

Compositional Recurrence Analysis

Azadeh Farzan

Zachary Kincaid

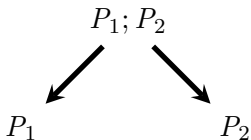
University of Toronto

September 28, 2015

Compositional program analysis

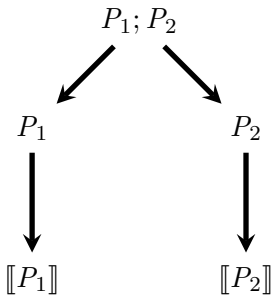
$$P_1; P_2$$

Compositional program analysis



Break program into parts

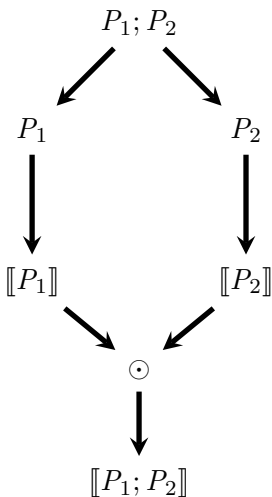
Compositional program analysis



Break program into parts

Analyze each part

Compositional program analysis

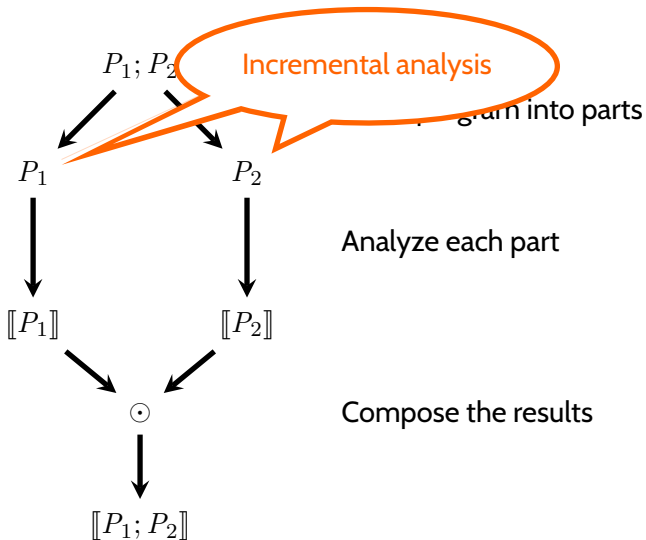


Break program into parts

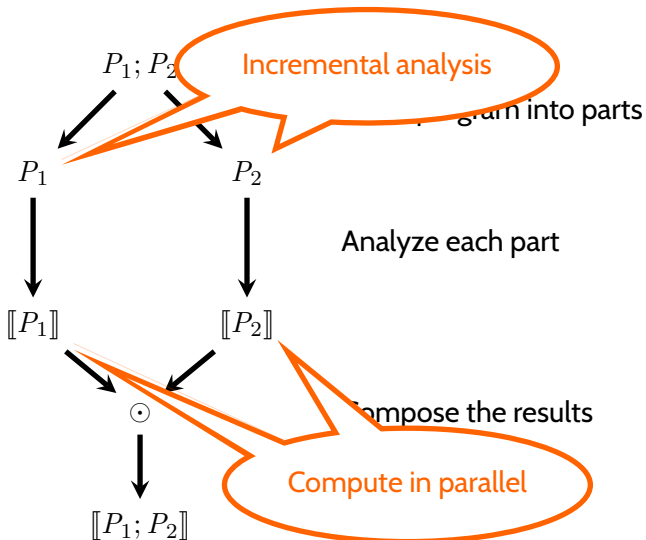
Analyze each part

Compose the results

Compositional program analysis



Compositional program analysis



Context

```
x := 0  
c := 1  
n := 100  
while(x < n):  
    x := x + c  
assert(x == n)
```


Context

`x := 0`

`c := 1`

`n := 100`

`while(x < n):`

`x := x + c`

`assert(x == n)`

100

1

100

Context

`x := 0`

`c := 1`

`n := 100`

while(`x < n`):

`x := x + c`

assert(`x == n`)


$$c = 1 \wedge n = 100 \wedge 0 \leq x \leq 100$$

Context

`x := 0`

`c := 1`

`n :=`

$\exists k. ((k \geq 1 \wedge x < n) \vee k = 0) \wedge x' = x + kc...$

while(`x < n`):

`x := x + c`

assert(`x == n`)

*How can we analyze programs
compositionally and precisely?*

Recurrence Analysis

```
while(*):  
    x := x + 1  
    y := y - 2
```

Recurrence Analysis

while(*) :

$x := x + 1$

$y := y - 2$

Recurrences:

$$x^{(k)} = x^{(k-1)} + 1$$

$$y^{(k)} = y^{(k-1)} - 2$$

Recurrence Analysis

while(*) :

$x := x + 1$

$y := y - 2$

Recurrences:

$$x^{(k)} = x^{(k-1)} + 1$$

$$y^{(k)} = y^{(k-1)} - 2$$

Closed forms:

$$x^{(k)} = x^{(0)} + 1k$$

$$y^{(k)} = y^{(0)} - 2k$$

Recurrence Analysis

while(*) :

$x := x + 1$

$y := y - 2$

Recurrences:

$$x^{(k)} = x^{(k-1)} + 1$$

$$y^{(k)} = y^{(k-1)} - 2$$

Closed forms:

$$x^{(k)} = x^{(0)} + 1k$$

$$y^{(k)} = y^{(0)} - 2k$$

Loop abstraction:

$$\exists k. k \geq 0 \wedge x' = x + k \wedge y' = y - 2k$$


```
while(x + y < 10):  
    z := z + 1  
    if (*):  
        x := x + rand(1,3)  
    else  
        y := y + 1
```

```
while(z < 100):  
    x := 0  
    y := 0  
    while(x + y < 10):  
        z := z + 1  
        if (*):  
            x := x + rand(1,3)  
        else  
            y := y + 1  
    w := w + x
```

*How can we use recurrence analysis to compute
approximations of arbitrary programs?*

Compositional Recurrence Analysis

Algebraic Program Analysis [Tarjan '81]

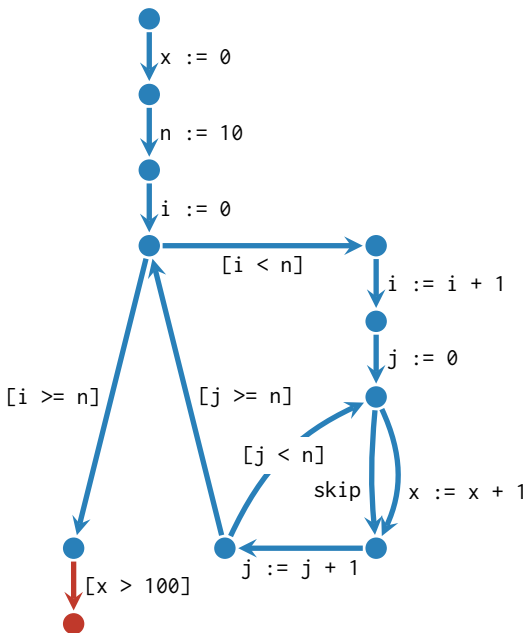
- 1 Compute a *path expression* to a point of interest (e.g., an assertion)
- 2 Evaluate the path expression in the *semantic algebra* defining the analysis

```
x := 0
n := 10
i := 0
outer: if(i >= n):
    goto end
    i := i + 1
inner: j := 0
    if(*):
        x := x + 1
        j := j + 1
        if(j < n):
            goto inner
        goto outer
end: assert(x <= 100)
```

```

x := 0
n := 10
i := 0
outer: if(i >= n):
    goto end
    i := i + 1
inner: j := 0
    if(*):
        x := x + 1
        j := j + 1
        if(j < n):
            goto inner
        goto outer
    assert(x <= 100)
end:

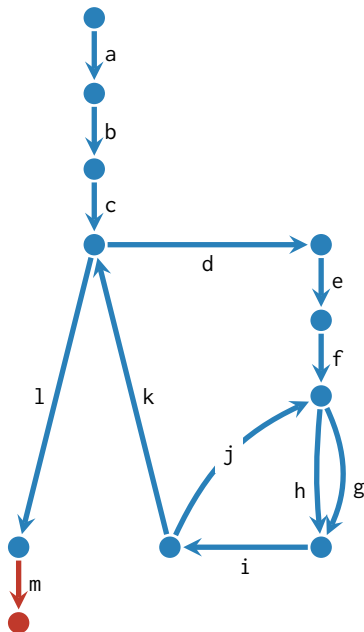
```



```

x := 0
n := 10
i := 0
outer: if(i >= n):
    goto end
    i := i + 1
inner: j := 0
    if(*):
        x := x + 1
        j := j + 1
        if(j < n):
            goto inner
        goto outer
    assert(x <= 100)
end:

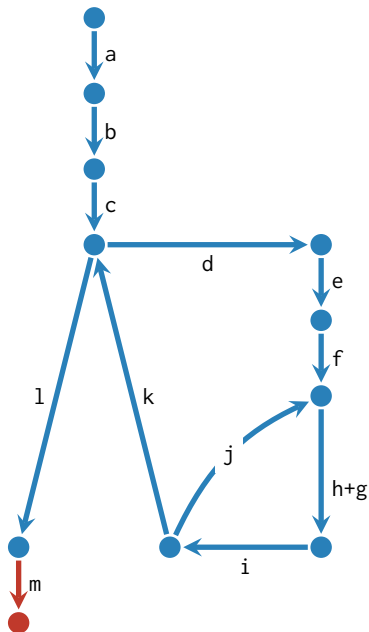
```




```

x := 0
n := 10
i := 0
outer: if(i >= n):
    goto end
    i := i + 1
inner: j := 0
    if(*):
        x := x + 1
        j := j + 1
        if(j < n):
            goto inner
        goto outer
    assert(x <= 100)
end:

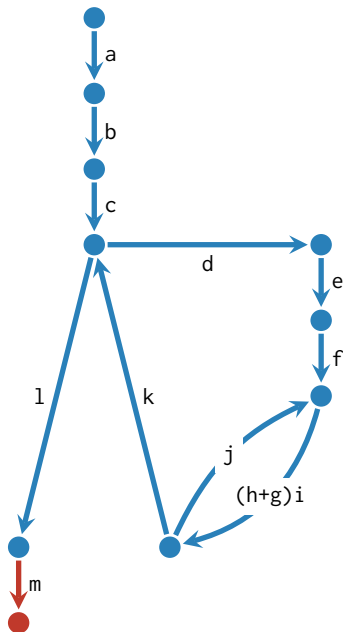
```



```

x := 0
n := 10
i := 0
outer: if(i >= n):
    goto end
    i := i + 1
inner: j := 0
    if(*):
        x := x + 1
        j := j + 1
        if(j < n):
            goto inner
        goto outer
    assert(x <= 100)
end:

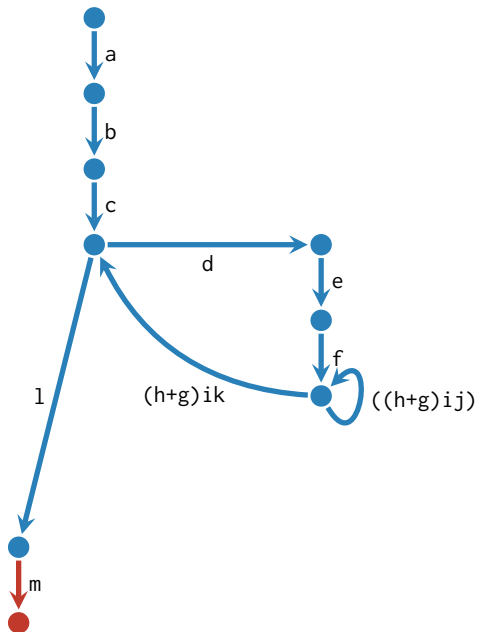
```



```

x := 0
n := 10
i := 0
outer: if(i >= n):
    goto end
    i := i + 1
inner: j := 0
    if(*):
        x := x + 1
        j := j + 1
        if(j < n):
            goto inner
        goto outer
end: assert(x <= 100)

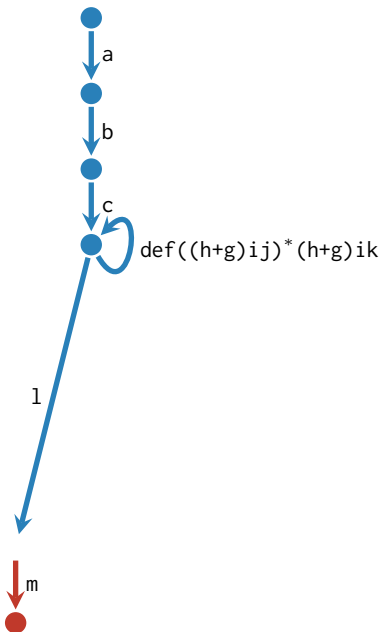
```



```

x := 0
n := 10
i := 0
outer: if(i >= n):
    goto end
    i := i + 1
inner: j := 0
    if(*):
        x := x + 1
        j := j + 1
        if(j < n):
            goto inner
        goto outer
end: assert(x <= 100)

```



```
x := 0
n := 10
i := 0
outer: if(i >= n):
    goto end
    i := i + 1
inner: j := 0
    if(*):
        x := x + 1
        j := j + 1
        if(j < n):
            goto inner
        goto outer
end: assert(x <= 100)
```

abc(def((h+g)ij)*(h+g)ik)*lm

Path expression:

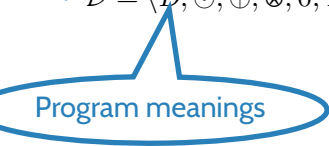
Regular expression over alphabet of control flow edges

Interpretation: $\mathcal{I} = \langle \mathcal{D}, \llbracket \cdot \rrbracket \rangle$

- $\mathcal{D} = \langle D, \odot, \oplus, \otimes, 0, 1 \rangle$ is a *semantic algebra*

Interpretation: $\mathcal{I} = \langle \mathcal{D}, \llbracket \cdot \rrbracket \rangle$

- $\mathcal{D} = \langle D, \odot, \oplus, \otimes, 0, 1 \rangle$ is a *semantic algebra*



Program meanings

Composition operators

Interpretation: $\mathcal{I} = \langle \mathcal{D}, \llbracket \cdot \rrbracket \rangle$

- $\mathcal{D} = \langle D, \overbrace{\odot, \oplus, \otimes}, 0, 1 \rangle$ is a *semantic algebra*

Program meanings

Interpretation: $\mathcal{I} = \langle \mathcal{D}, \llbracket \cdot \rrbracket \rangle$

- $\mathcal{D} = \langle D, \odot, \oplus, \otimes, 0, 1 \rangle$ is a *semantic algebra*
- $\llbracket \cdot \rrbracket : \text{Control flow edges} \rightarrow D$ is a *semantic function*

Interpretation: $\mathcal{I} = \langle \mathcal{D}, \llbracket \cdot \rrbracket \rangle$

- $\mathcal{D} = \langle D, \odot, \oplus, \otimes, 0, 1 \rangle$ is a *semantic algebra*
- $\llbracket \cdot \rrbracket : \text{Control flow edges} \rightarrow D$ is a *semantic function*

$$\begin{aligned} \llbracket abc(def((h+g)ij)^*(h+g)ik)^*lm \rrbracket = & \llbracket a \rrbracket \odot \llbracket b \rrbracket \odot \llbracket c \rrbracket \\ & \odot \left(\llbracket d \rrbracket \odot \llbracket e \rrbracket \odot \llbracket f \rrbracket \right. \\ & \quad \odot \left((\llbracket h \rrbracket \oplus \llbracket g \rrbracket) \odot \llbracket i \rrbracket \odot \llbracket j \rrbracket \right)^{\otimes} \\ & \quad \left. \odot (\llbracket h \rrbracket \oplus \llbracket g \rrbracket) \odot \llbracket i \rrbracket \odot \llbracket k \rrbracket \right)^{\otimes} \\ & \odot \llbracket l \rrbracket \odot \llbracket m \rrbracket \end{aligned}$$

Interpretation: $\mathcal{I} = \langle \mathcal{D}, \llbracket \cdot \rrbracket \rangle$

- $\mathcal{D} = \langle D, \odot, \oplus, \otimes, 0, 1 \rangle$ is a *semantic algebra*
- $\llbracket \cdot \rrbracket : \text{Control flow edges} \rightarrow D$ is a *semantic function*

Compositional Recurrence Analysis

- D : set of arithmetic *transition formulas*

$$\llbracket x := x + 1 \rrbracket \triangleq x' = x + 1 \wedge y' = y \wedge i' = i \wedge j' = j \wedge n' = n$$

Interpretation: $\mathcal{I} = \langle \mathcal{D}, \llbracket \cdot \rrbracket \rangle$

- $\mathcal{D} = \langle D, \odot, \oplus, \otimes, 0, 1 \rangle$ is a *semantic algebra*
- $\llbracket \cdot \rrbracket : \text{Control flow edges} \rightarrow D$ is a *semantic function*

Compositional Recurrence Analysis

- D : set of arithmetic *transition formulas*

$$\llbracket x := x + 1 \rrbracket \triangleq x' = x + 1 \wedge y' = y \wedge i' = i \wedge j' = j \wedge n' = n$$

- $\varphi \odot \psi \triangleq \exists \vec{x}''. \varphi[\vec{x}' \mapsto \vec{x}''] \wedge \psi[\vec{x} \mapsto \vec{x}'']$

Interpretation: $\mathcal{I} = \langle \mathcal{D}, \llbracket \cdot \rrbracket \rangle$

- $\mathcal{D} = \langle D, \odot, \oplus, \otimes, 0, 1 \rangle$ is a *semantic algebra*
- $\llbracket \cdot \rrbracket : \text{Control flow edges} \rightarrow D$ is a *semantic function*

Compositional Recurrence Analysis

- D : set of arithmetic *transition formulas*

$$\llbracket x := x + 1 \rrbracket \triangleq x' = x + 1 \wedge y' = y \wedge i' = i \wedge j' = j \wedge n' = n$$

- $\varphi \odot \psi \triangleq \exists \vec{x}''. \varphi[\vec{x}' \mapsto \vec{x}''] \wedge \psi[\vec{x} \mapsto \vec{x}'']$
- $\varphi \oplus \psi \triangleq \varphi \vee \psi$

Interpretation: $\mathcal{I} = \langle \mathcal{D}, \llbracket \cdot \rrbracket \rangle$

- $\mathcal{D} = \langle D, \odot, \oplus, \otimes, 0, 1 \rangle$ is a *semantic algebra*
- $\llbracket \cdot \rrbracket$: Control flow edges $\rightarrow D$ is a *semantic function*

Compositional Recurrence Analysis

- D : set of arithmetic *transition formulas*

$$\llbracket x := x + 1 \rrbracket \triangleq x' = x + 1 \wedge y' = y \wedge i' = i \wedge j' = j \wedge n' = n$$

- $\varphi \odot \psi \triangleq \exists \vec{x}''. \varphi[\vec{x}' \mapsto \vec{x}''] \wedge \psi[\vec{x} \mapsto \vec{x}'']$
- $\varphi \oplus \psi \triangleq \varphi \vee \psi$
- $\varphi^{\otimes} \triangleq \dots$

$$[[p^*]] = [[p]]^{\oplus}$$

Problem

Given a transition formula φ (representing the body of a loop), compute a formula $\varphi^{\oplus}$ representing any number of iterations of the loop.

Problem

Given a transition formula φ (representing the body of a loop), compute a formula $\varphi^{\oplus}$ representing any number of iterations of the loop.

First, *linearize* φ : compute a *linear* formula $\text{lin}(\varphi)$ such that $\varphi \models \text{lin}(\varphi)$.

Problem

Given a transition formula φ (representing the body of a loop), compute a formula $\varphi^{\oplus}$ representing any number of iterations of the loop.

First, *linearize* φ : compute a *linear* formula $\text{lin}(\varphi)$ such that $\varphi \models \text{lin}(\varphi)$.

Linearization via *optimization modulo theories*:

If $\varphi \models x \in [1, 10]$ and $y \in [2, 3]$, then

$$\varphi \models y \leq xy \leq 10y \wedge 2x \leq xy \leq 3x$$

Simple recurrences

```
while(*):  
    c := 2 * x  
    if (c = 1):  
        x := x + 2  
    else  
        x := x + 1  
    y := y - 2
```

Simple recurrences

while(*):

 c := 2 * x

if (c = 1):

 x := x + 2

else

 x := x + 1

y := y - 2

$c' = 2x$

$\wedge ((c' = 1$

$\wedge x' = x + 2)$

$\vee (c' \neq 1$

$\wedge x' = x + 1))$

$\wedge y' = y - 2$

Simple recurrences

while(*):

c := 2 * **x**

if (**c** = 1):

x := **x** + 2

else

x := **x** + 1

y := **y** - 2

c' = 2**x**

$\wedge ((\mathbf{c}' = 1$

$\wedge \mathbf{x}' = \mathbf{x} + 2)$

$\vee (\mathbf{c}' \neq 1$

$\wedge \mathbf{x}' = \mathbf{x} + 1))$

$\wedge \mathbf{y}' = \mathbf{y} - 2$

$m : [\mathbf{c} \mapsto 0, \mathbf{x} \mapsto 0, \mathbf{y} \mapsto 0, \mathbf{c}' \mapsto 0, \mathbf{x}' \mapsto 1, \mathbf{y}' \mapsto -2] \models \varphi_{\text{body}}$

Simple recurrences

while(*):

c := 2 * **x**

if (**c** = 1):

x := **x** + 2

else

x := **x** + 1

y := **y** - 2

c' = 2**x**

$\wedge ((\mathbf{c}' = 1$

$\wedge \mathbf{x}' = \mathbf{x} + 2)$

$\vee (\mathbf{c}' \neq 1$

$\wedge \mathbf{x}' = \mathbf{x} + 1))$

$\wedge \mathbf{y}' = \mathbf{y} - 2$

$m : [c \mapsto 0, x \mapsto 0, y \mapsto 0, c' \mapsto 0, x' \mapsto 1, y' \mapsto -2] \models \varphi_{\text{body}}$

$$(c' - c)^m = 0$$

$$(x' - x)^m = 1$$

$$(y' - y)^m = -2$$

Simple recurrences

while(*):

c := 2 * **x**

if (**c** = 1):

x := **x** + 2

else

x := **x** + 1

y := **y** - 2

c' = 2**x**

$\wedge ((\mathbf{c}' = 1$
 $\wedge \mathbf{x}' = \mathbf{x} + 2)$

$\vee (\mathbf{c}' \neq 1$
 $\wedge \mathbf{x}' = \mathbf{x} + 1))$

$\wedge \mathbf{y}' = \mathbf{y} - 2$

$m : [c \mapsto 0, x \mapsto 0, y \mapsto 0, c' \mapsto 0, x' \mapsto 1, y' \mapsto -2] \models \varphi_{\text{body}}$

$$(c' - c)^m = 0$$

$$(x' - x)^m = 1$$

$$(y' - y)^m = -2$$

$$\varphi_{\text{body}} \models c' = c + 0?$$

$$\varphi_{\text{body}} \models x' = x + 1?$$

$$\varphi_{\text{body}} \models y' = y - 2?$$

Simple recurrences

while(*):

c := 2 * **x**

if (**c** = 1):

x := **x** + 2

else

x := **x** + 1

y := **y** - 2

c' = 2**x**

$\wedge ((\mathbf{c}' = 1$

$\wedge \mathbf{x}' = \mathbf{x} + 2)$

$\vee (\mathbf{c}' \neq 1$

$\wedge \mathbf{x}' = \mathbf{x} + 1))$

$\wedge \mathbf{y}' = \mathbf{y} - 2$

$m : [c \mapsto 0, x \mapsto 0, y \mapsto 0, c' \mapsto 0, x' \mapsto 1, y' \mapsto -2] \models \varphi_{\text{body}}$

$$(c' - c)^m = 0$$

$$(x' - x)^m = 1$$

$$(y' - y)^m = -2$$

$$\varphi_{\text{body}} \not\models c' = c + 0$$

$$\varphi_{\text{body}} \models x' = x + 1$$

$$\varphi_{\text{body}} \models y' = y - 2$$

Stratified recurrences

while(*) :

$x := x + 1$

$y := y + x$

$z := z + y$

Linear recurrences (in)equations

while($0 \leq i < 100$):

$$0 \leq i \wedge i < 100$$

if (*):

$$x := x + i$$

$$\wedge x' = x + i$$

else

$$y := y + i$$

$$\wedge y' = y + i$$

$$i := i + 1$$

$$\wedge i' = i + 1$$

Linear recurrences (in)equations

while ($0 \leq i < 100$):	$0 \leq i \wedge i < 100$
if (*):	
$x := x + i$	$\wedge x' = x + i$
else	
$y := y + i$	$\wedge y' = y + i$
$i := i + 1$	$\wedge i' = i + 1$

- 1 Introduce *difference variables* for non-induction variables:

$$\psi \triangleq \varphi_{\text{body}} \wedge \delta_x = x' - x \wedge \delta_y = y' - y$$

Linear recurrences (in)equations

while ($0 \leq i < 100$):	$0 \leq i \wedge i < 100$
if (*):	
$x := x + i$	$\wedge x' = x + i$
else	
$y := y + i$	$\wedge y' = y + i$
$i := i + 1$	$\wedge i' = i + 1$

- 1 Introduce *difference variables* for non-induction variables:

$$\psi \triangleq \varphi_{\text{body}} \wedge \delta_x = x' - x \wedge \delta_y = y' - y$$

- 2 Project + compute the *convex hull*:
 - Smallest polyhedron P such that $\exists x, y, x', y', i'. \psi \models P$

Linear recurrences (in)equations

while ($0 \leq i < 100$):	$0 \leq i \wedge i < 100$
if (*):	
$x := x + i$	$\wedge x' = x + i$
else	
$y := y + i$	$\wedge y' = y + i$
$i := i + 1$	$\wedge i' = i + 1$

$$\delta_x + \delta_y = i$$

$$0 \leq \delta_x \leq i$$

$$0 \leq \delta_y \leq i$$

...

Linear equations over δ 's and induction variables

Linear recurrences (in)equations

while ($0 \leq i < 100$):	$0 \leq i \wedge i < 100$
if (*):	
$x := x + i$	$\wedge x' = x + i$
else	
$y := y + i$	$\wedge y' = y + i$
$i := i + 1$	$\wedge i' = i + 1$

$\delta_x + \delta_y = i$		
$0 \leq \delta_x \leq i$		
$0 \leq \delta_y \leq i$		$(x' - x) + (y' - y) = i$
...		$0 \leq (x' - x) \leq i$
		$0 \leq (y' - y) \leq i$

Linear recurrences (in)equations

while($0 \leq i < 100$): $0 \leq i \wedge i < 100$
 if (*): $\wedge x' = x + i$
 $x := x + i$
 else $\wedge y' = y + i$
 $y := y + i$
 $i := i + 1$ $\wedge i' = i + 1$

$$\begin{array}{l} \delta_x + \delta_y = i \\ 0 \leq \delta_x \leq i \\ 0 \leq \delta_y \leq i \\ \dots \end{array} \quad \longrightarrow \quad \begin{array}{l} (x' - x) + (y' - y) = i \\ 0 \leq (x' - x) \leq i \\ 0 \leq (y' - y) \leq i \end{array} \quad \longrightarrow \quad \begin{array}{l} x' + y' = x + y + i \\ x \leq x' \leq x + i \\ x \leq y' \leq y + i \end{array}$$

Linear recurrences (in)equations

while($0 \leq i < 100$): $0 \leq i \wedge i < 100$
 if (*):
 $x := x + i$ $\wedge x' = x + i$
 else
 $y := y + i$ $\wedge y' = y + i$
 $i := i + 1$ $\wedge i' = i + 1$

$$\begin{array}{l} \delta_x + \delta_y = i \\ 0 \leq \delta_x \leq i \\ 0 \leq \delta_y \leq i \\ \dots \end{array} \quad \longrightarrow \quad \begin{array}{l} (x' - x) + (y' - y) = i \\ 0 \leq (x' - x) \leq i \\ 0 \leq (y' - y) \leq i \end{array} \quad \longrightarrow \quad \begin{array}{l} x' + y' = x + y + i \\ x \leq x' \leq x + i \\ x \leq y' \leq y + i \end{array}$$

$$x^{(k)} + y^{(k)} = x^{(0)} + y^{(0)} + ki^{(0)} + k(k+1)/2$$

$$x^{(0)} \leq x^{(k)} \leq x^{(0)} + ki^{(0)} + k(k+1)/2$$

$$x^{(0)} \leq y^{(k)} \leq y^{(0)} + ki^{(0)} + k(k+1)/2$$

Putting it all together

$$\varphi_{\text{body}} \models \bigwedge_r \sum_i a_{ri} x'_{ri} \leq \sum_i a_{ri} x_{ri} + \sum_j b_{rj} y_{rj} + c_r$$



Extracted recurrences

Putting it all together

$$\varphi_{\text{body}} \models \bigwedge_r \sum_i a_{ri} x'_{ri} \leq \sum_i a_{ri} x_{ri} + \sum_j b_{rj} y_{rj} + c_r$$

$$\sum_{ri} a_{ri} x_{ri}^{(k)} \leq \sum_{ri} a_{ri} x_{ri}^{(0)} + \sum_j p_{rj}(k) y_{rj}^{(0)} + k c_r$$

Closed form

Putting it all together

$$\varphi_{\text{body}} \models \bigwedge_r \sum_i a_{ri} \mathbf{x}'_{ri} \leq \sum_i a_{ri} \mathbf{x}_{ri} + \sum_j b_{rj} \mathbf{y}_{rj} + c_r$$

$$\sum_{ri} a_{ri} \mathbf{x}_{ri}^{(k)} \leq \sum_{ri} a_{ri} \mathbf{x}_{ri}^{(0)} + \sum_j p_{rj}(k) \mathbf{y}_{rj}^{(0)} + k c_r$$

$$\varphi_{\text{body}}^{\oplus} \triangleq \bigwedge_i \mathbf{x}'_i = \mathbf{x}_i$$

$$\vee (\exists k. k \geq 1 \wedge (\exists \vec{x}'. \varphi_{\text{body}}) \wedge (\exists \vec{x}. \varphi_{\text{body}}))$$

$$\wedge \bigwedge_r \sum_i a_{ri} \mathbf{x}'_{ri} \leq \sum_i a_{ri} \mathbf{x}_{ri} + \sum_j p_{rj}(k) \mathbf{y}_{rj} + k c_r$$

Experimental evaluation on

- 74 safe benchmarks from SVComp15
- 7 safe non-linear benchmarks

Tool	Solved
CRA	65%
CRA+Oct	88%
SeaHorn	85%
CPAChecker	47%

Summary

CRA is *compositional* yet *precise*

Summary

CRA is *compositional* yet *precise*

Compositional analysis
+ SMT-based recurrence detection

Approximate recurrence analysis for *arbitrary loops*