Compositional Recurrence Analysis Revisited

Zachary Kincaid[1] Jason Breck[2] Ashkan Forouhi Boroujeni[2] Thomas Reps[2,3]

[1]Princeton University [2]University of Wisconsin-Madison [3]GrammaTech, Inc.

June 19, 2017

How can we apply loop analyses to recursive procedures?

Over-approximating the behavior of loops

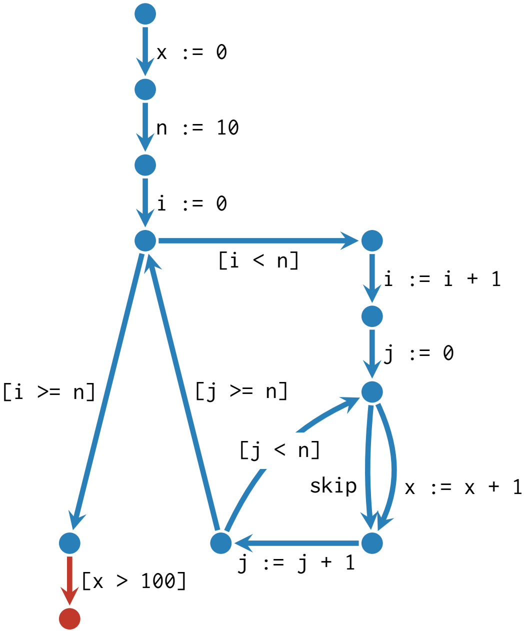
- Iterative program analysis [Cousot & Cousot POPL 1977]
    - Repeatedly evaluate the program under an abstract semantics until convergence upon a property that over-approximates all reachable states.

Over-approximating the behavior of loops

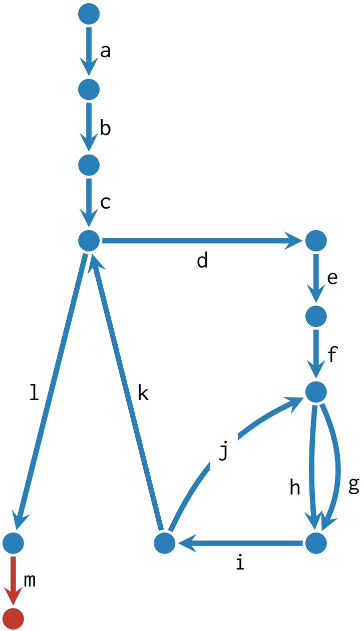
- Iterative program analysis [Cousot & Cousot POPL 1977]
  - Repeatedly evaluate the program under an abstract semantics until convergence upon a property that over-approximates all reachable states.
- Algebraic program analysis [Tarjan JACM 1981]
  1. Compute a *path expression* to a point of interest (e.g., an assertion)
  2. Evaluate the path expression in the *semantic algebra* defining the analysis to yield a property that over-approximates all paths.

```
        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)
```

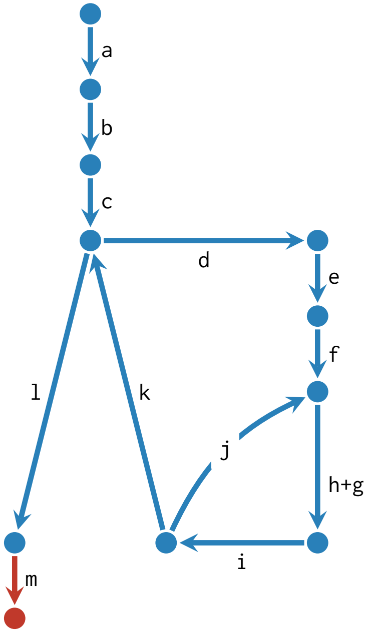
```
        x := 0
        n := 10
        i := 0
outer:  if(i >= n):
          goto end
        i := i + 1
inner:  j := 0
        if(*):
          x := x + 1
        j := j + 1
        if(j < n):
          goto inner
        goto outer
end:    assert(x <= 100)
```



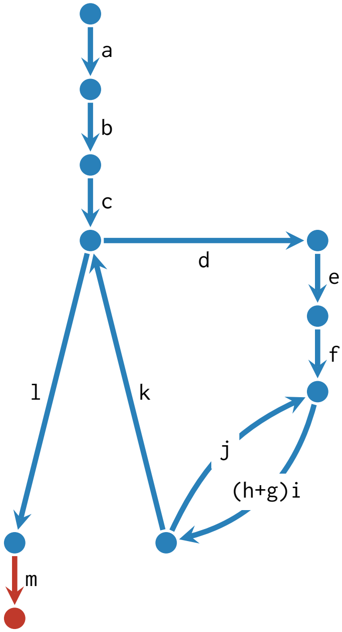
```
        x := 0
        n := 10
        i := 0
outer:  if(i >= n):
           goto end
        i := i + 1
inner:  j := 0
        if(*):
           x := x + 1
        j := j + 1
        if(j < n):
           goto inner
        goto outer
end:    assert(x <= 100)
```



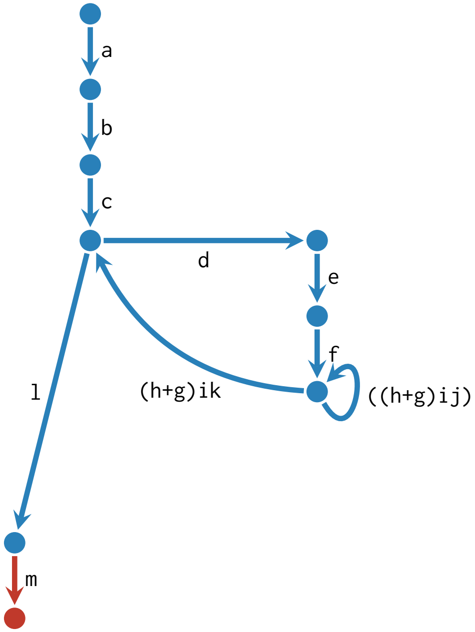
```
         x := 0
         n := 10
         i := 0
outer:   if(i >= n):
            goto end
         i := i + 1
inner:   j := 0
         if(*):
            x := x + 1
         j := j + 1
         if(j < n):
            goto inner
         goto outer
end:     assert(x <= 100)
```



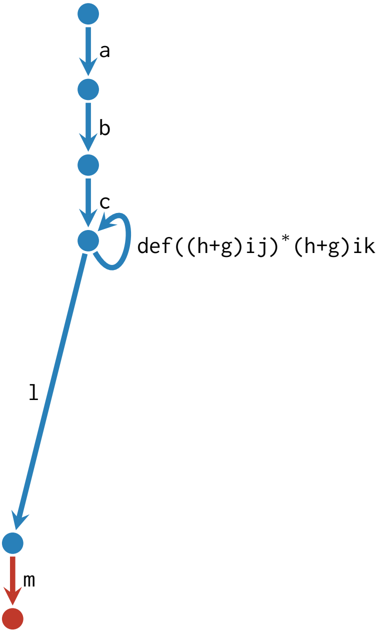
```
        x := 0
        n := 10
        i := 0
outer:  if(i >= n):
          goto end
        i := i + 1
inner:  j := 0
        if(*):
          x := x + 1
        j := j + 1
        if(j < n):
          goto inner
        goto outer
end:    assert(x <= 100)
```



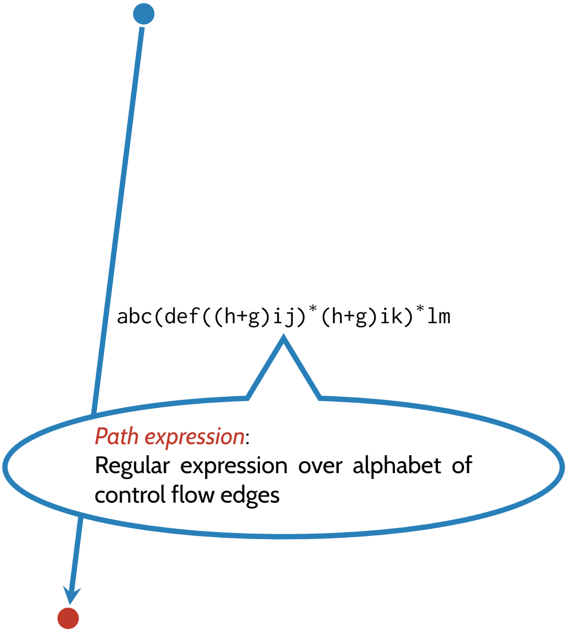
```
        x := 0
        n := 10
        i := 0
outer:  if(i >= n):
           goto end
        i := i + 1
inner:  j := 0
        if(*):
           x := x + 1
        j := j + 1
        if(j < n):
           goto inner
        goto outer
end:    assert(x <= 100)
```



```
        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)
```



Evaluation of a path expression:

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

Evaluation of a path expression:

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

Space of program properties

Composition operators

Evaluation of a path expression:

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

Space of program properties

Evaluation of a path expression:

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

Evaluation of a path expression:

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

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

Compositional recurrence analysis [Farzan & Kincaid FMCAD 2015]

- D is the set of *transition formulas* in non-linear integer arithmetic

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

Compositional recurrence analysis [Farzan & Kincaid FMCAD 2015]

- D is the set of *transition formulas* in non-linear integer arithmetic

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

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

Compositional recurrence analysis [Farzan & Kincaid FMCAD 2015]

- D is the set of *transition formulas* in non-linear integer arithmetic

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

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

Compositional recurrence analysis [Farzan & Kincaid FMCAD 2015]

- D is the set of *transition formulas* in non-linear integer arithmetic

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

- $\varphi \otimes \psi \triangleq \exists \mathbf{x}''. \varphi[\mathbf{x}' \mapsto \mathbf{x}''] \land \psi[\mathbf{x} \mapsto \mathbf{x}'']$
- $\varphi \oplus \psi \triangleq \varphi \lor \psi$
- $\varphi^* \triangleq \ldots$

CRA's iteration operator

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

loop body

$$\mathtt{i} < \mathtt{n} \land \begin{pmatrix} & (\mathtt{x}' = \mathtt{x} + \mathtt{i} \land \mathtt{y}' = \mathtt{y}) \\ \lor & (\mathtt{y}' = \mathtt{y} + \mathtt{i} \land \mathtt{x}' = \mathtt{x}) \end{pmatrix} \land \mathtt{i}' = \mathtt{i} + 1 \land \mathtt{n}' = \mathtt{n}$$

loop abstraction

$$\exists k. k \geq 0 \land \mathtt{i}' = \mathtt{i} + k \land \mathtt{x}' + \mathtt{y}' = \mathtt{x} + \mathtt{y} + k(k+1)/2 + k\mathtt{i}_0 \land \mathtt{x}' \geq \mathtt{x} \land \mathtt{y}' \geq \mathtt{y}$$

CRA's iteration operator

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

loop body

$$\mathtt{i} < \mathtt{n} \land \left(\begin{array}{ll} & (\mathtt{x}' = \mathtt{x} + \mathtt{i} \land \mathtt{y}' = \mathtt{y}) \\ \lor & (\mathtt{y}' = \mathtt{y} + \mathtt{i} \land \mathtt{x}' = \mathtt{x}) \end{array} \right) \land \mathtt{i}' = \mathtt{i} + 1 \land \mathtt{n}' = \mathtt{n}$$

recurrences

$$\mathtt{i}^{(k)} = \mathtt{i}^{(k-1)} + 1$$

$$\mathtt{x}^{(k)} + \mathtt{y}^{(k)} = \mathtt{x}^{(k-1)} + \mathtt{y}^{(k-1)} + \mathtt{i}$$

$$\mathtt{x}^{(k)} \geq \mathtt{x}^{(k-1)}$$

$$\mathtt{y}^{(k)} \geq \mathtt{y}^{(k-1)}$$

loop abstraction

$$\exists k. k \geq 0 \land \mathtt{i}' = \mathtt{i} + k \land \mathtt{x}' + \mathtt{y}' = \mathtt{x} + \mathtt{y} + k(k+1)/2 + k\mathtt{i}_0 \land \mathtt{x}' \geq \mathtt{x} \land \mathtt{y}' \geq \mathtt{y}$$

CRA's iteration operator

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

loop body

$$\mathtt{i} < \mathtt{n} \land \begin{pmatrix} & (\mathtt{x}' = \mathtt{x} + \mathtt{i} \land \mathtt{y}' = \mathtt{y}) \\ \lor & (\mathtt{y}' = \mathtt{y} + \mathtt{i} \land \mathtt{x}' = \mathtt{x}) \end{pmatrix} \land \mathtt{i}' = \mathtt{i} + 1 \land \mathtt{n}' = \mathtt{n}$$

recurrences

$$\mathtt{i}^{(k)} = \mathtt{i}^{(k-1)} + 1$$
$$\mathtt{x}^{(k)} + \mathtt{y}^{(k)} = \mathtt{x}^{(k-1)} + \mathtt{y}^{(k-1)} + \mathtt{i}$$
$$\mathtt{x}^{(k)} \geq \mathtt{x}^{(k-1)}$$
$$\mathtt{y}^{(k)} \geq \mathtt{y}^{(k-1)}$$

closed forms

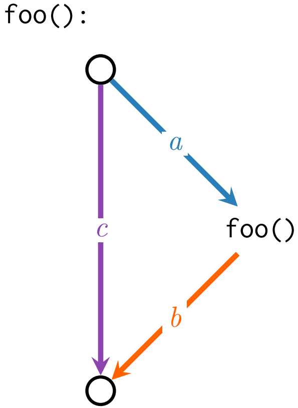
$$\mathtt{i}^{(k)} = \mathtt{i}^{(0)} + k$$
$$\mathtt{x}^{(k)} + \mathtt{y}^{(k)} = \mathtt{x}^{(0)} + \mathtt{y}^{(0)} + \frac{k(k+1)}{2} + k\mathtt{i}_0$$
$$\mathtt{x}^{(k)} \geq \mathtt{x}^{(0)}$$
$$\mathtt{y}^{(k)} \geq \mathtt{y}^{(0)}$$

loop abstraction

$$\exists k. k \geq 0 \land \mathtt{i}' = \mathtt{i} + k \land \mathtt{x}' + \mathtt{y}' = \mathtt{x} + \mathtt{y} + k(k+1)/2 + k\mathtt{i}_0 \land \mathtt{x}' \geq \mathtt{x} \land \mathtt{y}' \geq \mathtt{y}$$

How can we apply CRA to recursive procedures?

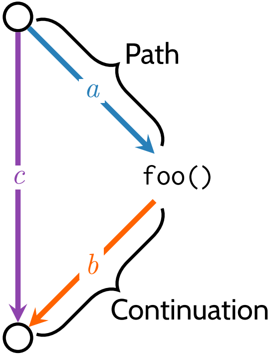
Recursive procedures have non-regular path languages



paths(foo) $= \{a^i c b^i : i \geq 0\}$ ***is not regular!***

Tensor domains [Reps, Turetsky, Prabhu POPL 2016]

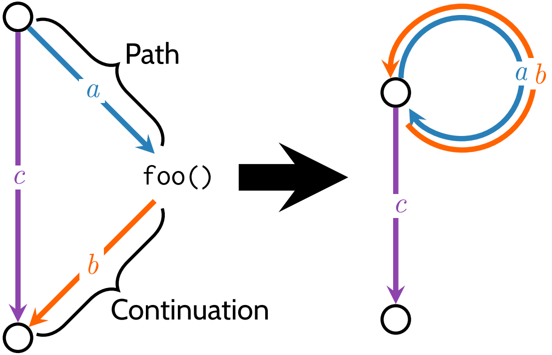
`foo():`



paths(`foo`) $= \{a^i c b^i : i \geq 0\}$ *is not regular!*

Tensor domains [Reps, Turetsky, Prabhu POPL 2016]

`foo():`



paths$(\texttt{foo}) = \{a^i c b^i : i \geq 0\}$ ***is not regular!***

Tensored paths

A tensored path (p, k) is a pair consisting of a path p and a continuation k.

Tensored paths

A tensored path (p, k) is a pair consisting of a path p and a continuation k.

- $T_1 \otimes T_2 \triangleq \{(p_2 p_1, k_1 k_2) : (p_1, k_1) \in T_1, (p_2, k_2) \in T_2\}$
  - $(cd, dc) \otimes (ab, ba) = (abcd, dcba)$

Tensored paths

A tensored path (p, k) is a pair consisting of a path p and a continuation k.

- $T_1 \otimes T_2 \triangleq \{(p_2 p_1, k_1 k_2) : (p_1, k_1) \in T_1, (p_2, k_2) \in T_2\}$
  - $(cd, dc) \otimes (ab, ba) = (abcd, dcba)$
- $T_1 \oplus T_2 \triangleq T_1 \cup T_2$

Tensored paths

A tensored path (p, k) is a pair consisting of a path p and a continuation k.

- $T_1 \otimes T_2 \triangleq \{(p_2 p_1, k_1 k_2) : (p_1, k_1) \in T_1, (p_2, k_2) \in T_2\}$
  - $(cd, dc) \otimes (ab, ba) = (abcd, dcba)$
- $T_1 \oplus T_2 \triangleq T_1 \cup T_2$
- $T^* \triangleq \bigcup_n T^n$
  - $(a, b)^* = \{(a^i, b^i) : i \geq 0\}$

Tensored paths

A tensored path (p, k) is a pair consisting of a path p and a continuation k.

- $T_1 \otimes T_2 \triangleq \{(p_2 p_1, k_1 k_2) : (p_1, k_1) \in T_1, (p_2, k_2) \in T_2\}$
  - $(cd, dc) \otimes (ab, ba) = (abcd, dcba)$
- $T_1 \oplus T_2 \triangleq T_1 \cup T_2$
- $T^* \triangleq \bigcup_n T^n$
  - $(a, b)^* = \{(a^i, b^i) : i \geq 0\}$

Tensor product pairs a paths and continuations

$$P \odot K \triangleq \{(p, k) : p \in p, k \in K\}$$

Detensor product places a path between a path & continuation

$$Q \ltimes T \triangleq \{pqk : q \in Q, (p, k) \in T$$

For example, $c \ltimes (a \odot b)^* = \{a^i c b^i : i \geq 0\}$.

Tensor domain of CRA

Tensored transition formula $\sim$ formula over *four* copies of the program variables

$$\underline{\mathbf{x}}, \underline{\mathbf{x}}', \overline{\mathbf{x}}, \overline{\mathbf{x}}'$$

Tensor domain of CRA

Tensored transition formula $\sim$ formula over *four* copies of the program variables

$$\underline{\mathbf{x}}, \underline{\mathbf{x}}', \overline{\mathbf{x}}, \overline{\mathbf{x}}'$$

Beginning of path

End of path

Tensor domain of CRA

Tensored tra… copies of the program variables

Beginning of continuation

End of continuation

$$\underline{\mathbf{x}}\,,\,\underline{\mathbf{x}}'\,,\,\overline{\mathbf{x}}\,,\,\overline{\mathbf{x}}'$$

Beginning of path

End of path

Tensor domain of CRA

Tensored transition formula $\sim$ formula over *four* copies of the program variables

$$\underline{\mathbf{x}}, \underline{\mathbf{x}}', \overline{\mathbf{x}}, \overline{\mathbf{x}}'$$

- $\Phi \otimes \Psi \triangleq \exists \underline{\mathbf{x}}'', \overline{\mathbf{x}}''. \Phi[\underline{\mathbf{x}} \mapsto \underline{\mathbf{x}}'', \overline{\mathbf{x}}' \mapsto \overline{\mathbf{x}}''] \land \Psi[\underline{\mathbf{x}}' \mapsto \underline{\mathbf{x}}'', \overline{\mathbf{x}} \mapsto \overline{\mathbf{x}}'']$

Tensor domain of CRA

Tensored transition formula $\sim$ formula over *four* copies of the program variables

$$\underline{\mathbf{x}}, \underline{\mathbf{x}}', \overline{\mathbf{x}}, \overline{\mathbf{x}}'$$

- $\Phi \otimes \Psi \triangleq \exists \underline{\mathbf{x}}'', \overline{\mathbf{x}}''. \Phi[\underline{\mathbf{x}} \mapsto \underline{\mathbf{x}}'', \overline{\mathbf{x}}' \mapsto \overline{\mathbf{x}}''] \wedge \Psi[\underline{\mathbf{x}}' \mapsto \underline{\mathbf{x}}'', \overline{\mathbf{x}} \mapsto \overline{\mathbf{x}}'']$
- $\Phi \oplus \Psi$, Φ^* as for the untensored case

Tensor domain of CRA

Tensored transition formula $\sim$ formula over *four* copies of the program variables

$$\underline{\mathbf{x}}\,,\,\underline{\mathbf{x}}'\,,\,\overline{\mathbf{x}}\,,\,\overline{\mathbf{x}}'$$

- $\Phi \otimes \Psi \triangleq \exists \underline{\mathbf{x}}'', \overline{\mathbf{x}}''. \Phi[\underline{\mathbf{x}} \mapsto \underline{\mathbf{x}}'', \overline{\mathbf{x}}' \mapsto \overline{\mathbf{x}}''] \land \Psi[\underline{\mathbf{x}}' \mapsto \underline{\mathbf{x}}'', \overline{\mathbf{x}} \mapsto \overline{\mathbf{x}}'']$
- $\Phi \oplus \Psi$, Φ^* as for the untensored case
- $\varphi \odot \psi \triangleq \varphi[\mathbf{x} \mapsto \underline{\mathbf{x}}, \mathbf{x}' \mapsto \underline{\mathbf{x}}'] \land \psi[\mathbf{x} \mapsto \overline{\mathbf{x}}, \mathbf{x}' \mapsto \overline{\mathbf{x}}']$
  - E.g., $(x' = x + 1) \odot (y' = y + 2) = (\underline{x}' = \underline{x} + 1 \land \overline{y}' = \overline{y} + 2)$
- $\varphi \ltimes \Psi \triangleq \exists \underline{\mathbf{x}}, \underline{\mathbf{x}}', \overline{\mathbf{x}}, \overline{\mathbf{x}}'. \varphi[\mathbf{x} \mapsto \underline{\mathbf{x}}', \mathbf{x}' \mapsto \overline{\mathbf{x}}] \land \Phi[\underline{\mathbf{x}} \mapsto \mathbf{x}, \overline{\mathbf{x}}' \mapsto \mathbf{x}']$
  - E.g., $(y' = x) \ltimes (\underline{x}' = \underline{x} + 1 \land \overline{y}' = \overline{y} + 2) = (y' = x + 3)$

Solving non-linear recursive systems

```
fib(n):
   if(i > 1):
      f1 := fib(n-1)
      f2 := fib(n-2)
      return f1 + f2
   else
      return 1
```

Solving non-linear recursive systems

$paths(\texttt{fib}) \sim$ least fixed point of $X = aXbXc + d$

Solving non-linear recursive systems

$paths(\texttt{fib}) \sim$ least fixed point of $X = aXbXc + d$

[Reps, Turetsky, Prabhu POPL 2016] solves this via *Newton iteration*:

Solve a sequence of *linearized* systems until convergence on a property that over-approximates all paths.

Newton iteration

$\nu_0 = 0$

$\nu_1 = d \ltimes \big((a \odot b\nu_0 c) \oplus (a\nu_0 b \odot c)\big)^*$

$\nu_2 = d \ltimes \big((a \odot b\nu_1 c) \oplus (a\nu_1 b \odot c)\big)^*$

$\vdots$

(repeat until $\nu_{n+1} = \nu_n$)

The problem with Newton iteration

1. Transition formulas have infinite ascending chains (convergence is not guaranteed)
2. Transition formula equivalence is undecidable (convergence can't be detected)

CRA's iteration operator

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

loop body

$$i < n \wedge \left(\begin{array}{ll} & (x' = x + i \wedge y' = y) \\ \vee & (y' = y + i \wedge x' = x) \end{array} \right) \wedge i' = i + 1 \wedge n' = n$$

α

recurrences

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

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

$$x^{(k)} \geq x^{(k-1)}$$

$$y^{(k)} \geq y^{(k-1)}$$

cl

loop abstraction

$$\exists k. k \geq 0 \wedge i' = i + k \wedge x' + y' = x + y + k(k+1)/2 + k i_0 \wedge x' \geq x \wedge y' \geq y$$

CRA's iteration operator

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

loop body

$$\mathtt{i} < \mathtt{n} \wedge \left(\begin{array}{ll} & (\mathtt{x}' = \mathtt{x} + \mathtt{i} \wedge \mathtt{y}' = \mathtt{y}) \\ \vee & (\mathtt{y}' = \mathtt{y} + \mathtt{i} \wedge \mathtt{x}' = \mathtt{x}) \end{array} \right) \wedge \mathtt{i}' = \mathtt{i} + 1 \wedge \mathtt{n}' = \mathtt{n}$$

α

recurrences

Polyhedron

$$\begin{aligned} \mathtt{i}^{(k)} &= \mathtt{i}^{(k-1)} + 1 \\ \mathtt{x}^{(k)} + \mathtt{y}^{(k)} &= \mathtt{x}^{(k-1)} + \mathtt{y}^{(k-1)} + \mathtt{i} \\ \mathtt{x}^{(k)} &\geq \mathtt{x}^{(k-1)} \\ \mathtt{y}^{(k)} &\geq \mathtt{y}^{(k-1)} \end{aligned}$$

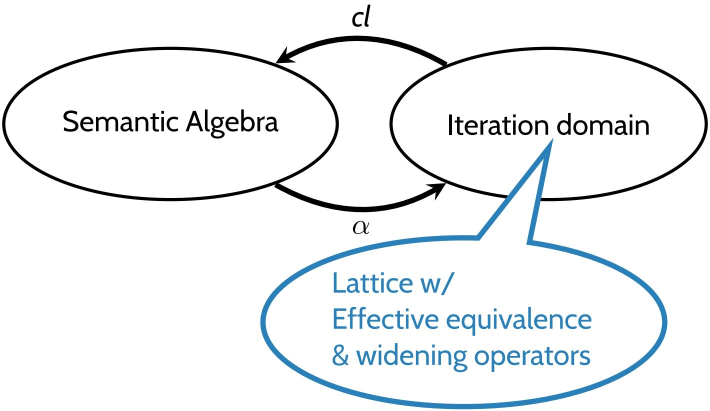
cl

loop abstraction

$$\exists k. k \geq 0 \wedge \mathtt{i}' = \mathtt{i} + k \wedge \mathtt{x}' + \mathtt{y}' = \mathtt{x} + \mathtt{y} + k(k+1)/2 + k\mathtt{i}_0 \wedge \mathtt{x}' \geq \mathtt{x} \wedge \mathtt{y}' \geq \mathtt{y}$$

Iteration domains

$$\varphi^* = \mathbf{cl}(\alpha(\varphi))$$

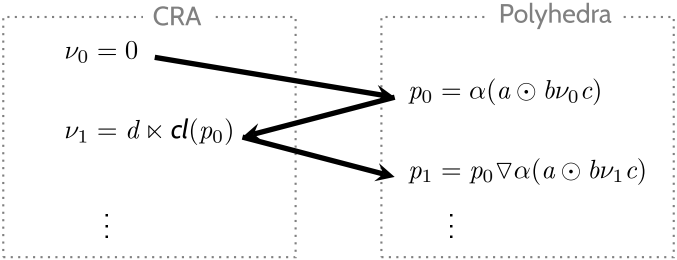


Key idea: we have an opportunity to detect / enforce convergence at every place we apply the $*$ *operator.*

$$X = aXbXc + d \rightsquigarrow X = d \ltimes (a \odot bXc)^*$$

All variables appear below $*$

$$X = aXbXc + d \rightsquigarrow X = d \ltimes (a \odot bXc)^*$$



(repeat until $p_{n+1} = p_n$)

Over-approximating recursive procedures

Given a system of recursive equations describing a set of paths,

1. Using the tensor domain, rewrite the system so that every variable appears below a star (similar to Gauss-Jordan elimination)
2. Compute solution to resulting system iteratively, using iteration domains to detect and enforce convergence.

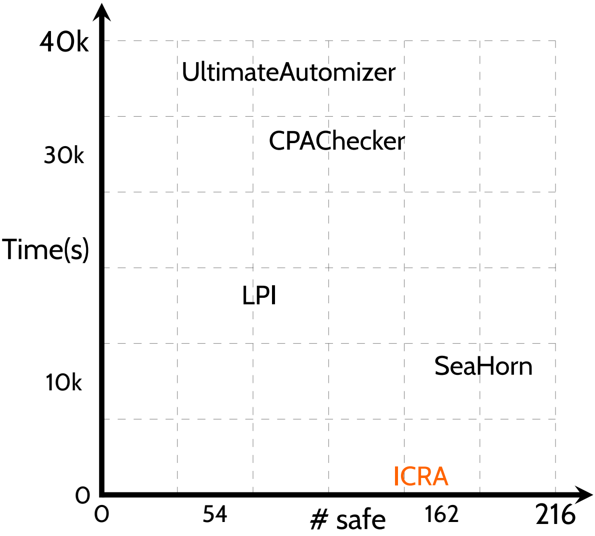
Implementation & Evaluation

ICRA was implemented on top of CRA and WALi

- (uses Cil C frontend, Z3 SMT solver, Apron abstract domain library)

Experimental set-up

- Ran on 216 *safe* benchmarks collected from SV-Comp, C4B (resource bound verification problems), and misc examples
- Compare with SeaHorn, CPAChecker, LPI, Ultimate Automizer


40k
30k
Time(s)
10k
0
UltimateAutomizer
CPAChecker
LPI
SeaHorn
ICRA
0
54
safe
162
216

Summary

Algebraic analyses can be extended to recursive procedures using

1. *Tensor domains*, to re-arrange recursion into loops
2. *Iteration domains*, to detect and enforce convergence

Experimental results

Benchmark Suite	Total #A	ICRA Time	ICRA #A	UAut. Time	UAut. #A	CPA Time	CPA #A	LPI Time	LPI #A	SEA Time	SEA #A
recursive	18/7	**40.7**	7	1952.1	8	1817.8	10	62.0	0	1334.0	**14**
rec.-simple	36/38	**168.7**	21	6979.3	28	2760.4	32	179.5	3	743.8	**36**
Rec. (tot.)	54/45	**209.4**	28	8931.4	36	4578.1	42	241.5	3	2077.8	**50**
loop-accel.	19/16	**20.8**	13	6696.5	7	4565.7	13	4227.7	13	2713.1	**15**
loop-invgen	18/1	**53.1**	**16**	1876.2	7	4909.6	2	1282.3	15	506.0	**16**
loop-lit	15/1	316.5	12	2722.9	5	2720.6	7	444.9	**13**	**305.2**	**13**
loops	34/32	**209.7**	22	3984.1	19	4380.1	**28**	3356.8	26	1821.5	27
loop-new	8/0	304.8	**7**	2147.9	1	1866.1	3	929.6	4	**302.8**	6
Loops (tot.)	94/50	**904.8**	70	17427.6	39	18442.2	53	10241.3	71	5648.6	**77**
C4B	35/0	**30.3**	**30**	6156.6	1	7817.8	2	6726.7	0	1867.6	29
misc	10/4	76.7	**10**	492.2	8	334.4	7	332.2	1	**5.3**	**10**
rec-loop-lit	15/1	312.7	9	2755.5	3	51.0	6	**40.4**	0	922.6	**12**
rec-loop-new	8/0	**6.2**	**5**	1546.9	2	25.6	2	19.6	0	905.7	4
Misc.-Rec.	33/5	395.6	24	4794.6	13	410.9	15	**392.2**	1	1833.7	**26**