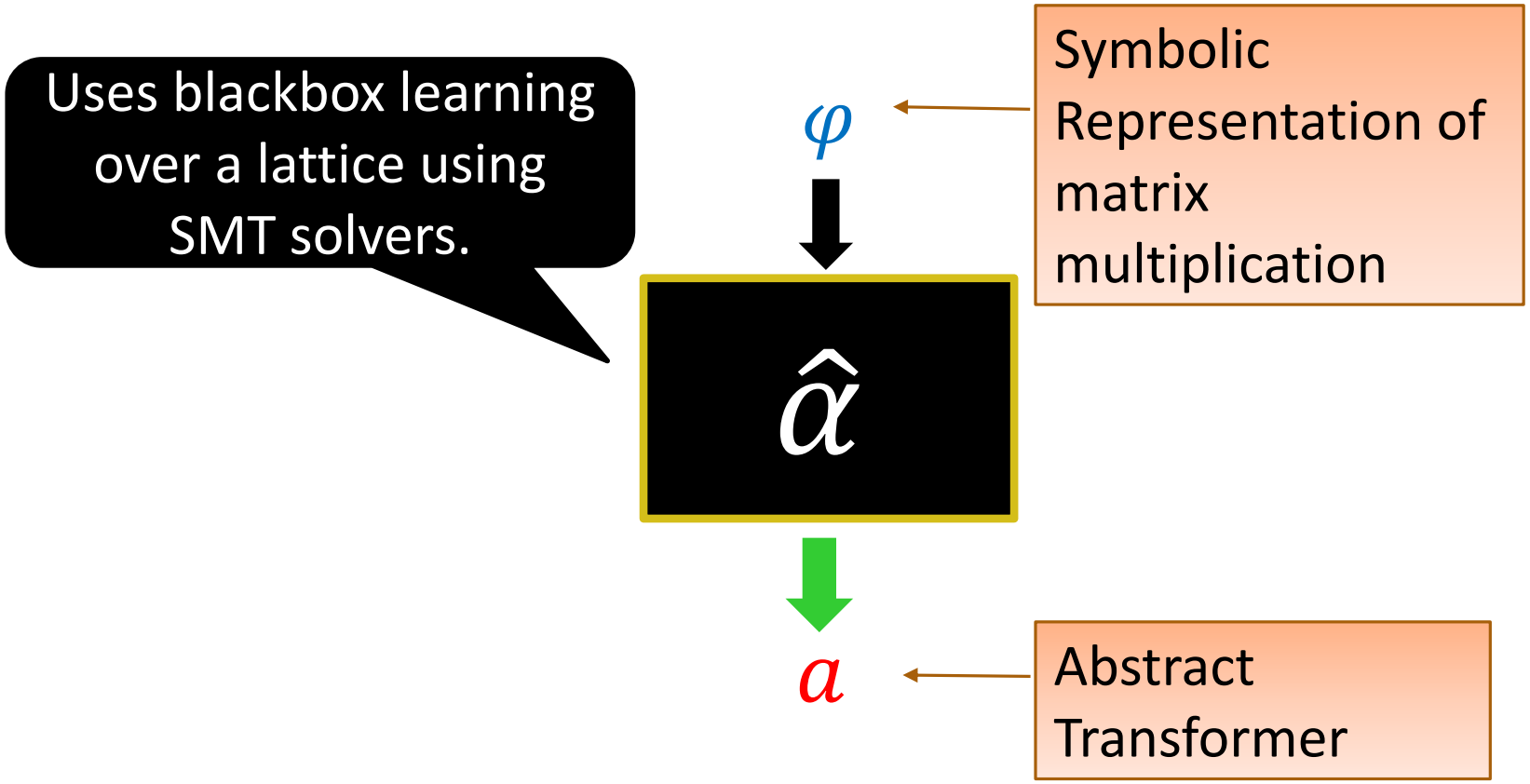
Abstract Composition



General Case

- Use Symbolic Abstraction
  - Employ SMT solvers to cleverly search the space of the resulting abstract composition
  - Offshore solving non-linear bitvector equations to the solver

Abstract Composition via Symbolic Abstraction



Special Case: **Non-relational** Base Domain

- Use abstract addition and multiplication operations to perform abstract composition.
- $$\left[\begin{array}{c|cc} [1,1] & [0,10] & [0,0] \\ \hline [0,0] & [1,1] & [2,3] \\ [0,0] & [0,0] & [1,1] \end{array}\right] \circ \left[\begin{array}{c|cc} [1,1] & [0,0] & [0,0] \\ \hline [0,0] & [2,4] & [0,0] \\ [0,0] & [1,3] & [1,2] \end{array}\right] =$$

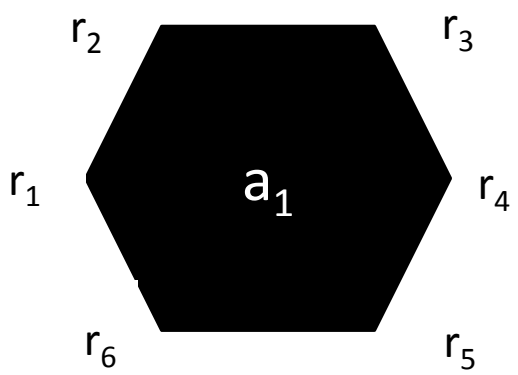
$$\left[\begin{array}{c|cc} [1,1] & [0,10] \,.^{\#}\, [2,4] & [0,0] \\ \hline [0,0] & ([1,1] \,.^{\#}\, [2,4]) +^{\#} ([2,3] \,.^{\#}\, [1,3]) & [2,3] \,.^{\#}\, [1,2] \\ [0,0] & [1,1] \,.^{\#}\, [1,3] & [1,1] \,.^{\#}\, [1,2] \end{array}\right] = \left[\begin{array}{c|cc} [1,1] & [0,40] & [0,0] \\ \hline [0,0] & [4,13] & [2,6] \\ [0,0] & [1,3] & [1,2] \end{array}\right]$$

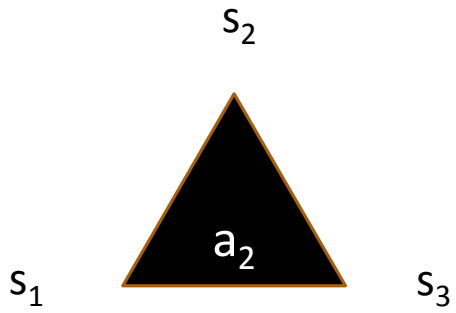
Examples of Non-relational base domains

- Small sets (SS_n): All sets with maximum cardinality n
- Intervals ($I_{z_2 w}$): [a,b] = {a, a+1, a+2, …, b}
- Strided Intervals ($SI_{z_2 w}$): s[a,b] = {a, a+s, a+2s, …, b}

Special Cases of Relational base domains

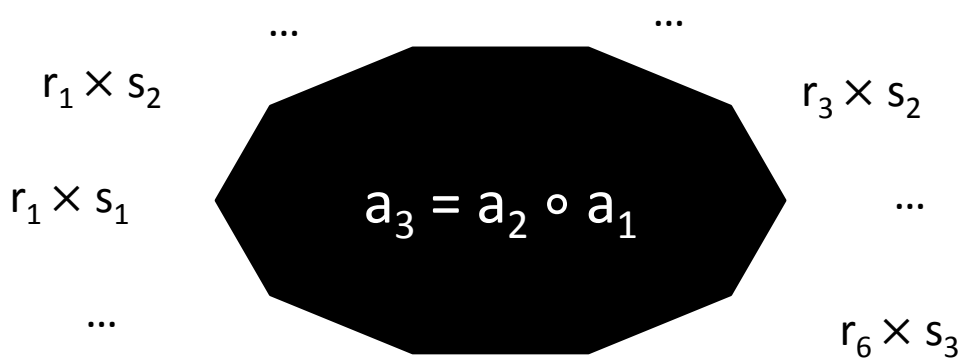
- (Affine-Closed, Weakly-Convex) Base Domains
  - Use the generator representation.
  - $a_1 = Gen(\{r_1, r_2, \ldots, r_{n1}\})$, $a_2 = Gen(\{s_1, s_2, \ldots, s_{n2}\})$.
  - r_i, s_j are affine transformers ((n+1)×(n+1) matrices)





Special Cases of Relational base domains

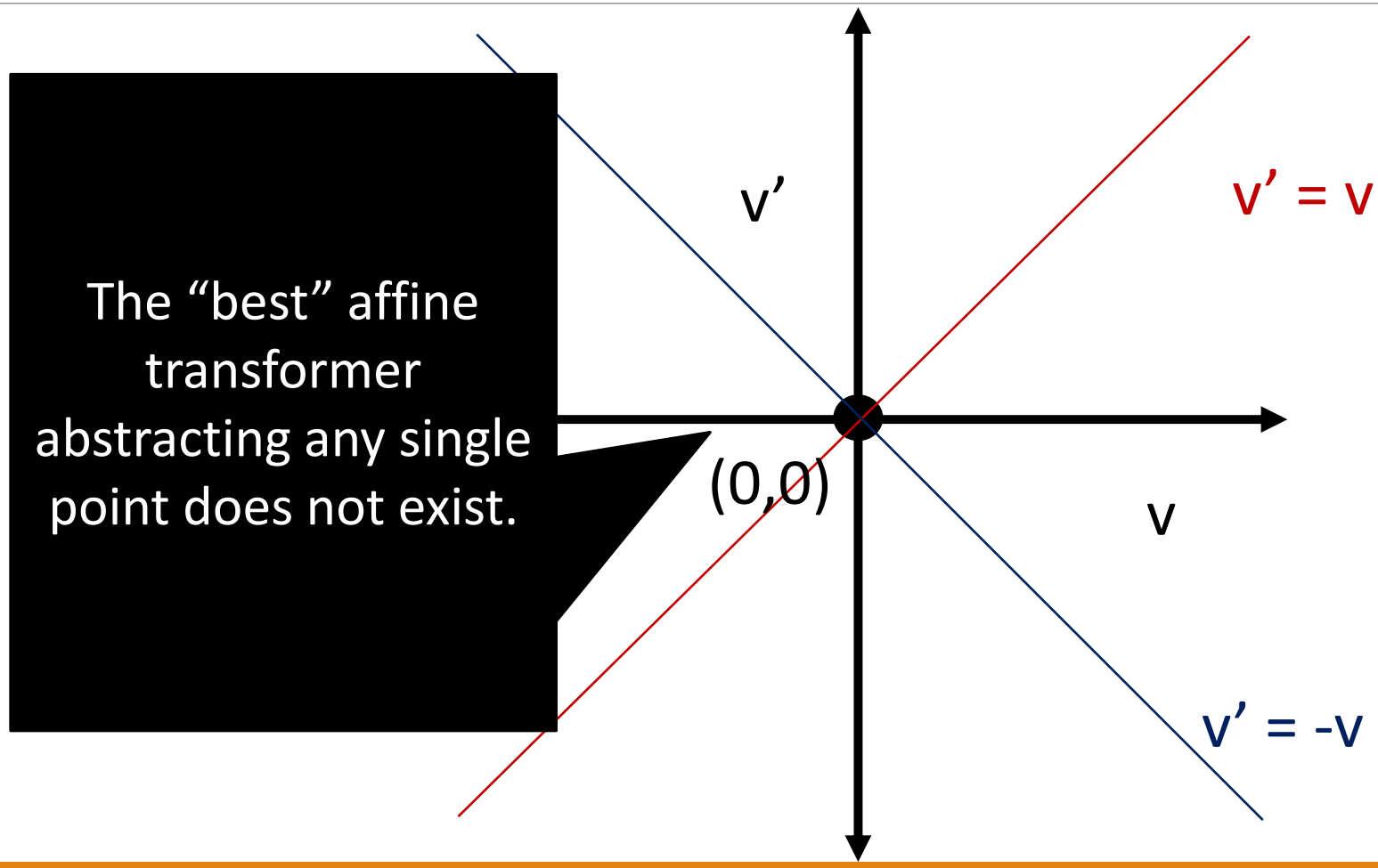
- Matrix multiplication over generators is sufficient (no SMT calls).
  - $a_3 = \text{Gen}(\{r_1 \times s_1, \ldots, r_1 \times s_{n2}, r_2 \times s_1, \ldots, r_{n1} \times s_{n2}\})$



Examples of base relational domains

- KS Domain: Affine Relations
- Bit-Vector Sound versions of
  - Polyhedra
  - Octagons

Discussion: No Greatest Lower bound in ATA[B]



Discussion

- **No** Galois Connection between ATA[B] and the concrete domain C (powerset over concrete states).
- Greatest upper bound does **not** exist for ATA[B], and, in general Least Upper Bound Operation does not exist either.
- Multiple incomparable ways to abstract 'assumes'
  - Example: ***assume(x<=5)*** with ATA[$I_{z_2 w}$]
    - x' = [1,1]x + [0,0]
    - x' = [0,0]x + [0,5]

Recap

- Introduced a generic framework of abstract domains: ATA[B].
- Parameter B allows control over precision/performance tradeoff.
- B and ATA[B] are, in general, incomparable.
- Fast abstract composition for some classes of B:
  - Non-relational Domains
  - Affine-closed or Weakly Convex Relational Domains
- ATA framework can be extended to integers and rationals as well.

Questions?

- Affine Transformer Abstraction Framework: ATA[B]
- Family of abstract domains
- Parametrized over a base domain ‘B’ for bitvectors