Numerical Invariants via Abstract Machines

Zachary Kincaid

Princeton University

Static Analysis Symposium
August 31, 2018

Compositional Recurrence Analysis (CRA)

- Technique for generating numerical invariants
- Joint work with Jason Breck, Ashkan Forouhi Boroujeni, John Cyphert, Azadeh Farzan, Thomas Reps.

Today's agenda: A recipe for building abstract interpreters

Compositional Recurrence Analysis

Compositional Recurrence Analysis

- Generates numerical invariants is an expressive assertion language
  - Linear arithmetic, polynomials, exponentials, logarithms
  - Equations and inequations, congruences, disjunctions

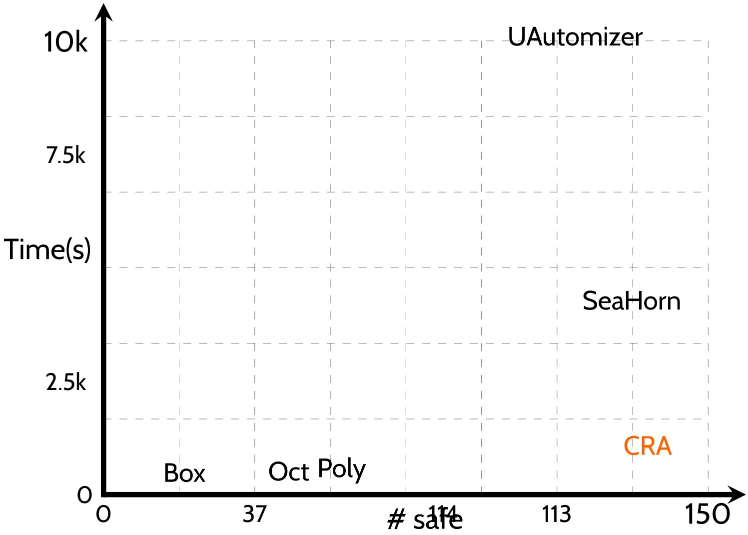
Compositional Recurrence Analysis

- Generates numerical invariants is an expressive assertion language
  - Linear arithmetic, polynomials, exponentials, logarithms
  - Equations and inequations, congruences, disjunctions
- Strongly compositional
  - Potential to scale, be parallelized, apply to incomplete programs, incremental analysis, ..

Compositional Recurrence Analysis

- Generates numerical invariants is an expressive assertion language
  - Linear arithmetic, polynomials, exponentials, logarithms
  - Equations and inequations, congruences, disjunctions
- Strongly compositional
  - Potential to scale, be parallelized, apply to incomplete programs, incremental analysis, ..
  - No context $\Rightarrow$ no forward propagation, no abstract refinement

HOLA/C4B/SVComp benchmarks (linear)



How can we answer questions about the behavior of software?

How can we answer questions about the behavior of software?

Practice

- Abstract domains
- Constraint-based analysis
- Interpolation
- Property-directed reachability
- …

How can we answer questions about the behavior of software?

Practice

- Abstract domains
- Constraint-based analysis
- Interpolation
- Property-directed reachability
- ...

Theory

- Boolean programs
- Linear dynamical systems
- Solvable polynomial systems
- Vector addition systems
- ...

How can we answer questions about the behavior of software?

Practice

- Abstract domains
- Constraint-based analysis
- Interpolation
- Property-directed reachability
- …

Theory

- Boolean programs
- Linear dynamical systems
- Solvable polynomial systems
- Vector addition systems
- …

Outline

Background

The recipe

Examples

Goal

Given a program:

$$
\begin{aligned}
x &\in \mathsf{Var} \\
n &\in \mathbb{Z} \\
e \in \mathsf{Expr} &::= x \mid n \mid e_1 + e_2 \mid n \cdot e \\
c \in \mathsf{Cond} &::= e_1 < e_2 \mid e_1 = e_2 \mid c_1 \wedge c_2 \mid c_1 \vee c_2 \\
P_1 \in \mathsf{Program} &::= x := e \mid \texttt{if}\ \ c\ \mathsf{then}\ P\ \mathsf{else}\ P \mid \mathsf{while}\ c\ \mathsf{do}\ P
\end{aligned}
$$

Goal

Given a program:

$$
\begin{aligned}
x &\in \mathsf{Var} \\
n &\in \mathbb{Z} \\
e \in \mathsf{Expr} &::= x \mid n \mid e_1 + e_2 \mid n \cdot e \\
c \in \mathsf{Cond} &::= e_1 < e_2 \mid e_1 = e_2 \mid c_1 \wedge c_2 \mid c_1 \vee c_2 \\
P_1 \in \mathsf{Program} &::= x := e \mid \texttt{if}\ \ c\ \mathsf{then}\ P\ \mathsf{else}\ P \mid \mathsf{while}\ c\ \mathsf{do}\ P
\end{aligned}
$$

Compute a transition formula

$$
\begin{aligned}
t \in \mathsf{Term} &::= \mathsf{x} \mid \mathsf{x'} \mid n \mid t_1 + t_2 \mid t_1 t_2 \mid v \\
F \in \mathsf{TF} &::= s < 0 \mid s = 0 \mid F_1 \vee F_2 \mid F_1 \wedge F_2 \mid \exists v. F
\end{aligned}
$$

Goal

Given a program:

$$
\begin{aligned}
x &\in \text{Var} \\
n &\in \mathbb{Z} \\
e \in \text{Expr} &::= x \mid n \mid e_1 + e_2 \mid n \cdot e \\
c \in \text{Cond} &::= e_1 < e_2 \mid e_1 = e_2 \mid c_1 \wedge c_2 \mid c_1 \vee c_2 \\
P_1 \in \text{Program} &::= x := e \mid \texttt{if}\ c \text{ then } P \text{ else } P \mid \text{while } c \text{ do } P
\end{aligned}
$$

Compute a transition formula

Post-state

$$
\begin{aligned}
t \in \text{Term} &::= \text{x} \mid \text{x'} \mid n \mid t_1 + t_2 \mid t_1 t_2 \mid v \\
F \in \text{TF} &::= s < 0 \mid s = 0 \mid F_1 \vee F_2 \mid F_1 \wedge F_2 \mid \exists v.F
\end{aligned}
$$

Running example

```
x := 0;
while (x < N) do
    x := x + 1;
    if (*) then
        y := y + x
    else
        z := z + x
```

Running example

```
x := 0;
while (x < N) do
    x := x + 1;
    if (*) then
        y := y + x
    else
        z := z + x
```

$$x' = N$$

Running example

```
x := 0;
while (x < N) do
    x := x + 1;
    if (*) then
        y := y + x
    else
        z := z + x
```

$$\left.\right\} \quad \begin{aligned} & x' = N \\ \wedge\, & y' \geq y \\ \wedge\, & z' \geq z \end{aligned}$$

Running example

```
x := 0;
while (x < N) do
    x := x + 1;
    if (*) then
        y := y + x
    else
        z := z + x
```

$$\left.\begin{array}{l} \quad x' = N \\ \wedge y' \geq y \\ \wedge z' \geq z \\ \wedge y' + z' = y + z + N(N+1)/2 \end{array}\right.$$

Effective denotational semantics

$$TF[\![\cdot]\!] \;:\; \text{Program} \rightarrow \text{TransitionFormula}$$

Effective denotational semantics

$$\textit{TF}[\![\cdot]\!] \;:\; \mathsf{Program} \to \mathsf{TransitionFormula}$$

$$\textit{TF}[\![\mathtt{x\ :=\ e}]\!] \triangleq \mathsf{x}' = e \wedge \bigwedge_{\mathsf{y} \neq \mathsf{x} \in \mathsf{Var}} \mathsf{y}' = \mathsf{y}$$

Effective denotational semantics

$$\mathit{TF}[\![\cdot]\!] : \mathsf{Program} \to \mathsf{TransitionFormula}$$

$$\mathit{TF}[\![\texttt{x := e}]\!] \triangleq \mathsf{x}' = e \land \bigwedge_{\mathsf{y} \neq \mathsf{x} \in \mathsf{Var}} \mathsf{y}' = \mathsf{y}$$

$$\mathit{TF}[\![\texttt{if } c \texttt{ then } P_1 \texttt{ else } P_2]\!] \triangleq (c \land \mathit{TF}[\![P_1]\!]) \lor (\neg c \land \mathit{TF}[\![P_2]\!])$$

Effective denotational semantics

$$\mathit{TF}[\![\cdot]\!] \;:\; \mathsf{Program} \rightarrow \mathsf{TransitionFormula}$$

$$\mathit{TF}[\![\texttt{x := e}]\!] \triangleq \mathsf{x}' = e \wedge \bigwedge_{\mathsf{y} \neq \mathsf{x} \in \mathsf{Var}} \mathsf{y}' = \mathsf{y}$$

$$\mathit{TF}[\![\texttt{if } c \texttt{ then } P_1 \texttt{ else } P_2]\!] \triangleq (c \wedge \mathit{TF}[\![P_1]\!]) \vee (\neg c \wedge \mathit{TF}[\![P_2]\!])$$

$$\mathit{TF}[\![P_1 ; P_2]\!] \triangleq \exists \mathsf{Var}''.\mathit{TF}[\![P_1]\!][\mathsf{Var}' \mapsto \mathsf{Var}''] \wedge \mathit{TF}[\![P_2]\!][\mathsf{Var} \mapsto \mathsf{Var}'']$$

Effective denotational semantics

$$\mathit{TF}[\![\cdot]\!] \; : \; \mathsf{Program} \to \mathsf{TransitionFormula}$$

$$\mathit{TF}[\![\texttt{x := e}]\!] \triangleq \mathsf{x}' = e \wedge \bigwedge_{\mathsf{y} \neq \mathsf{x} \in \mathsf{Var}} \mathsf{y}' = \mathsf{y}$$

$$\mathit{TF}[\![\texttt{if } c \texttt{ then } P_1 \texttt{ else } P_2]\!] \triangleq (c \wedge \mathit{TF}[\![P_1]\!]) \vee (\neg c \wedge \mathit{TF}[\![P_2]\!])$$

$$\mathit{TF}[\![P_1 \, ; P_2]\!] \triangleq \exists \mathsf{Var}''. \mathit{TF}[\![P_1]\!][\mathsf{Var}' \mapsto \mathsf{Var}''] \wedge \mathit{TF}[\![P_2]\!][\mathsf{Var} \mapsto \mathsf{Var}'']$$

$$\mathit{TF}[\![\texttt{while } c \texttt{ do } P]\!] \triangleq \ldots$$

Effective denotational semantics

$$\mathit{TF}[\![\cdot]\!] \ : \ \text{Program} \to \text{TransitionFormula}$$

$$\mathit{TF}[\![\texttt{x := e}]\!] \triangleq \mathsf{x}' = e \wedge \bigwedge_{\mathsf{y} \neq \mathsf{x} \in \mathsf{Var}} \mathsf{y}' = \mathsf{y}$$

$$\mathit{TF}[\![\texttt{if } c \texttt{ then } P_1 \texttt{ else } P_2]\!] \triangleq (c \wedge \mathit{TF}[\![P_1]\!]) \vee (\neg c \wedge \mathit{TF}[\![P_2]\!])$$

$$\mathit{TF}[\![P_1 ; P_2]\!] \triangleq \exists \mathsf{Var}''.\mathit{TF}[\![P_1]\!][\mathsf{Var}' \mapsto \mathsf{Var}''] \wedge \mathit{TF}[\![P_2]\!][\mathsf{Var} \mapsto \mathsf{Var}'']$$

$$\mathit{TF}[\![\texttt{while } c \texttt{ do } P]\!] \triangleq \ldots$$

Aside

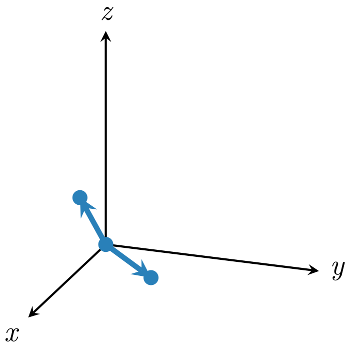
- Arbitrary control flow [Tarjan ’81]
- Recursive procedures [PLDI’17]

Compositionality $\Rightarrow$ TF for a loop is a function of the TF of its body

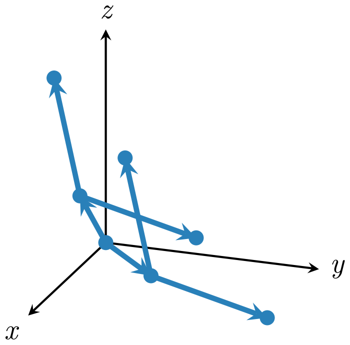
```
x := 0;
while (x < N) do
    x := x + 1;
    if (*) then
        y := y + x
    else
        z := z + x
```

$$x' = x + 1 \wedge \left(\begin{array}{l} (y' = y + x' \wedge z' = z) \\ \vee (y' = y \wedge z' = z + x') \end{array} \right)$$

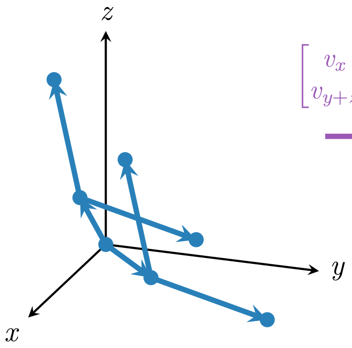
$$\begin{aligned} & x' = x + 1 \\ \wedge & \begin{pmatrix} (y' = y + x' \wedge z' = z) \\ \vee (y' = y \wedge z' = z + x') \end{pmatrix} \end{aligned}$$



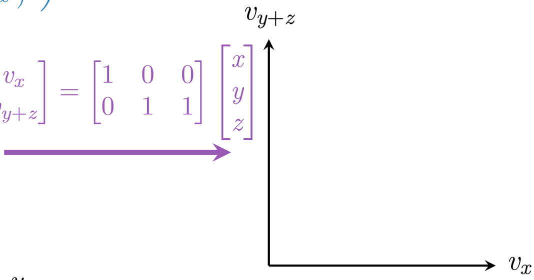
$$
\begin{aligned}
& x' = x + 1 \\
\wedge & \begin{pmatrix} \quad (y' = y + x' \wedge z' = z) \\ \vee (y' = y \wedge z' = z + x') \end{pmatrix}
\end{aligned}
$$



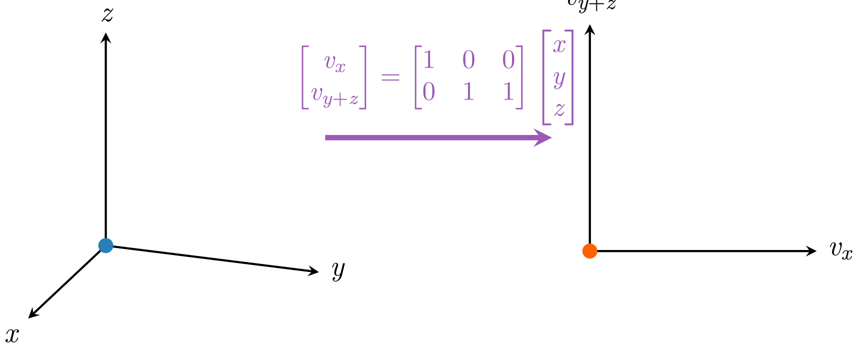
$$
\begin{aligned}
& x' = x + 1 \\
\wedge & \left(\begin{array}{l} (y' = y + x' \wedge z' = z) \\ \vee (y' = y \wedge z' = z + x') \end{array} \right)
\end{aligned}
$$

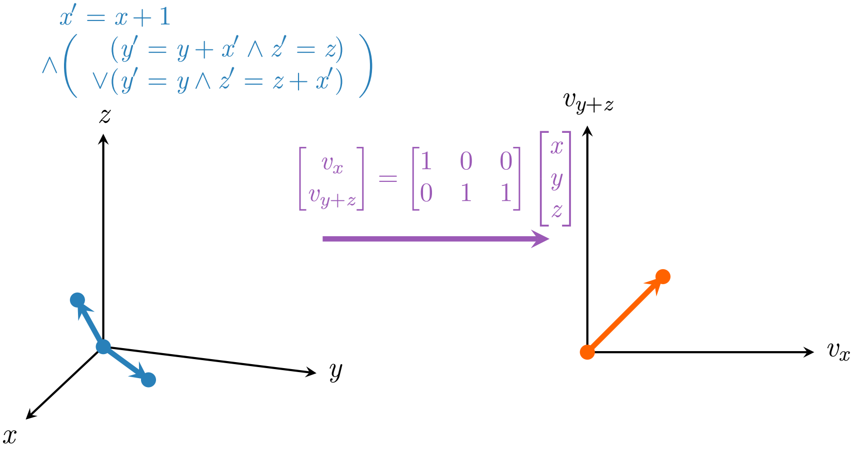


$$
\begin{bmatrix} v_x \\ v_{y+z} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}
$$

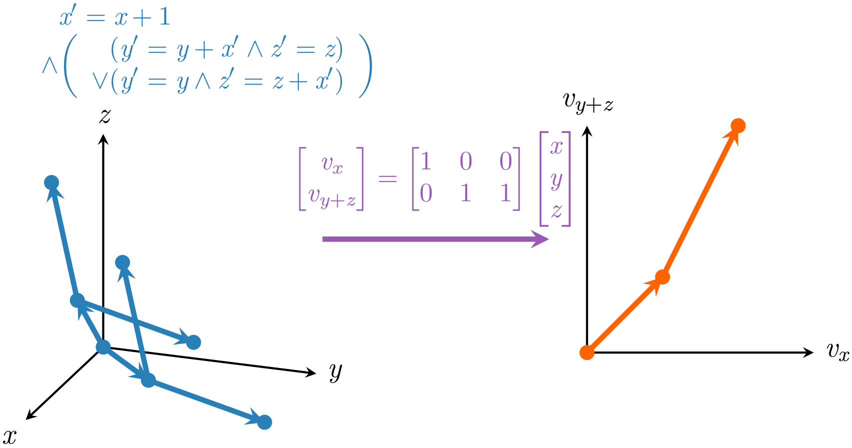


$$
\begin{aligned}
& x' = x + 1 \\
\wedge & \left(\begin{array}{l} (y' = y + x' \wedge z' = z) \\ \vee (y' = y \wedge z' = z + x') \end{array} \right)
\end{aligned}
$$

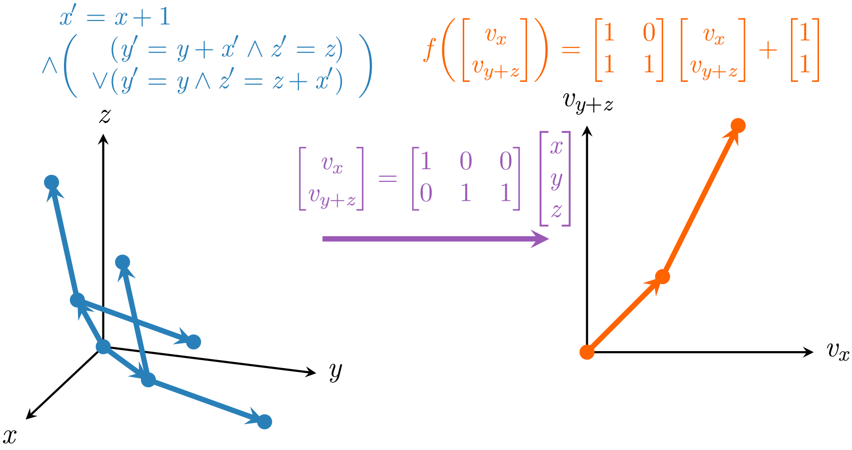




$x' = x + 1$
$\wedge \left(\begin{array}{l} (y' = y + x' \wedge z' = z) \\ \vee (y' = y \wedge z' = z + x') \end{array} \right)$
z
y
x
$\begin{bmatrix} v_x \\ v_{y+z} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$
v_{y+z}
v_x



$x' = x + 1$
$\wedge \left(\begin{array}{l} (y' = y + x' \wedge z' = z) \\ \vee (y' = y \wedge z' = z + x') \end{array} \right)$
z
y
x
$\begin{bmatrix} v_x \\ v_{y+z} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$
v_{y+z}
v_x


$x' = x + 1$
$\wedge \left(\begin{array}{l} (y' = y + x' \wedge z' = z) \\ \vee (y' = y \wedge z' = z + x') \end{array} \right)$
$f\left(\begin{bmatrix} v_x \\ v_{y+z} \end{bmatrix} \right) = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_{y+z} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix}$
$\begin{bmatrix} v_x \\ v_{y+z} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$
z
y
x
v_{y+z}
v_x

$$x' = x + 1 \\ \wedge \begin{pmatrix} (y' = y + x' \wedge z' = z) \\ \vee (y' = y \wedge z' = z + x') \end{pmatrix}$$

$$f\left(\begin{bmatrix} v_x \\ v_{y+z} \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_{y+z} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} v_x \\ v_{y+z} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$f^k\left(\begin{bmatrix} v_x \\ v_{y+z} \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 \\ k & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_{y+z} \end{bmatrix} + \begin{bmatrix} k \\ k(k+1)/2 \end{bmatrix}$$

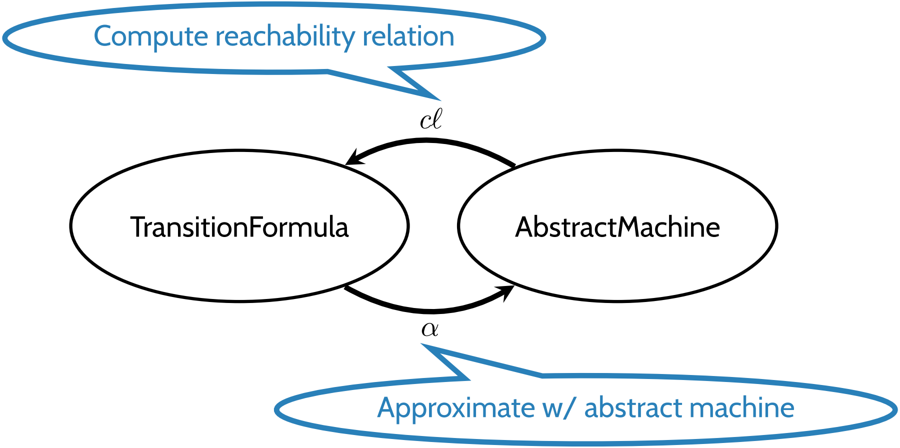
$$
\begin{array}{l}
x' = x + 1 \\
\wedge \left(\begin{array}{l} (y' = y + x' \wedge z' = z) \\ \vee (y' = y \wedge z' = z + x') \end{array} \right)
\end{array}
$$

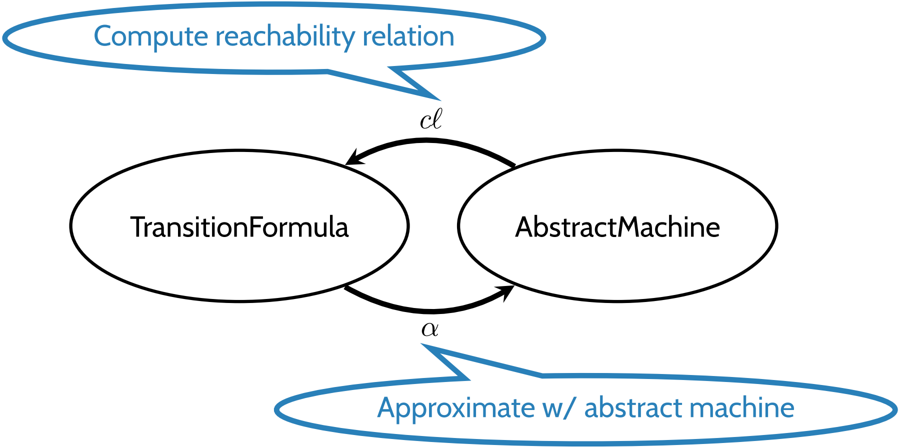
$$
f\left(\begin{bmatrix} v_x \\ v_{y+z} \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_{y+z} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix}
$$

$$
\begin{bmatrix} v_x \\ v_{y+z} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}
$$

$$
f^k\left(\begin{bmatrix} v_x \\ v_{y+z} \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 \\ k & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_{y+z} \end{bmatrix} + \begin{bmatrix} k \\ k(k+1)/2 \end{bmatrix}
$$

$$
\exists k. \left(\begin{array}{l} k \geq 0 \\ \wedge x' = x + k \\ \wedge (y' + z') = (y + z) + kx + k(k+1)/2 \end{array} \right)
$$


Compute reachability relation
$c\ell$
TransitionFormula
AbstractMachine
α
Approximate w/ abstract machine



$$\mathit{TF}[\![\texttt{while } c \texttt{ do } P]\!] = c\ell(\alpha(c \wedge \mathit{TF}[\![P]\!])) \wedge \neg c'$$

Simulation

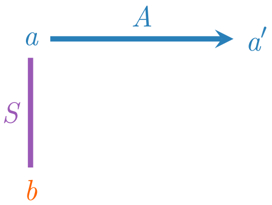
Let $(A, \xrightarrow{A})$ and $(B, \xrightarrow{B})$ be transition systems.
A relation $S \subseteq A \times B$ is a (total) simulation if

1. For all $a \in A$, there is some $b \in B$ with $(a, b) \in S$

Simulation

Let $(A, \xrightarrow{A})$ and $(B, \xrightarrow{B})$ be transition systems.
A relation $S \subseteq A \times B$ is a (total) simulation if

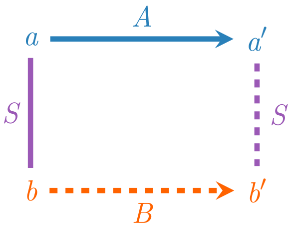
1. For all $a \in A$, there is some $b \in B$ with $(a, b) \in S$
2. For all a, b, a' such that $(a, b) \in S$ and $a \rightarrow_A a'$, there is some b' such that $b \rightarrow_B b'$ and $(a', b') \in S$.



Simulation

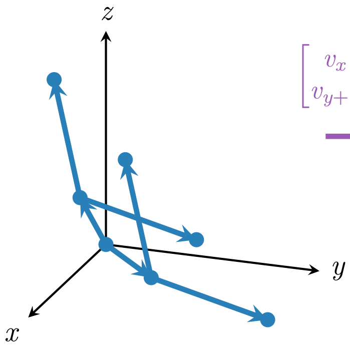
Let $(A, \xrightarrow{A})$ and $(B, \xrightarrow{B})$ be transition systems.
A relation $S \subseteq A \times B$ is a (total) simulation if

1. For all $a \in A$, there is some $b \in B$ with $(a, b) \in S$
2. For all a, b, a' such that $(a, b) \in S$ and $a \rightarrow_A a'$, there is some b' such that $b \rightarrow_B b'$ and $(a', b') \in S$.

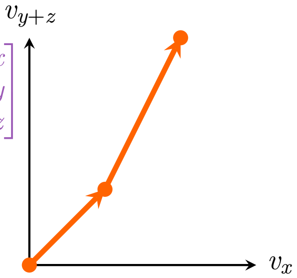


$$
\begin{aligned}
& x' = x + 1 \\
\wedge & \left(\begin{aligned} & (y' = y + x' \wedge z' = z) \\ \vee & (y' = y \wedge z' = z + x') \end{aligned} \right)
\end{aligned}
$$

$$
f\left(\begin{bmatrix} v_x \\ v_{y+z} \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_{y+z} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix}
$$



$$
\begin{bmatrix} v_x \\ v_{y+z} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}
$$



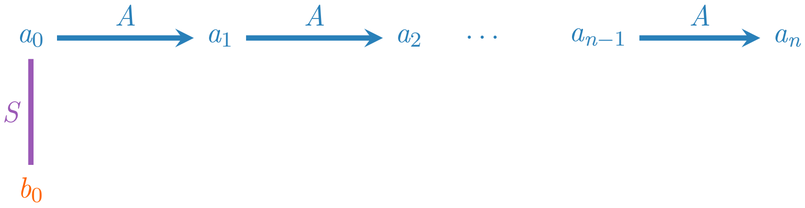
$$\begin{aligned} & x' = x + 1 \\ \wedge & \left(\begin{array}{l} \quad (y' = y + x' \wedge z' = z) \\ \vee (y' = y \wedge z' = z + x') \end{array} \right) \end{aligned}$$

$$\begin{aligned} & v_x = x \\ \wedge\ & v_{y+z} = y + z \\ \wedge\ & \ell_y \leq y \\ \wedge\ & \ell_z \leq z \end{aligned}$$

$$g\left(\begin{bmatrix} v_x \\ v_{y+z} \\ \ell_y \\ \ell_z \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_{y+z} \\ \ell_y \\ \ell_z \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

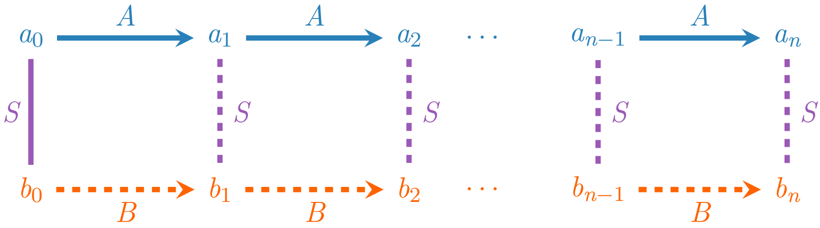
Approximating transitive closure

$$(B, \xrightarrow{B}) \text{ simulates } (A, \xrightarrow{A}) \Rightarrow (B, \xrightarrow{B}^*) \text{ simulates } (A, \xrightarrow{A}^*)$$



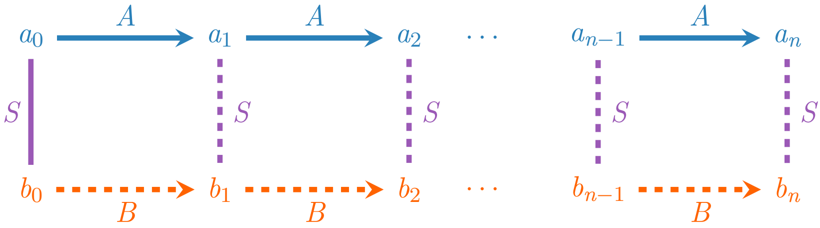
Approximating transitive closure

$(B, \xrightarrow{B})$ simulates $(A, \xrightarrow{A}) \Rightarrow (B, \xrightarrow{B}^*)$ simulates $(A, \xrightarrow{A}^*)$



Approximating transitive closure

$(B, \xrightarrow{B})$ simulates $(A, \xrightarrow{A}) \Rightarrow (B, \xrightarrow{B}^*)$ simulates $(A, \xrightarrow{A}^*)$



$$a \xrightarrow{A}^* a' \Rightarrow \forall b. S(a, b) \Rightarrow \exists b'. b \xrightarrow{B}^* b' \wedge S(a', b')$$

$$
\begin{array}{l} x' = x + 1 \\ \wedge \left(\begin{array}{l} (y' = y + x' \wedge z' = z) \\ \vee (y' = y \wedge z' = z + x') \end{array} \right) \end{array}
$$

$$
f\left(\begin{bmatrix} v_x \\ v_{y+z} \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_{y+z} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix}
$$

$$
\begin{bmatrix} v_x \\ v_{y+z} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}
$$

$$
f^k\left(\begin{bmatrix} v_x \\ v_{y+z} \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 \\ k & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_{y+z} \end{bmatrix} + \begin{bmatrix} k \\ k(k+1)/2 \end{bmatrix}
$$

$$
\exists k. \left(\begin{array}{l} k \geq 0 \\ \wedge x' = x + k \\ \wedge (y' + z') = (y + z) + kx + k(k+1)/2 \end{array} \right)
$$

The recipe

Parameters

1. Class of abstract machines M
   - Affine maps
   - Solvable polynomials maps [Rodríguez-Carbonell & Kapur '04]
   - Difference bound relations [Comon & Jurski '98]
   - Octagonal relations [Bozga et al '09]
   - Integer vector addition systems [Haase & Halfon '14]
   - ...
2. Class of simulations S

The recipe

Parameters

1. Class of abstract machines M
2. Class of simulations S
    - Identity relation
    - Linear relations
    - Polyhedral relations
    - ...

The recipe

Parameters

1. Class of abstract machines M
2. Class of simulations S

Recipe:

1. Define closure operator $c\ell : M \rightarrow TF$
2. Define abstraction function $\alpha : TF \rightarrow M$
   - For any F, identify a simulation $s_F \in S$ between TF and $\alpha(F)$
3. Take $F^{\circledast} \triangleq \forall \vec{y}. s_F(\vec{x}, \vec{y}) \Rightarrow \exists \vec{y}'. c\ell(\alpha(F))(\vec{y}, \vec{y}') \wedge s_F(\vec{x}', \vec{y}')$.

A view from category theory

Fixing a class of simulations S and a class of abstract machines M, form two categories

- **TF**: transition formulas
  - Objects are transition formulas, arrows are simulations in S

A view from category theory

Fixing a class of simulations S and a class of abstract machines M, form two categories

- **TF**: transition formulas
  - Objects are transition formulas, arrows are simulations in S
- **M**: abstract machines
  - Objects are machines in M, arrows are simulations in S

A view from category theory

Fixing a class of simulations S and a class of abstract machines M, form two categories

- **TF**: transition formulas
  - Objects are transition formulas, arrows are simulations in S
- **M**: abstract machines
  - Objects are machines in M, arrows are simulations in S

Order theory	Category theory
Poset	Category
$x \leq y$	$S : x \to y$

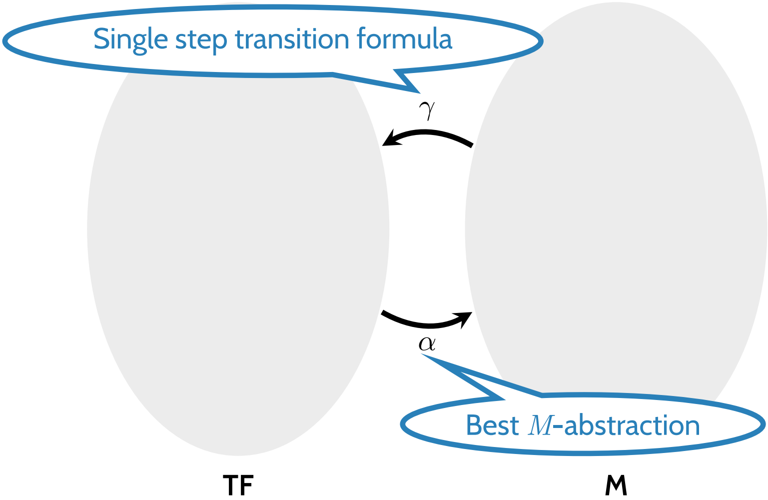
A view from category theory

Fixing a class of simulations S and a class of abstract machines M, form two categories

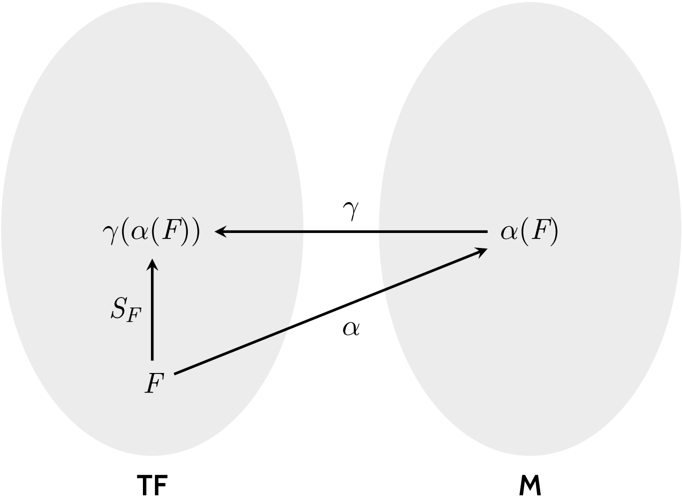
- **TF**: transition formulas
  - Objects are transition formulas, arrows are simulations in S
- **M**: abstract machines
  - Objects are machines in M, arrows are simulations in S

Order theory	Category theory
Poset	Category
$x \leq y$	$S : x \rightarrow y$
monotone function	functor
Galois connection	adjoint functors

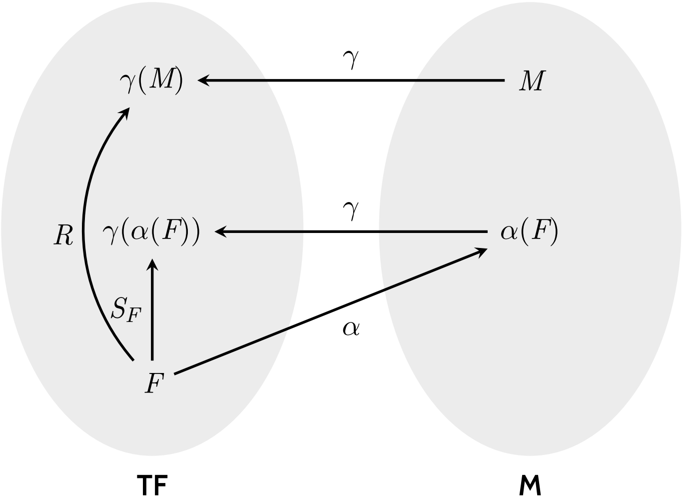
Best abstractions



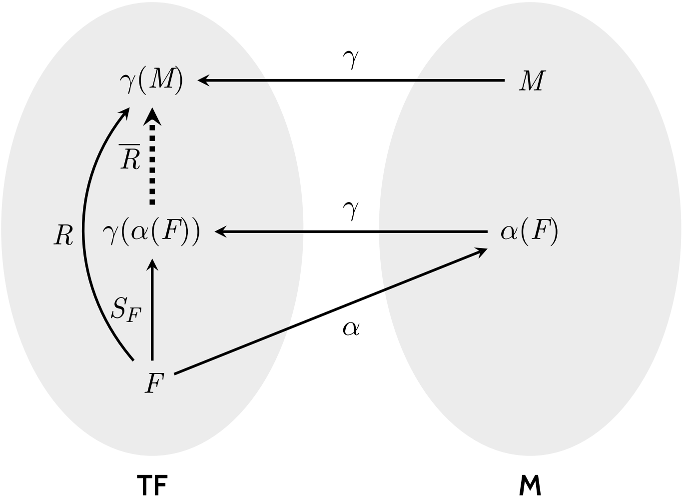
Best abstractions



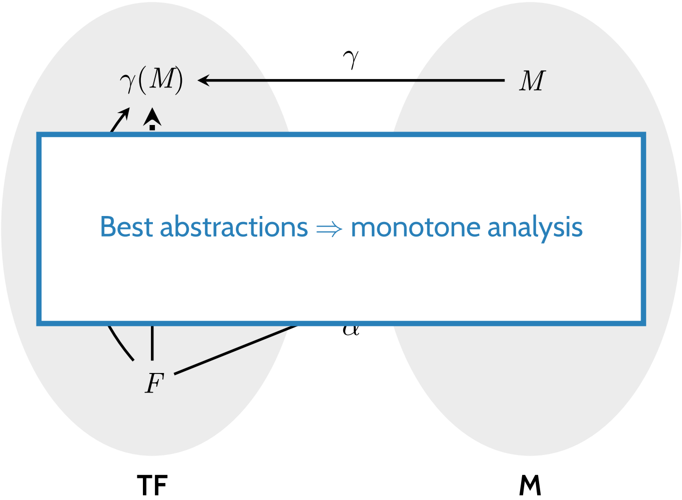
Best abstractions



Best abstractions



Best abstractions



Best abstractions ⇒ monotone analysis

Example: Cartesian relations

- Abstract machines: transition relations of the form $P \times Q$

Example: Cartesian relations

- Abstract machines: transition relations of the form $P \times Q$
- Closure: $c\ell(P, Q) \triangleq \vec{y} = \vec{y}\,' \vee \big(P(\vec{y}) \wedge Q(\vec{y}\,')\big)$

Example: Cartesian relations

- Abstract machines: transition relations of the form $P \times Q$
- Closure: $c\ell(P, Q) \triangleq \vec{y} = \vec{y}\,' \vee \big(P(\vec{y}) \wedge Q(\vec{y}\,')\big)$
- Simulations: identity relations

Example: Cartesian relations

- Abstract machines: transition relations of the form $P \times Q$
- Closure: $c\ell(P, Q) \triangleq \vec{y} = \vec{y}\,' \vee \big(P(\vec{y}) \wedge Q(\vec{y}\,')\big)$
- Simulations: identity relations
- Best abstraction: $\alpha(F) \triangleq (\exists \vec{x}'.F, \exists \vec{x}.F)$

Example: lossy sums

- Abstract machines: $f(\vec{y}) = \vec{y} + \vec{b}$

Example: lossy sums

- Abstract machines: $f(\vec{y}) = \vec{y} + \vec{b}$
- Closure: $c\ell(f) \triangleq \exists k. k \geq 0 \land \vec{y}' = \vec{y} + k\vec{b}$

Example: lossy sums

- Abstract machines: $f(\vec{y}) = \vec{y} + \vec{b}$
- Closure: $c\ell(f) \triangleq \exists k. k \geq 0 \wedge \vec{y}' = \vec{y} + k\vec{b}$
- Simulations: $\vec{y} \leq A\vec{x}$

Example: lossy sums

- Abstract machines: $f(\vec{y}) = \vec{y} + \vec{b}$
- Closure: $c\ell(f) \triangleq \exists k. k \geq 0 \wedge \vec{y}' = \vec{y} + k\vec{b}$
- Simulations: $\vec{y} \leq A\vec{x}$
- Best abstraction $\alpha(F)$:
  - Compute convex hull $A\vec{\delta} \leq \vec{b}$ of $\exists \vec{x}, \vec{x}'. F \wedge (\vec{\delta} = \vec{x}' - \vec{x})$.
  - $\vec{y} \leq A\vec{x}$ is a simulation between F and $f(\vec{y}) = \vec{y} + \vec{b}$.

Example: unit spectrum affine maps

- Abstract machines: $f(\vec{y}) = A\vec{y} + \vec{b}$, only eigenvalue of A is 1

Example: unit spectrum affine maps

- Abstract machines: $f(\vec{y}) = A\vec{y} + \vec{b}$, only eigenvalue of A is 1
- Closure: via Jordan Normal Form, expressed in polynomial arithmetic

Example: unit spectrum affine maps

- Abstract machines: $f(\vec{y}) = A\vec{y} + \vec{b}$, only eigenvalue of A is 1
- Closure: via Jordan Normal Form, expressed in polynomial arithmetic
- Simulations: $\vec{y} = A\vec{x}$

Example: unit spectrum affine maps

- Abstract machines: $f(\vec{y}) = A\vec{y} + \vec{b}$, only eigenvalue of A is 1
- Closure: via Jordan Normal Form, expressed in polynomial arithmetic
- Simulations: $\vec{y} = A\vec{x}$
- Best abstraction:
  1. Extract affine hull of F using an SMT solver
  2. Linear algebra tricks to put equations in the correct form

Example: solvable polynomial maps

- Abstract machines: polynomial maps without non-linear circular dependencies
  - $f(x, y) = (x + y, x - y)$: ✓
  - $g(x, y) = (x + y^2, y + x^2)$: ✗

Example: solvable polynomial maps

- Abstract machines: polynomial maps without non-linear circular dependencies
  - $f(x, y) = (x + y, x - y)$: ✓
  - $g(x, y) = (x + y^2, y + x^2)$: ✗
- Closure: via Berg's operational calculus
  - Polynomials + exponentials + operators

Example: solvable polynomial maps

- Abstract machines: polynomial maps without non-linear circular dependencies
  - $f(x, y) = (x + y, x - y)$: ✓
  - $g(x, y) = (x + y^2, y + x^2)$: ✗
- Closure: via Berg's operational calculus
  - Polynomials + exponentials + operators
- Simulations: $\vec{y} = A\vec{x}$

Example: solvable polynomial maps

- Abstract machines: polynomial maps without non-linear circular dependencies
  - $f(x, y) = (x + y, x - y)$: ✓
  - $g(x, y) = (x + y^2, y + x^2)$: ✗
- Closure: via Berg's operational calculus
  - Polynomials + exponentials + operators
- Simulations: $\vec{y} = A\vec{x}$
- Best abstractions not computable – non-linear arithmetic
  - Heuristics based on Gröbner bases, congruence closure, polyhedra

Compositional recurrence analysis

- [FMCAD'15]: reduced product of
  - cartesian relations
  - unit spectrum affine maps
  - lossy sums

- [POPL'18]: reduced product of
  - cartesian relations
  - solvable polynomial maps
  - lossy sums

Summary

Recipe for putting abstract machines to work in abstract interpreters

- Compositional
- Precise
- Predictable

Summary

Recipe for putting abstract machines to work in abstract interpreters

- Compositional
- Precise
- Predictable

Lots of room to work in this space

- Invent new abstract machines
- Develop abstraction procedures

Thanks!

- Farzan, Kincaid: Compositional Recurrence Analysis. FMCAD'15
- Kincaid, Breck, Boroujeni, Reps. Compositional Recurrence Analysis Revisited. PLDI'17
- Kincaid, Breck, Cyphert, Reps. Non-linear Reasoning for Invariant Synthesis. POPL'18