Compositional Recurrence Analysis

Azadeh Farzan **Zachary Kincaid**

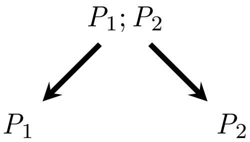
University of Toronto

September 28, 2015

Compositional program analysis

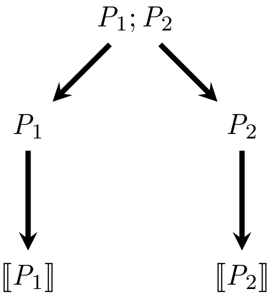
$P_1; P_2$

Compositional program analysis



Break program into parts

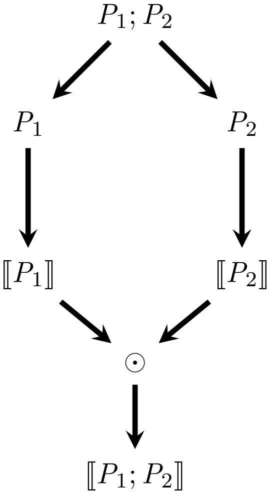
Compositional program analysis



Break program into parts

Analyze each part

Compositional program analysis

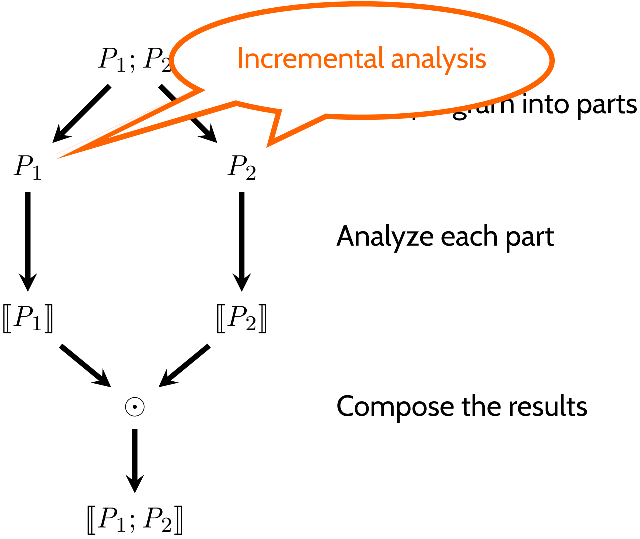


Break program into parts

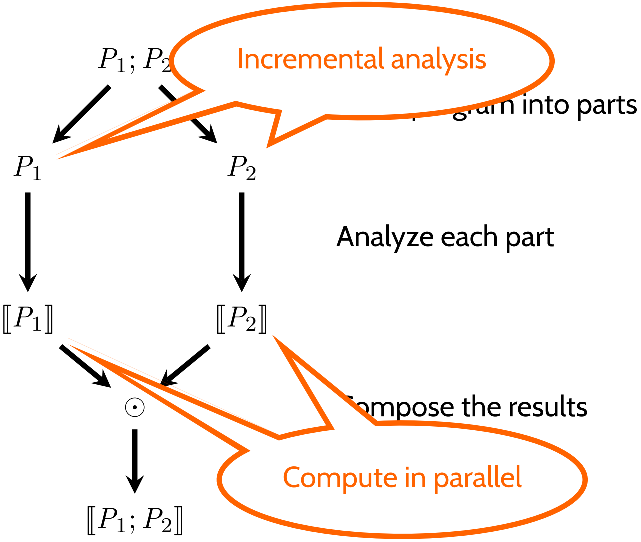
Analyze each part

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)
```

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

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

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:

$$\mathsf{x}^{(k)} = \mathsf{x}^{(k-1)} + 1$$
$$\mathsf{y}^{(k)} = \mathsf{y}^{(k-1)} - 2$$

Recurrence Analysis

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

Recurrences:

$$\mathsf{x}^{(k)} = \mathsf{x}^{(k-1)} + 1$$
$$\mathsf{y}^{(k)} = \mathsf{y}^{(k-1)} - 2$$

Closed forms:

$$\mathsf{x}^{(k)} = \mathsf{x}^{(0)} + 1k$$
$$\mathsf{y}^{(k)} = \mathsf{y}^{(0)} - 2k$$

Recurrence Analysis

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

Recurrences:

$$\mathsf{x}^{(k)} = \mathsf{x}^{(k-1)} + 1$$
$$\mathsf{y}^{(k)} = \mathsf{y}^{(k-1)} - 2$$

Closed forms:

$$\mathsf{x}^{(k)} = \mathsf{x}^{(0)} + 1k$$
$$\mathsf{y}^{(k)} = \mathsf{y}^{(0)} - 2k$$

Loop abstraction:

$$\exists k. k \geq 0 \land \mathsf{x}' = \mathsf{x} + k \land \mathsf{y}' = \mathsf{y} - 2k$$

```
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?

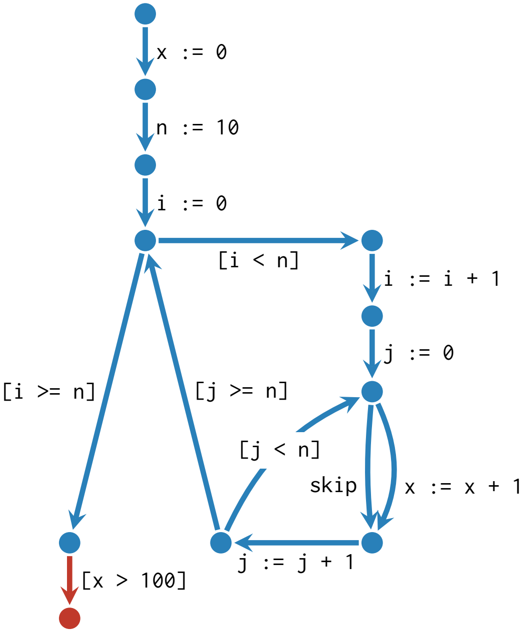
Compositional Recurrence Analysis

Algebraic Program Analysis [Tarjan '81]

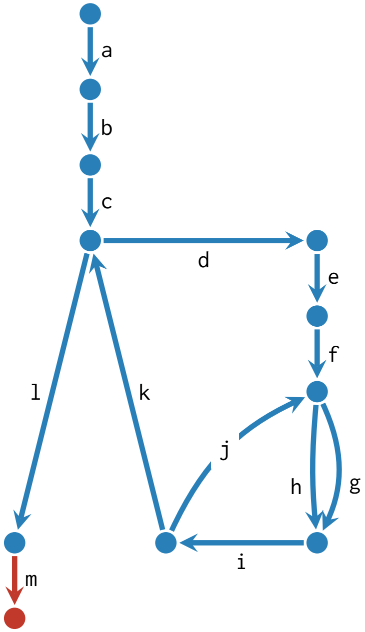
1. Compute a *path expression* to a point of interest (e.g., an assertion)
2. Evaluate the path expression in the *semantic algebra* defining the analysis

```
       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)
```

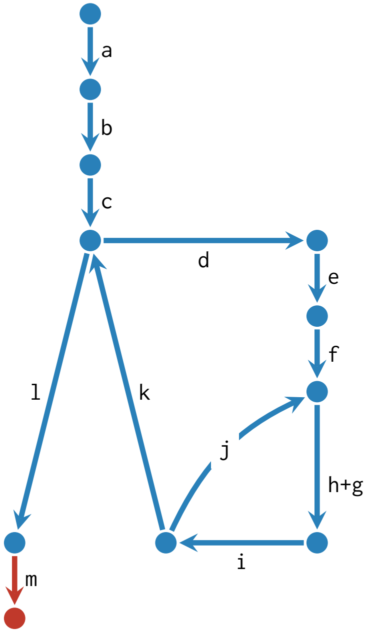
```
        x := 0
        n := 10
        i := 0
outer:  if(i >= n):
          goto end
        i := i + 1
inner:  j := 0
        if(*):
          x := x + 1
        j := j + 1
        if(j < n):
          goto inner
        goto outer
end:    assert(x <= 100)
```



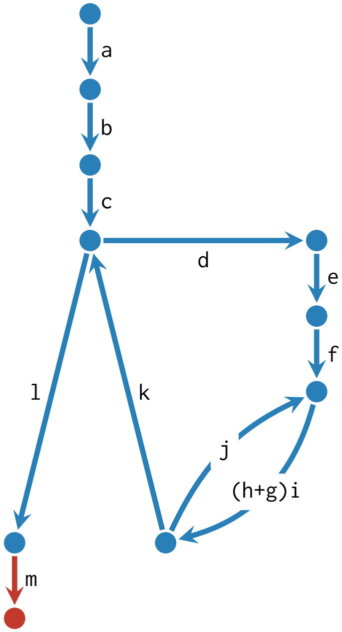
```
        x := 0
        n := 10
        i := 0
outer:  if(i >= n):
           goto end
        i := i + 1
inner:  j := 0
        if(*):
           x := x + 1
        j := j + 1
        if(j < n):
           goto inner
        goto outer
end:    assert(x <= 100)
```



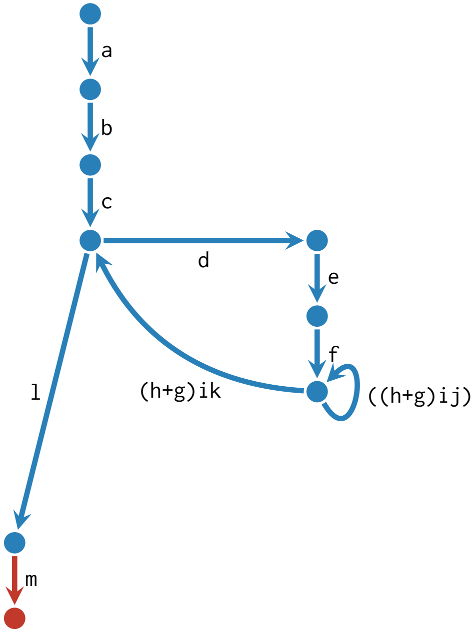
```
       x := 0
       n := 10
       i := 0
outer: if(i >= n):
         goto end
       i := i + 1
inner: j := 0
       if(*):
         x := x + 1
       j := j + 1
       if(j < n):
         goto inner
       goto outer
end:   assert(x <= 100)
```



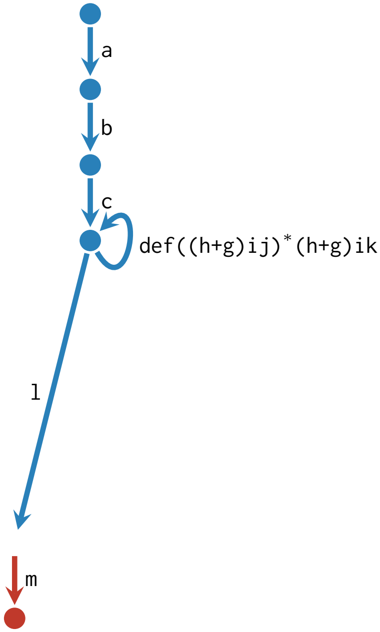
```
       x := 0
       n := 10
       i := 0
outer: if(i >= n):
         goto end
       i := i + 1
inner: j := 0
       if(*):
         x := x + 1
       j := j + 1
       if(j < n):
         goto inner
       goto outer
end:   assert(x <= 100)
```



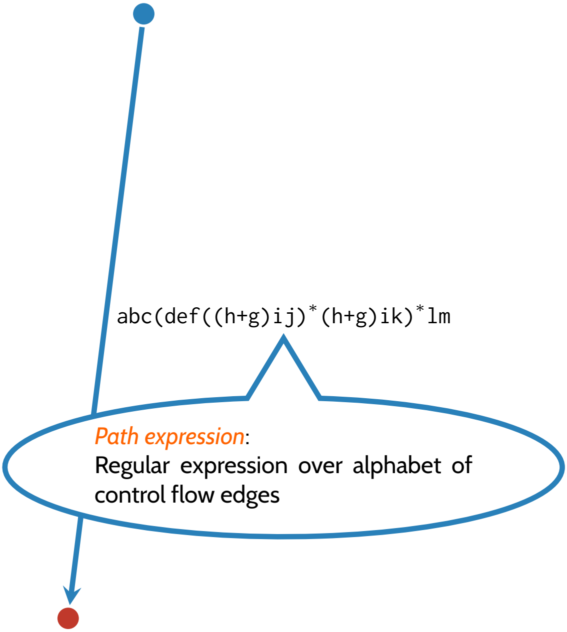
```
        x := 0
        n := 10
        i := 0
outer:  if(i >= n):
          goto end
        i := i + 1
inner:  j := 0
        if(*):
          x := x + 1
        j := j + 1
        if(j < n):
          goto inner
        goto outer
end:    assert(x <= 100)
```



```
       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)
```



```
        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)
```



Interpretation: $\mathscr{I} = \langle \mathcal{D}, [\![\cdot]\!] \rangle$

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

Interpretation: $\mathscr{I} = \langle \mathcal{D}, [\![\cdot]\!] \rangle$

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

Program meanings

Composition operators

Interpretation: $\mathscr{I} = \langle \mathcal{D}, [\![\cdot]\!] \rangle$

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

Program meanings

Interpretation: $\mathscr{I} = \langle \mathcal{D}, [\![\cdot]\!] \rangle$

- $\mathcal{D} = \langle D, \odot, \oplus, \circledast, 0, 1 \rangle$ is a *semantic algebra*
- $[\![\cdot]\!]$: Control flow edges $\rightarrow D$ is a *semantic function*

Interpretation: $\mathscr{I} = \langle \mathcal{D}, [\![\cdot]\!] \rangle$

- $\mathcal{D} = \langle D, \odot, \oplus, \circledast, 0, 1 \rangle$ is a *semantic algebra*
- $[\![\cdot]\!]$: Control flow edges $\to D$ is a *semantic function*

$$
\begin{aligned}
[\![\mathtt{abc(def((h+g)ij)^*(h+g)ik)^*lm}]\!] = {} & [\![\mathtt{a}]\!] \odot [\![\mathtt{b}]\!] \odot [\![\mathtt{c}]\!] \\
& \odot \Big([\![\mathtt{d}]\!] \odot [\![\mathtt{e}]\!] \odot [\![\mathtt{f}]\!] \\
& \qquad \odot \big(([\![\mathtt{h}]\!] \oplus [\![\mathtt{g}]\!]) \odot [\![\mathtt{i}]\!] \odot [\![\mathtt{j}]\!] \big)^{\circledast} \\
& \qquad \odot ([\![\mathtt{h}]\!] \oplus [\![\mathtt{g}]\!]) \odot [\![\mathtt{i}]\!] \odot [\![\mathtt{k}]\!] \Big)^{\circledast} \\
& \odot [\![\mathtt{l}]\!] \odot [\![\mathtt{m}]\!]
\end{aligned}
$$

Interpretation: $\mathscr{I} = \langle \mathcal{D}, [\![\cdot]\!] \rangle$

- $\mathcal{D} = \langle D, \odot, \oplus, \circledast, 0, 1 \rangle$ is a *semantic algebra*
- $[\![\cdot]\!]$: Control flow edges $\to D$ is a *semantic function*

Compositional Recurrence Analysis

- D: set of arithmetic *transition formulas*

$$[\![\texttt{x := x + 1}]\!] \triangleq \texttt{x}' = \texttt{x} + 1 \land \texttt{y}' = \texttt{y} \land \texttt{i}' = \texttt{i} \land \texttt{j}' = \texttt{j} \land \texttt{n}' = \texttt{n}$$

Interpretation: $\mathscr{I} = \langle \mathcal{D}, [\![\cdot]\!] \rangle$

- $\mathcal{D} = \langle D, \odot, \oplus, \circledast, 0, 1 \rangle$ is a *semantic algebra*
- $[\![\cdot]\!]$: Control flow edges $\rightarrow D$ is a *semantic function*

Compositional Recurrence Analysis

- D: set of arithmetic *transition formulas*

$$[\![\texttt{x := x + 1}]\!] \triangleq \texttt{x}' = \texttt{x} + 1 \land \texttt{y}' = \texttt{y} \land \texttt{i}' = \texttt{i} \land \texttt{j}' = \texttt{j} \land \texttt{n}' = \texttt{n}$$

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

Interpretation: $\mathscr{I} = \langle \mathcal{D}, [\![\cdot]\!] \rangle$

- $\mathcal{D} = \langle D, \odot, \oplus, \circledast, 0, 1 \rangle$ is a *semantic algebra*
- $[\![\cdot]\!]$: Control flow edges $\rightarrow D$ is a *semantic function*

Compositional Recurrence Analysis

- D: set of arithmetic *transition formulas*

$$[\![\texttt{x := x + 1}]\!] \triangleq \texttt{x}' = \texttt{x} + 1 \land \texttt{y}' = \texttt{y} \land \texttt{i}' = \texttt{i} \land \texttt{j}' = \texttt{j} \land \texttt{n}' = \texttt{n}$$

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

Interpretation: $\mathscr{I} = \langle \mathcal{D}, [\![\cdot]\!] \rangle$

- $\mathcal{D} = \langle D, \odot, \oplus, \circledast, 0, 1 \rangle$ is a *semantic algebra*
- $[\![\cdot]\!]$: Control flow edges $\rightarrow D$ is a *semantic function*

Compositional Recurrence Analysis

- D: set of arithmetic *transition formulas*

$$[\![\texttt{x := x + 1}]\!] \triangleq \texttt{x}' = \texttt{x} + 1 \land \texttt{y}' = \texttt{y} \land \texttt{i}' = \texttt{i} \land \texttt{j}' = \texttt{j} \land \texttt{n}' = \texttt{n}$$

- $\varphi \odot \psi \triangleq \exists \vec{x}''.\varphi[\vec{x}' \mapsto \vec{x}''] \land \psi[\vec{x} \mapsto \vec{x}'']$
- $\varphi \oplus \psi \triangleq \varphi \lor \psi$
- $\varphi^{\circledast} \triangleq \ldots$

$$[\![p^*]\!] = [\![p]\!]^{\circledast}$$

Problem

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

Problem

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

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

Problem

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

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

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
```

$$
\begin{aligned}
&c' = 2x \\
&\wedge\big((c' = 1 \\
&\qquad \wedge x' = x + 2) \\
&\quad \vee(c' \neq 1 \\
&\qquad \wedge x' = x + 1)\big) \\
&\wedge y' = y - 2
\end{aligned}
$$

Simple recurrences

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

$$
\begin{aligned}
& c' = 2x \\
& \land \big((c' = 1 \\
& \qquad \land x' = x + 2) \\
& \quad \lor (c' \neq 1 \\
& \qquad \land x' = x + 1)\big) \\
& \land y' = y - 2
\end{aligned}
$$

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

Simple recurrences

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

$$\begin{aligned}&c' = 2x\\ &\wedge\big((c' = 1\\ &\qquad \wedge x' = x + 2)\\ &\quad \vee(c' \neq 1\\ &\qquad \wedge x' = x + 1)\big)\\ &\wedge y' = y - 2\end{aligned}$$

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

$$\begin{aligned}(c' - c)^m &= 0\\ (x' - x)^m &= 1\\ (y' - y)^m &= -2\end{aligned}$$

Simple recurrences

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

$$
\begin{aligned}
&c' = 2x \\
&\land\big((c' = 1 \\
&\qquad \land x' = x + 2) \\
&\quad \lor (c' \neq 1 \\
&\qquad \land x' = x + 1)\big) \\
&\land y' = y - 2
\end{aligned}
$$

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

$$(c' - c)^m = 0 \qquad \varphi_{\text{body}} \models \; c' = c + 0?$$

$$(x' - x)^m = 1 \qquad \varphi_{\text{body}} \models \; x' = x + 1?$$

$$(y' - y)^m = -2 \qquad \varphi_{\text{body}} \models \; y' = y - 2?$$

Simple recurrences

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

$$c' = 2x \land \big((c' = 1 \land x' = x + 2) \lor (c' \neq 1 \land x' = x + 1)\big) \land y' = y - 2$$

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

$$(c' - c)^m = 0 \qquad \varphi_{\text{body}} \not\models \; c' = c + 0$$

$$(x' - x)^m = 1 \qquad \varphi_{\text{body}} \models \; x' = x + 1$$

$$(y' - y)^m = -2 \qquad \varphi_{\text{body}} \models \; y' = y - 2$$

Stratified recurrences

```
while(*):
    x := x + 1
    y := y + x
    z := z + y
```

Linear recurrences (in)equations

```
while(0 <= i < 100):
    if (*):
        x := x + i
    else
        y := y + i
    i := i + 1
```

$$0 \leq \mathtt{i} \land \mathtt{i} < 100$$

$$\land \mathtt{x}' = \mathtt{x} + \mathtt{i}$$

$$\land \mathtt{y}' = \mathtt{y} + \mathtt{i}$$

$$\land \mathtt{i}' = \mathtt{i} + 1$$

Linear recurrences (in)equations

```
while(0 <= i < 100):
    if (*):
        x := x + i
    else
        y := y + i
    i := i + 1
```

$$0 \leq \mathtt{i} \wedge \mathtt{i} < 100$$
$$\wedge \mathtt{x}' = \mathtt{x} + \mathtt{i}$$
$$\wedge \mathtt{y}' = \mathtt{y} + \mathtt{i}$$
$$\wedge \mathtt{i}' = \mathtt{i} + 1$$

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

$$\psi \triangleq \varphi_{\text{body}} \wedge \delta_{\mathsf{x}} = \mathsf{x}' - \mathsf{x} \wedge \delta_{\mathsf{y}} = \mathsf{y}' - \mathsf{y}$$

Linear recurrences (in)equations

```
while(0 <= i < 100):
    if (*):
        x := x + i
    else
        y := y + i
    i := i + 1
```

$$0 \leq \mathtt{i} \wedge \mathtt{i} < 100$$
$$\wedge \mathtt{x}' = \mathtt{x} + \mathtt{i}$$
$$\wedge \mathtt{y}' = \mathtt{y} + \mathtt{i}$$
$$\wedge \mathtt{i}' = \mathtt{i} + 1$$

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

$$\psi \triangleq \varphi_{\text{body}} \wedge \delta_{\mathsf{x}} = \mathsf{x}' - \mathsf{x} \wedge \delta_{\mathsf{y}} = \mathsf{y}' - \mathsf{y}$$

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

Linear recurrences (in)equations

```
while(0 <= i < 100):
    if (*):
        x := x + i
    else
        y := y + i
    i := i + 1
```

$$0 \leq i \land i < 100$$
$$\land x' = x + i$$
$$\land y' = y + i$$
$$\land i' = i + 1$$

$\delta_x + \delta_y = i$

$0 \leq \delta_x \leq i$

$0 \leq \delta_y \leq i$

...

Linear equations over δ's and induction variables

Linear recurrences (in)equations

```
while(0 <= i < 100):
    if (*):
        x := x + i
    else
        y := y + i
    i := i + 1
```

$$0 \leq i \wedge i < 100$$
$$\wedge x' = x + i$$
$$\wedge y' = y + i$$
$$\wedge i' = i + 1$$

$\delta_x + \delta_y = i$

$0 \leq \delta_x \leq i$

$0 \leq \delta_y \leq i$

...

→

$(x' - x) + (y' - y) = i$

$0 \leq (x' - x) \leq i$

$0 \leq (y' - y) \leq i$

Linear recurrences (in)equations

```
while(0 <= i < 100):
    if (*):
        x := x + i
    else
        y := y + i
    i := i + 1
```

$0 \leq \mathrm{i} \wedge \mathrm{i} < 100$
$\quad \wedge \mathrm{x}' = \mathrm{x} + \mathrm{i}$
$\quad \wedge \mathrm{y}' = \mathrm{y} + \mathrm{i}$
$\wedge \mathrm{i}' = \mathrm{i} + 1$

$\delta_x + \delta_y = \mathrm{i}$
$0 \leq \delta_x \leq \mathrm{i}$
$0 \leq \delta_y \leq \mathrm{i}$
...

→

$(\mathrm{x}' - \mathrm{x}) + (\mathrm{y}' - \mathrm{y}) = i$
$0 \leq (\mathrm{x}' - \mathrm{x}) \leq \mathrm{i}$
$0 \leq (\mathrm{y}' - \mathrm{y}) \leq \mathrm{i}$

→

$\mathrm{x}' + \mathrm{y}' = \mathrm{x} + \mathrm{y} + \mathrm{i}$
$\mathrm{x} \leq \mathrm{x}' \leq \mathrm{x} + \mathrm{i}$
$\mathrm{x} \leq \mathrm{y}' \leq \mathrm{y} + \mathrm{i}$

Linear recurrences (in)equations

```
while(0 <= i < 100):
    if (*):
        x := x + i
    else
        y := y + i
    i := i + 1
```

$$0 \le \mathtt{i} \land \mathtt{i} < 100$$
$$\land \mathtt{x}' = \mathtt{x} + \mathtt{i}$$
$$\land \mathtt{y}' = \mathtt{y} + \mathtt{i}$$
$$\land \mathtt{i}' = \mathtt{i} + 1$$

$$\delta_x + \delta_y = i$$
$$0 \le \delta_x \le i$$
$$0 \le \delta_y \le i$$
...

→

$$(x' - x) + (y' - y) = i$$
$$0 \le (x' - x) \le i$$
$$0 \le (y' - y) \le i$$

→

$$x' + y' = x + y + i$$
$$x \le x' \le x + i$$
$$x \le y' \le y + i$$

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

$$x^{(0)} \le x^{(k)} \le x^{(0)} + k i^{(0)} + k(k+1)/2$$

$$x^{(0)} \le y^{(k)} \le y^{(0)} + k i^{(0)} + k(k+1)/2$$

Putting it all together

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

Extracted recurrences

Putting it all together

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

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

Closed form

Putting it all together

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

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

$$\begin{aligned} \varphi^{\circledast}_{\mathsf{body}} \triangleq & \bigwedge_i \mathsf{x}'_i = \mathsf{x}_i \\ & \vee \Big(\exists k . k \geq 1 \wedge (\exists \vec{x}' . \varphi_{\mathsf{body}}) \wedge (\exists \vec{x} . \varphi_{\mathsf{body}}) \\ & \quad \wedge \bigwedge_r \sum_i a_{ri} \mathsf{x}'_{ri} \leq \sum_i a_{ri} \mathsf{x}_{ri} + \sum_j p_{rj}(k) \mathsf{y}_{rj} + k c_r \Big) \end{aligned}$$

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*