

Non-Linear Reasoning for Invariant Synthesis

Zachary Kincaid¹ John Cyphert² Jason Breck² Thomas Reps^{2,3}

¹Princeton University

²University of Wisconsin-Madison

³GrammarTech, Inc

January 12, 2018 @ 15:50

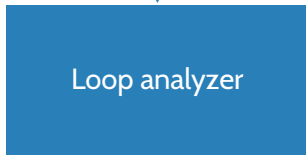
The problem: generating non-linear numerical loop invariants

- Resource-bound analysis
- Side channel analysis
- Secure information flow
- ...

```
while(i < n):
```

```
  x = x + i
```

```
  i = i + 1
```

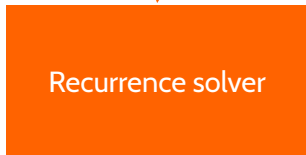


Loop analyzer



$$\exists k. k \geq 0 \wedge \left(\begin{array}{l} i' = i + k \\ x' = x + \frac{k(k-1)}{2} + ki \end{array} \right)$$

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



Recurrence solver



$$\begin{aligned} i^{(k)} &= i^{(0)} + k \\ x^{(k)} &= x^{(0)} + \frac{k(k-1)}{2} + ki^{(0)} \end{aligned}$$

```
while(i < n):
```

```
  x = x + i
```

```
  i = i + 1
```

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

- branching
- nested loops
- non-determinism

Loop analysis

Recurrence solver

$$\exists k. k \geq 0 \wedge \begin{pmatrix} i' = i + k \\ x' = x + \frac{k(k-1)}{2} + ki \end{pmatrix}$$

$$\begin{aligned} i^{(k)} &= i^{(0)} + k \\ x^{(k)} &= x^{(0)} + \frac{k(k-1)}{2} + ki^{(0)} \end{aligned}$$

```
while(i < n):
```

```
  x = x + i
```

```
  i = i + 1
```

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

- branching
- nested loops
- non-determinism

Loop analysis

Recurrence solver

$$\exists k. k \geq 0 \wedge \begin{pmatrix} i' = i \\ x' = x + \frac{k(k-1)}{2} + ki^{(0)} \end{pmatrix}$$

algebraic numbers

```
binary-search(A,target):
```

```
lo = 1, hi = size(A), ticks = 0
```

```
while (lo <= hi):
```

```
    ticks++;
```

```
    mid = lo + (hi-lo)/2
```

```
    if A[mid] == target:
```

```
        return mid
```

```
    else if A[mid] < target:
```

```
        lo = mid+1
```

```
    else :
```

```
        hi = mid-1
```

} log(A) times

binary-search(A, target):

lo = 1, hi = size(A), ticks = 0

while (lo <= hi):

 ticks++;

 mid = lo + (hi-lo)/2

if A[mid] == target:

return mid

else if A[mid] < target:

 lo = mid+1

else :

 hi = mid-1

$$\left. \begin{array}{l} ticks' = ticks + 1 \\ \wedge mid' = lo + (hi - lo)/2 \\ \wedge ((A[mid] < target \\ \wedge lo' = mid + 1 \\ \wedge hi' = hi) \\ \vee (A[mid] > target \\ \wedge lo' = lo \\ \wedge hi' = mid - 1)) \end{array} \right\}$$

binary

lo =

while (lo

ticks++;

mid = lo + (hi-lo)/2

if A[mid] == target:

return mid

else if A[mid] < target:

lo = mid+1

else :

hi = mid-1

$$ticks^{(k+1)} = ticks^{(k)} + 1$$

$$(hi' - lo')^{(k+1)} \leq (hi - lo)^{(k)} / 2 - 1$$

$$ticks = ticks + 1$$

$$mid = lo + (hi - lo) / 2$$

$$\wedge ((A[mid] < target$$

$$\wedge lo' = mid + 1$$

$$\wedge hi' = hi)$$

$$\vee (A[mid] > target$$

$$\wedge lo' = lo$$

$$\wedge hi' = mid - 1))$$

binary

lo =

while (lo

ticks++;

mid = lo + (hi-lo)/2

if A[mid] == target:

return mid

else if A[mid] < target:

lo = mid+1

else :

hi = mid-1

$$ticks^{(k)} = ticks^{(0)} + k$$

$$(hi' - lo')^{(k)} \leq \left(\frac{1}{2}\right)^k (hi - lo + 2)^{(0)} - 2$$

$$ticks = ticks + 1$$

$$\wedge mid = lo + (hi - lo)/2$$

$$\wedge ((A[mid] < target$$

$$\wedge lo' = mid + 1$$

$$\wedge hi' = hi)$$

$$\vee (A[mid] > target$$

$$\wedge lo' = lo$$

$$\wedge hi' = mid - 1))$$

binary-search(A, target):

lo = 1, hi = size(A), ticks = 0

while (lo <= hi):

 ticks++;

 mid = lo + (hi-lo)/2

if A[mid] == target:

return mid

else if A[mid] < target:

 lo = mid+1

else :

 hi = mid-1

$\exists k. k \geq 0$

$ticks' = ticks + k$

$(hi' - lo') \leq \left(\frac{1}{2}\right)^k (hi - lo + 2) - 2$

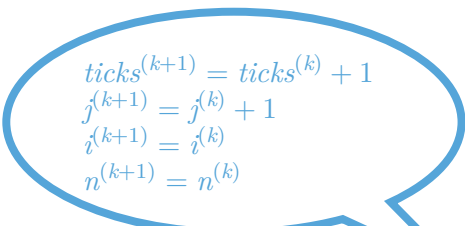
```
for (i = 0; i < n; i++):  
    for (j = 0; j < i; j++):  
        ticks++
```

```

for (i = 0; i < n; i++):
    for (j = 0; j < i; j++):
        ticks++

```

$$\left. \begin{array}{l}
 j < i \\
 \wedge j' = j + 1 \\
 \wedge ticks' = ticks + 1 \\
 \wedge i' = i \\
 \wedge n' = n
 \end{array} \right\}$$



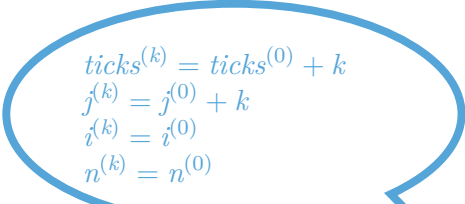
$$\begin{aligned}
 ticks^{(k+1)} &= ticks^{(k)} + 1 \\
 j^{(k+1)} &= j^{(k)} + 1 \\
 i^{(k+1)} &= i^{(k)} \\
 n^{(k+1)} &= n^{(k)}
 \end{aligned}$$

```

for (i = 0; i < n; i++):
    for (j = 0; j < i; j++):
        ticks++

```

$j < i$
 $\wedge j' = j + 1$
 $\wedge ticks' = ticks + 1$
 $\wedge i' = i$
 $\wedge n' = n$



$$\begin{aligned}
 ticks^{(k)} &= ticks^{(0)} + k \\
 j^{(k)} &= j^{(0)} + k \\
 i^{(k)} &= i^{(0)} \\
 n^{(k)} &= n^{(0)}
 \end{aligned}$$

```

for (i = 0; i < n; i++):
    for (j = 0; j < i; j++):
        ticks++

```

$j < i$
 $\wedge j' = j + 1$
 $\wedge ticks' = ticks + 1$
 $\wedge i' = i$
 $\wedge n' = n$

$$\begin{array}{l}
 \text{for } (i = 0; i < n; i++): \\
 \quad \text{for } (j = 0; j < i; j++): \\
 \quad \quad \text{ticks}++
 \end{array}
 \left. \vphantom{\begin{array}{l} \text{for } (i = 0; i < n; i++): \\ \text{for } (j = 0; j < i; j++): \\ \text{ticks}++ \end{array}} \right\}
 \begin{array}{l}
 \wedge \quad i' = i \\
 \wedge \quad n' = n \\
 \wedge \quad j' \leq i \\
 \wedge \quad \left(\begin{array}{l} \exists k. \quad k \geq 0 \\ \wedge \quad \text{ticks}' = \text{ticks} + k \\ \wedge \quad j' = j + k \end{array} \right)
 \end{array}$$

$$\begin{array}{l}
\text{for } (i = 0; i < n; i++): \\
\quad \text{for } (j = 0; j < i; j++): \\
\quad \quad \text{ticks}++
\end{array}
\left\{ \begin{array}{l}
\wedge \quad i < n \\
\wedge \quad i' = i + 1 \\
\wedge \quad n' = n \\
\wedge \quad j' = i \\
\wedge \quad \left(\begin{array}{l}
\exists k. \quad k \geq 0 \\
\wedge \text{ticks}' = \text{ticks} + k \\
\wedge j' = k
\end{array} \right)
\end{array} \right.$$

$$\begin{aligned} ticks^{(k+1)} &= ticks^{(k)} + i^{(k)} \\ i^{(k+1)} &= i^{(k)} + 1 \\ n^{(k+1)} &= n^{(k)} \end{aligned}$$

```
for (i = 0; i < n; i++):
  for (j = 0; j < i; j++):
    ticks++
```

$$\left. \begin{array}{l} i < n \\ i' = i + 1 \\ i < n \\ j' = i \end{array} \right\} \wedge \left(\begin{array}{l} \exists k. \quad k \geq 0 \\ \wedge ticks' = ticks + k \\ \wedge j' = k \end{array} \right)$$

$$\begin{aligned}
 ticks^{(k)} &= ticks^{(0)} + k(k+1)/2 + ki^{(0)} \\
 i^{(k)} &= i^{(0)} + k \\
 n^{(k)} &= n^{(0)}
 \end{aligned}$$

```

for (i = 0; i < n; i++):
  for (j = 0; j < i; j++):
    ticks++
  
```

}

$$\begin{aligned}
 & \wedge i' = i + 1 \\
 & \wedge n' = n \\
 & \wedge j' = i \\
 & \wedge \left(\begin{array}{l} \exists k. \quad k \geq 0 \\ \quad \wedge ticks' = ticks + k \\ \quad \wedge j' = k \end{array} \right)
 \end{aligned}$$

$$\begin{array}{l}
\textbf{for } (i = 0; i < n; i++): \\
\quad \textbf{for } (j = 0; j < i; j++): \\
\quad \quad \text{ticks}++
\end{array}
\left. \vphantom{\begin{array}{l} \textbf{for } (i = 0; i < n; i++): \\ \textbf{for } (j = 0; j < i; j++): \\ \text{ticks}++ \end{array}} \right\}
\begin{array}{l}
i' = n \\
\wedge n' = n \\
\wedge j' = i \\
\wedge \left(\begin{array}{l} \exists k. k \geq 0 \\ \wedge \text{ticks}' = \text{ticks} + \frac{k(k+1)}{2} + ki \\ \wedge i' = i + k \end{array} \right)
\end{array}$$

Warm up: the linear case

Suppose loop body formula $F(\mathbf{x}, \mathbf{x}')$ is *linear*.

Goal: find a linear system $\mathbf{y}' = A\mathbf{y} + \mathbf{b}$ + linear transformation T s.t

$$F(\mathbf{x}, \mathbf{x}') \models (T\mathbf{x}') = A(T\mathbf{x}) + \mathbf{b}$$

W

Binary search: project onto *ticks*, (*hi* - *lo*)

Suppose loop body formula $F(\mathbf{x}, \mathbf{x}')$ is *linear*.

Goal: find a linear system $\mathbf{y}' = A\mathbf{y} + \mathbf{b}$ + linear transformation T s.t

$$F(\mathbf{x}, \mathbf{x}') \models (T\mathbf{x}') = A(T\mathbf{x}) + \mathbf{b}$$

Warm up: the linear case

Suppose loop body formula $F(\mathbf{x}, \mathbf{x}')$ is *linear*.

Goal: find a linear system $\mathbf{y}' = A\mathbf{y} + \mathbf{b}$ + linear transformation T s.t

$$F(\mathbf{x}, \mathbf{x}') \models (T\mathbf{x}') = A(T\mathbf{x}) + \mathbf{b}$$

Algorithm:

- 1 Compute the *affine hull* of F by sampling linearly independent models of F using an SMT solver.

Result is system of (all) equations $A\mathbf{x}' = B\mathbf{x} + \mathbf{c}$ entailed by $F(\mathbf{x}, \mathbf{x}')$

Warm up: the linear case

Suppose loop body formula $F(\mathbf{x}, \mathbf{x}')$ is *linear*.

Goal: find a linear system $\mathbf{y}' = A\mathbf{y} + \mathbf{b}$ + linear transformation T s.t

$$F(\mathbf{x}, \mathbf{x}') \models (T\mathbf{x}') = A(T\mathbf{x}) + \mathbf{b}$$

Algorithm:

- 1 Compute the *affine hull* of F by sampling linearly independent models of F using an SMT solver.

Result is system of (all) equations $A\mathbf{x}' = B\mathbf{x} + \mathbf{c}$ entailed by $F(\mathbf{x}, \mathbf{x}')$

- 2 Fixpoint computation:

We have: $A\mathbf{x}' = B\mathbf{x} + \mathbf{c}$



Linear transformation T

We need: $\mathbf{y}' = B\mathbf{y} + \mathbf{c}$

Warm up: the linear case

Suppose loop body formula $F(\mathbf{x}, \mathbf{x}')$ is *linear*.

Goal: find a linear system $\mathbf{y}' = A\mathbf{y} + \mathbf{b}$ + linear transformation T s.t

$$F(\mathbf{x}, \mathbf{x}') \models (T\mathbf{x}') = A(T\mathbf{x}) + \mathbf{b}$$

Algorithm:

- 1 Compute the *affine hull* of F by sampling linearly independent models of F using an SMT solver.

Result is system of (all) equations $A\mathbf{x}' = B\mathbf{x} + \mathbf{c}$ entailed by $F(\mathbf{x}, \mathbf{x}')$

- 2 Fixpoint computation:

$$\text{We have: } A\mathbf{x}' = B\mathbf{x} + \mathbf{c}$$



$$T_0\mathbf{x}' = T_0B\mathbf{x} + T_0\mathbf{c}$$

$$\text{We need: } \mathbf{y}' = B\mathbf{y} + \mathbf{c}$$

Warm up: the linear case

Suppose loop body formula $F(\mathbf{x}, \mathbf{x}')$ is *linear*.

Goal: find a linear system $\mathbf{y}' = A\mathbf{y} + \mathbf{b}$ + linear transformation T s.t

$$F(\mathbf{x}, \mathbf{x}') \models (T\mathbf{x}') = A(T\mathbf{x}) + \mathbf{b}$$

Algorithm:

- 1 Compute the *affine hull* of F by sampling linearly independent models of F using an SMT solver.

Result is system of (all) equations $A\mathbf{x}' = B\mathbf{x} + \mathbf{c}$ entailed by $F(\mathbf{x}, \mathbf{x}')$

- 2 Fixpoint computation:

We have: $A\mathbf{x}' = B\mathbf{x} + \mathbf{c}$



$$T_0\mathbf{x}' = T_0B\mathbf{x} + T_0\mathbf{c}$$



$$T_1\mathbf{x}' = T_1B\mathbf{x} + T_1\mathbf{c}$$



We need: $\mathbf{y}' = B\mathbf{y} + \mathbf{c}$

} computes *best* abstraction

Reasoning about non-linear arithmetic

$$\begin{array}{l}
 \text{for } (i = 0; i < n; i++): \\
 \quad \text{if } (*): \text{ continue} \\
 \quad \text{for } (j = 0; j < n; j++): \\
 \quad \quad \text{for } (k = 0; k < n; k++): \\
 \quad \quad \quad \text{ticks}++
 \end{array}
 \left\{ \begin{array}{l}
 i' = i + 1 \\
 \wedge i < n \\
 \wedge n' = n \\
 \left(\begin{array}{l}
 \text{ticks}' = \text{ticks} \\
 \wedge j' = j \\
 \wedge k' = k
 \end{array} \right) \\
 \wedge \left(\begin{array}{l}
 \exists y \geq 0. \\
 \vee \left(\begin{array}{l}
 \text{ticks}' = \text{ticks} + y \times n \\
 \wedge j' = y = n \\
 \wedge k' = n
 \end{array} \right)
 \end{array} \right)
 \end{array} \right)
 \end{array}$$

$$\begin{array}{l}
 \text{for } (i = 0; i < n; i++): \\
 \quad \text{if } (*): \text{ continue} \\
 \quad \text{for } (j = 0; j < n; j++): \\
 \quad \quad \text{for } (k = 0; k < n; k++): \\
 \quad \quad \quad \text{ticks}++
 \end{array}
 \left\{
 \begin{array}{l}
 i' = i + 1 \\
 \wedge i < n \\
 \wedge n' = n \\
 \wedge \left(\begin{array}{l} \text{ticks}' = \text{ticks} \\ \wedge j' = j \\ \wedge k' = k \end{array} \right) \\
 \vee \left(\begin{array}{l} \exists y \geq 0. \\ \text{ticks}' = \text{ticks} + y \times n \\ \wedge i' = y = n \\ \wedge k' = n \end{array} \right)
 \end{array}
 \right)$$

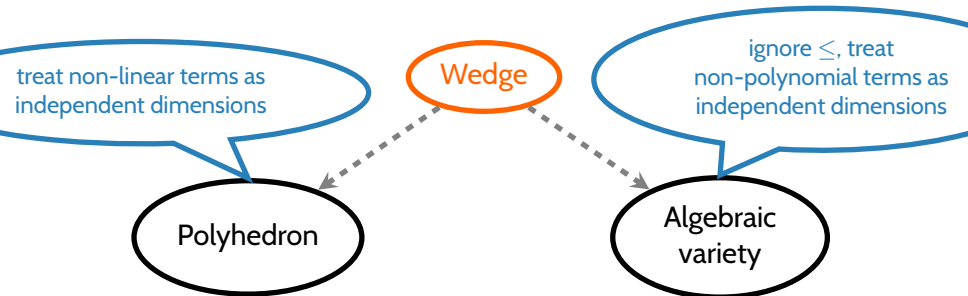
$\text{ticks}' \leq \text{ticks} + n^2$

The *wedge* abstract domain

- The wedge domain is an abstract domain for reasoning about non-linear integer/rational arithmetic
 - The properties expressible by wedges correspond to the *conjunctive fragment* of non-linear arithmetic ($x \times y$, x/y , x^y , $\log_x(y)$, $x \bmod y$, ...)

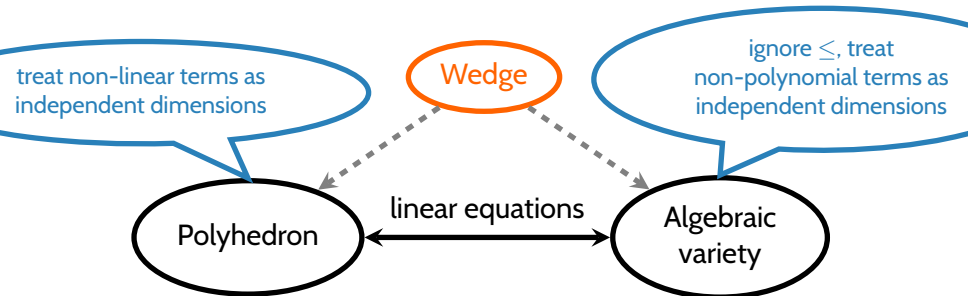
The *wedge* abstract domain

- The wedge domain is an abstract domain for reasoning about non-linear integer/rational arithmetic
 - The properties expressible by wedges correspond to the *conjunctive fragment* of non-linear arithmetic ($x \times y$, x/y , x^y , $\log_x(y)$, $x \bmod y$, ...)



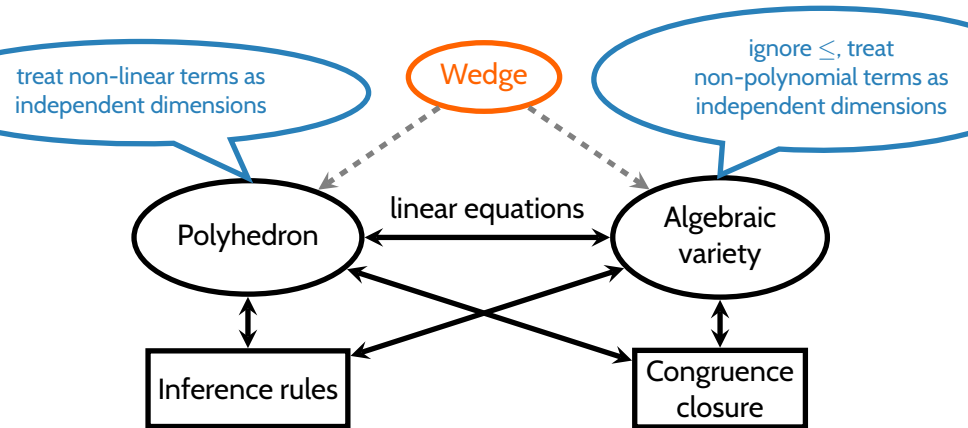
The *wedge* abstract domain

- The wedge domain is an abstract domain for reasoning about non-linear integer/rational arithmetic
 - The properties expressible by wedges correspond to the *conjunctive fragment* of non-linear arithmetic ($x \times y$, x/y , x^y , $\log_x(y)$, $x \bmod y$, ...)



The *wedge* abstract domain

- The wedge domain is an abstract domain for reasoning about non-linear integer/rational arithmetic
 - The properties expressible by wedges correspond to the *conjunctive fragment* of non-linear arithmetic ($x \times y$, x/y , x^y , $\log_x(y)$, $x \bmod y$, ...)



Symbolic abstraction

$$i' = i + 1$$

$$\wedge i < n$$

$$\wedge n' = n$$

$$\wedge \left(\begin{array}{l} \left(\begin{array}{l} \text{ticks}' = \text{ticks} \\ \wedge j' = j \\ \wedge k' = k \end{array} \right) \\ \vee \left(\begin{array}{l} \exists y \geq 0. \\ \left(\begin{array}{l} \text{ticks}' = \text{ticks} + y \times n \\ \wedge j' = y = n \\ \wedge k' = n \end{array} \right) \end{array} \right) \end{array} \right)$$

Symbolic abstraction

$$\left(\begin{array}{l} i' = i + 1 \\ \wedge i < n \\ \wedge n' = n \\ \wedge \text{ticks}' = \text{ticks} \\ \wedge j' = j \end{array} \right) \vee \left(\begin{array}{l} i' = i + 1 \\ \wedge i < n \\ \wedge n' = n \\ \wedge \text{ticks}' = \text{ticks} + sk_y \times n \\ \wedge j' = sk_y = n \\ \wedge k' = n \end{array} \right)$$

Symbolic abstraction

$$\left(\begin{array}{l} i' = i + 1 \\ \wedge i < n \\ \wedge n' = n \\ \wedge \text{ticks}' = \text{ticks} \\ \wedge j' = j \\ \wedge 0 \leq n \times n \end{array} \right) \vee \left(\begin{array}{l} i' = i + 1 \\ \wedge i < n \\ \wedge n' = n \\ \wedge \text{ticks}' = \text{ticks} + n \times n \\ \wedge j' = n \\ \wedge k' = n \\ \wedge 0 \leq n \times n \end{array} \right)$$

Symbolic abstraction

$$\left(\begin{array}{l} i' = i + 1 \\ \wedge i < n \\ \wedge n' = n \\ \wedge \text{ticks} \leq \text{ticks}' \leq \text{ticks} + n \times n \\ \wedge j' = j \\ \wedge 0 \leq n \times n \end{array} \right)$$

Extracting recurrences

Given: non-linear transition formula $F(\mathbf{x}, \mathbf{x}')$

- 1 Compute wedge w that over-approximates F
- 2 Extract recurrences from w

Extracting recurrences

Given: non-linear transition formula $F(\mathbf{x}, \mathbf{x}')$

- 1 Compute wedge w that over-approximates F
- 2 Extract recurrences from w

Class of extractable recurrences:

$$(T\mathbf{x}') = A(T\mathbf{x}) + \mathbf{t}$$

Additive term $\mathbf{t}$ involves polynomials & exponentials.

How can we solve recurrence equations?

Operational Calculus Recurrence Solver [Berg 1967]

Operational calculus is an algebra of *infinite sequences*. Idea:

- 1 Translate recurrence into equation in operational calculus

- $x^{(k+1)} = x^{(k)} + 1 \rightsquigarrow qx - (q - \underline{1})\underline{x}_0 = x + \underline{1}$

- 2 Solve the equation

- $\underline{x} = \underline{x}_0 + \frac{\underline{1}}{q - \underline{1}}$

- 3 Translate solution back

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

Operational Calculus

Field of operators:

- Operator is a sequence with finitely many **negative positions**

$$a = (a_{-2}, a_{-1} \parallel a_0, a_1, a_2, \dots)$$

$$b = (\parallel b_0, b_1, b_2, \dots)$$

- Addition is pointwise: $(a + b)_i \triangleq a_i + b_i$
- Multiplication is convolution difference:

$$(ab)_n = \sum_{i=-\infty}^n a_i b_{n-i} + \sum_{i=-\infty}^{n-1} a_i b_{n-i-1}$$

Operational Calculus

Field of operators:

- Operator is a sequence with finitely many **negative positions**

$$a = (\textcolor{blue}{a}_{-2}, \textcolor{blue}{a}_{-1} \parallel a_0, a_1, a_2, \dots)$$

$$b = (\parallel b_0, b_1, b_2, \dots)$$

- Addition is pointwise: $(a + b)_i \triangleq a_i + b_i$
- Multiplication is convolution difference:

$$(ab)_n = \sum_{i=-\infty}^n a_i b_{n-i} + \sum_{i=-\infty}^{n-1} a_i b_{n-i-1}$$

- Left shift operator $q = (\textcolor{blue}{1} \parallel 1, 1, 1, \dots)$

$$qa = (\textcolor{blue}{a}_{-2} \textcolor{blue}{a}_{-1} a_0 \parallel a_1, a_2, a_3, \dots)$$

Recurrence $\rightarrow$ operational calculus

Recurrences are equations in operational calculus

$$x^{(k+1)} = x^k + t \rightsquigarrow qx - (q - \underline{1})x_0 = x + \mathcal{T}_k(t)$$

- Think of x as an sequence ($\parallel x_0, x_1, x_2, \dots$)

Recurrence $\rightarrow$ operational calculus

Recurrences are equations in operational calculus

$$x^{(k+1)} = x^k + t \rightsquigarrow qx - (q - \underline{1})x_0 = x + \mathcal{T}_k(t)$$

- Think of x as an sequence ($\parallel x_0, x_1, x_2, \dots$)
- Use left-shift operator to write recurrence as an equation

$$qx = (\textcolor{blue}{x}_0 \parallel x_1, x_2, x_3, \dots)$$

$$(q - 1)\underline{x}_0 = (\textcolor{blue}{x}_0 \parallel 0, 0, 0, \dots)$$

$$qx - (q - 1)\underline{x}_0 = (\parallel x_1, x_2, x_3, \dots)$$

Recurrence $\rightarrow$ operational calculus

Recurrences are equations in operational calculus

$$x^{(k+1)} = x^k + t \rightsquigarrow qx - (q - \underline{1})x_0 = x + \mathcal{T}_k(t)$$

- Think of x as an sequence ($\parallel x_0, x_1, x_2, \dots$)
- Use left-shift operator to write recurrence as an equation

$$qx = (\textcolor{blue}{x}_0 \parallel x_1, x_2, x_3, \dots)$$

$$(q - 1)\underline{x}_0 = (\textcolor{blue}{x}_0 \parallel 0, 0, 0, \dots)$$

$$qx - (q - 1)\underline{x}_0 = (\parallel x_1, x_2, x_3, \dots)$$

Can translate any expression in the grammar

$$s, t \in Expr(k) ::= c \in \mathbb{Q} \mid k \mid c^k \mid s + t \mid st$$

$$\mathcal{T}_k(c) = \underline{c}$$

$$\mathcal{T}_k(ct) = \underline{c}\mathcal{T}_k(t)$$

$$\vdots$$

$$\mathcal{T}_k(s + t) = \mathcal{T}_k(s) + \mathcal{T}_k(t)$$

$$\mathcal{T}_k(k) = \frac{\underline{1}}{q - \underline{1}}$$

$$\vdots$$

Operational Calculus $\rightarrow$ classical algebra

$$\mathcal{T}_k(c) = \underline{c}$$

$$\mathcal{T}_k(ct) = \underline{c}\mathcal{T}_k(t)$$

$$\mathcal{T}_k(s+t) = \mathcal{T}_k(s) + \mathcal{T}_k(t)$$

$$\mathcal{T}_k(k) = \frac{\underline{1}}{q - \underline{1}}$$

$$\vdots$$

$$\mathcal{T}_k^{-1}(\underline{c}) = c$$

$$\mathcal{T}_k^{-1}(\underline{c}t) = c\mathcal{T}_k^{-1}(t)$$

$$\mathcal{T}_k^{-1}(s+t) = \mathcal{T}_k^{-1}(s) + \mathcal{T}_k^{-1}(t)$$

$$\mathcal{T}_k^{-1}\left(\frac{\underline{1}}{q - \underline{1}}\right) = k$$

$$\vdots$$

Operational Calculus $\rightarrow$ classical algebra

$$\mathcal{T}_k(c) = \underline{c}$$

$$\mathcal{T}_k(ct) = \underline{c}\mathcal{T}_k(t)$$

$$\mathcal{T}_k(s + t) = \mathcal{T}_k(s) + \mathcal{T}_k(t)$$

$$\mathcal{T}_k(k) = \frac{\underline{1}}{q - \underline{1}}$$

$$\vdots$$

$$\mathcal{T}_k^{-1}(\underline{c}) = c$$

$$\mathcal{T}_k^{-1}(\underline{c}t) = c\mathcal{T}_k^{-1}(t)$$

$$\mathcal{T}_k^{-1}(s + t) = \mathcal{T}_k^{-1}(s) + \mathcal{T}_k^{-1}(t)$$

$$\mathcal{T}_k^{-1}\left(\frac{\underline{1}}{q - \underline{1}}\right) = k$$

$$\vdots$$

$$\mathcal{T}_k^{-1}(t) = ?$$

Operational Calculus $\rightarrow$ classical algebra translation is not complete!

Operational Calculus $\rightarrow$ classical algebra

$$\mathcal{T}_k(c) = \underline{c}$$

$$\mathcal{T}_k(ct) = \underline{c}\mathcal{T}_k(t)$$

$$\mathcal{T}_k(s+t) = \mathcal{T}_k(s) + \mathcal{T}_k(t)$$

$$\mathcal{T}_k(k) = \frac{\underline{1}}{q - \underline{1}}$$

$\vdots$

$$\mathcal{T}_k^{-1}(\underline{c}) = c$$

$$\mathcal{T}_k^{-1}(\underline{c}t) = c\mathcal{T}_k^{-1}(t)$$

$$\mathcal{T}_k^{-1}(s+t) = \mathcal{T}_k^{-1}(s) + \mathcal{T}_k^{-1}(t)$$

$$\mathcal{T}_k^{-1}\left(\frac{\underline{1}}{q - \underline{1}}\right) = k$$

$\vdots$

$$\mathcal{T}_k^{-1}(t) = f_t(k)$$

Operational Calculus

Implicitly interpreted function

complete!

Operational Calculus $\rightarrow$ classical algebra

$$\mathcal{T}_k(c) = \underline{c}$$

$$\mathcal{T}_k(ct) = \underline{c}\mathcal{T}_k(t)$$

$$\mathcal{T}_k(s+t) = \mathcal{T}_k(s) + \mathcal{T}_k(t)$$

$$\mathcal{T}_k(k) = \frac{\underline{1}}{q - \underline{1}}$$

$$\vdots$$

$$\mathcal{T}_k(f_t(k)) = t$$

$$\mathcal{T}_k^{-1}(\underline{c}) = c$$

$$\mathcal{T}_k^{-1}(\underline{c}t) = c\mathcal{T}_k^{-1}(t)$$

$$\mathcal{T}_k^{-1}(s+t) = \mathcal{T}_k^{-1}(s) + \mathcal{T}_k^{-1}(t)$$

$$\mathcal{T}_k^{-1}\left(\frac{\underline{1}}{q - \underline{1}}\right) = k$$

$$\vdots$$

$$\mathcal{T}_k^{-1}(t) = f_t(k)$$

Operational Calculus

Implicitly interpreted function

complete!

Experiments

ICRA: built on top of Z3, Apron.

Analyzes recursive procedures via [Kincaid, Breck, Boroujeni, Reps PLDI 2017]

Benchmark Suite	Total	ICRA		UAut.		CPA		SEA	
	#A	Time	#A	Time	#A	Time	#A	Time	#A
HOLA	46	123.5	33	1571.9	20	2004.1	11	259.5	38
functional	21	77.9	11	732.8	0	1155.7	0	722.3	2
relational	10	8.1	10	473	0	603.0	0	121.8	4
Total	77	209.5	54	2777.7	20	3762.8	11	1103.6	44

Contributions:

- Wedge abstract domain
- Algorithm for extracting recurrences from loop bodies with control flow & non-determinism
- Recurrence solver that avoids algebraic numbers

Result: non-linear invariant generation for arbitrary loops