Non-Linear Reasoning for Invariant Synthesis

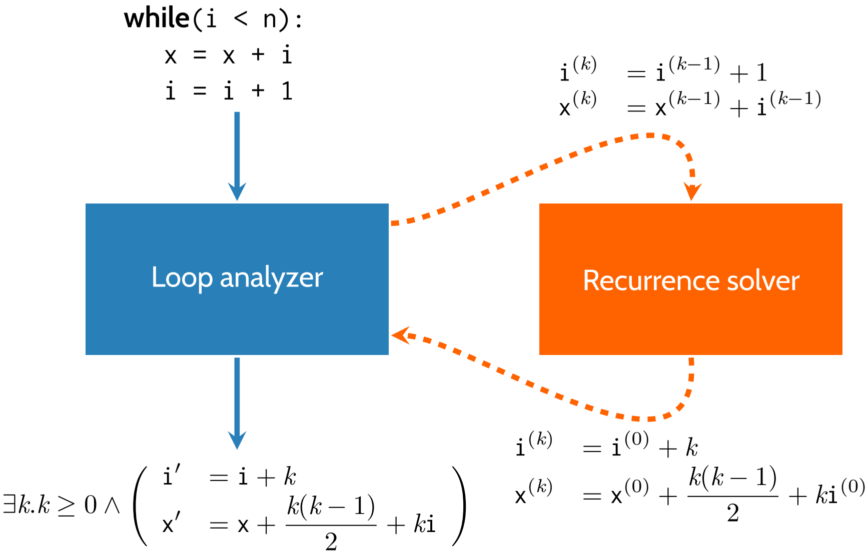
Zachary Kincaid[1] John Cyphert[2] Jason Breck[2] Thomas Reps[2,3]

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

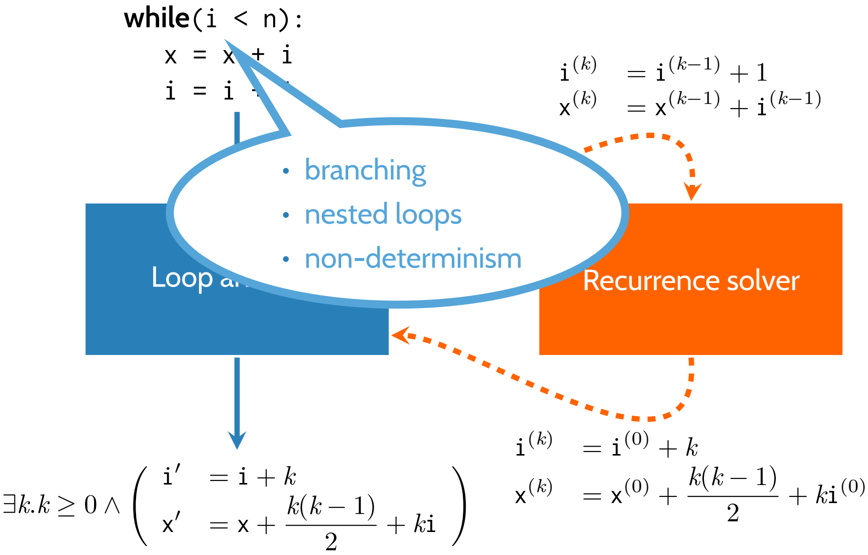
January 12, 2018 @ 15:50

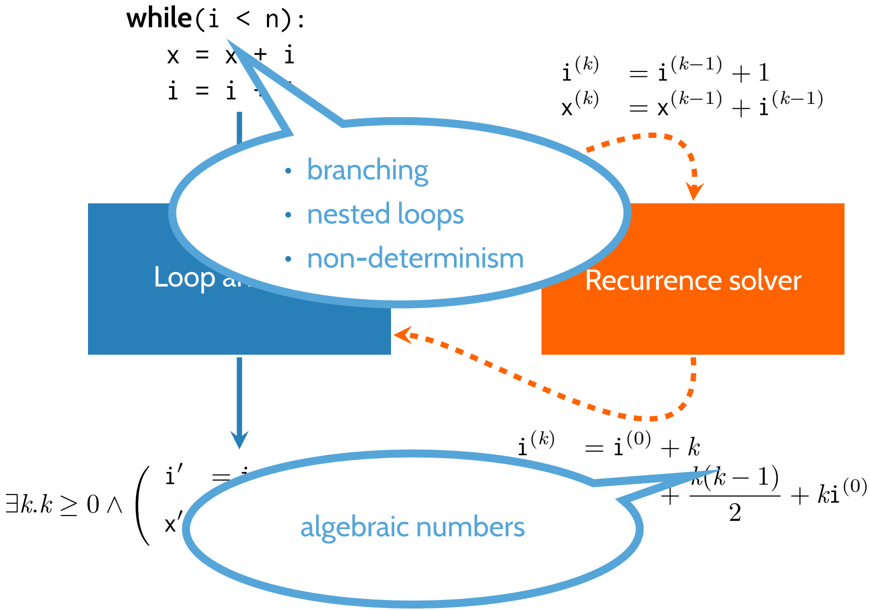
The problem: generating non-linear numerical loop invariants

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



while(i < n):
x = x + i
i = i + 1
$\mathtt{i}^{(k)} = \mathtt{i}^{(k-1)} + 1$
$\mathtt{x}^{(k)} = \mathtt{x}^{(k-1)} + \mathtt{i}^{(k-1)}$
Loop analyzer
Recurrence solver
$\mathtt{i}^{(k)} = \mathtt{i}^{(0)} + k$
$\mathtt{x}^{(k)} = \mathtt{x}^{(0)} + \frac{k(k-1)}{2} + k\mathtt{i}^{(0)}$
$\exists k. k \geq 0 \wedge \left(\begin{array}{ll} \mathtt{i}' & = \mathtt{i} + k \\ \mathtt{x}' & = \mathtt{x} + \frac{k(k-1)}{2} + k\mathtt{i} \end{array} \right)$


while(i < n):
x = x + i
i = i +
branching
nested loops
non-determinism
Loop a
Recurrence solver
$\mathtt{i}^{(k)} = \mathtt{i}^{(k-1)} + 1$
$\mathtt{x}^{(k)} = \mathtt{x}^{(k-1)} + \mathtt{i}^{(k-1)}$
$\mathtt{i}^{(k)} = \mathtt{i}^{(0)} + k$
$\mathtt{x}^{(k)} = \mathtt{x}^{(0)} + \frac{k(k-1)}{2} + k\mathtt{i}^{(0)}$
$\exists k. k \geq 0 \land \left(\begin{array}{ll} \mathtt{i}' & = \mathtt{i} + k \\ \mathtt{x}' & = \mathtt{x} + \frac{k(k-1)}{2} + k\mathtt{i} \end{array} \right)$



while(i < n):
x = x + i
i = i +
$\mathtt{i}^{(k)} = \mathtt{i}^{(k-1)} + 1$
$\mathtt{x}^{(k)} = \mathtt{x}^{(k-1)} + \mathtt{i}^{(k-1)}$
branching
nested loops
non-determinism
Loop a
Recurrence solver
$\mathtt{i}^{(k)} = \mathtt{i}^{(0)} + k$
$+ \frac{k(k-1)}{2} + k\mathtt{i}^{(0)}$
$\exists k. k \geq 0 \wedge$
$\mathtt{i}'$
$\mathtt{x}'$
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
```

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

```
binar
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
```

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

$$
\begin{aligned}
&= ticks + 1\\
&\land mid \quad lo + (hi - lo)/2\\
&\land((A[mid] < target\\
&\quad \land lo' = mid + 1\\
&\quad \land hi' = hi)\\
&\ \lor(A[mid] > target\\
&\quad \land lo' = lo\\
&\quad \land hi' = mid - 1))
\end{aligned}
$$

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

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

```
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
```

$$
\begin{aligned}
&\quad = ticks + 1\\
&\wedge mid \quad lo + (hi - lo)/2\\
&\wedge((A[mid] < target\\
&\qquad \wedge lo' = mid + 1\\
&\qquad \wedge hi' = hi)\\
&\quad \vee(A[mid] > target\\
&\qquad \wedge lo' = lo\\
&\qquad \wedge hi' = mid - 1))
\end{aligned}
$$

```
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++
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$
\left.\right\}
\begin{array}{l}
\quad i' = n \\
\wedge n' = n \\
\wedge j' = i \\
\wedge \left(\begin{array}{l} \exists k . k \geq 0 \\ \qquad \wedge ticks' = ticks + \dfrac{k(k+1)}{2} + ki \\ \qquad \wedge i' = i + k \end{array} \right)
\end{array}
$$

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:

$$\begin{array}{l} \text{We have: } A\mathbf{x}' = B\mathbf{x} + \mathbf{c} \\ \Big\downarrow \text{ Linear transformation } T \\ \text{We need: } \mathbf{y}' = B\mathbf{y} + \mathbf{c} \end{array}$$

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

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

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

$$\Downarrow$$

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

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:

$$\left.\begin{array}{c} \text{We have: } A\mathbf{x}' = B\mathbf{x} + \mathbf{c} \\ \downarrow \\ T_0\mathbf{x}' = T_0 B\mathbf{x} + T_0\mathbf{c} \\ \downarrow \\ T_1\mathbf{x}' = T_1 B\mathbf{x} + T_1\mathbf{c} \end{array}\right\} \text{computes best abstraction}$$

$$\vdots$$

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

Reasoning about non-linear arithmetic

```
for (i = 0; i < n; i++):
  if (*): continue
  for (j = 0; j < n; j++):
    for (k = 0; k < n; k++):
      ticks++
```

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

```
for (i = 0; i < n; i++):
  if (*): continue
  for (j = 0; j < n; j++):
    for (k = 0; k < n; k++):
      ticks++
```

$$
\left.\begin{array}{l}\\\\\\\end{array}\right\}
\begin{array}{l}
\;\; \mathsf{i}' = \mathsf{i} + 1 \\
\land \mathsf{i} < \mathsf{n} \\
\land \mathsf{n}' = \mathsf{n} \\
\land \left(\begin{array}{l}
\left(\begin{array}{l} \;\; \mathsf{ticks}' = \mathsf{ticks} \\ \land \mathsf{j}' = \mathsf{j} \\ \land \mathsf{k}' = \mathsf{k} \end{array} \right) \\
\lor \left(\begin{array}{l} \exists y \geq 0. \\ \left(\begin{array}{l} \;\; \mathsf{ticks}' = \mathsf{ticks} + y \times n \\ \land \mathsf{j}' = y = \mathsf{n} \\ \land \mathsf{k}' = \mathsf{n} \end{array} \right) \end{array} \right)
\end{array} \right)
\end{array}
$$

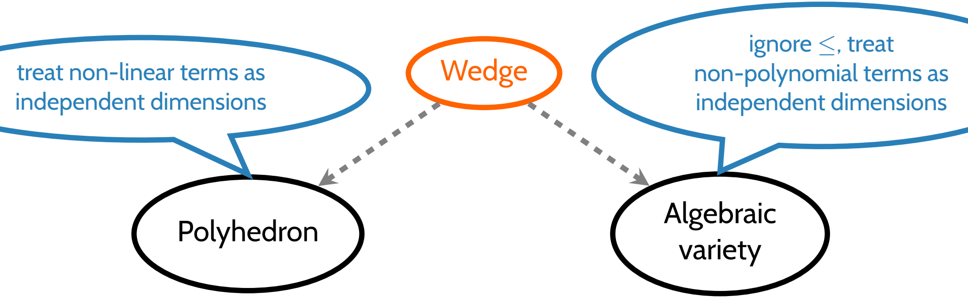
$\mathsf{ticks}' \leq \mathsf{ticks} + \mathsf{n}^2$

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 ($x \times y, x/y, x^y, \log_x(y), x$ *mod* $y, ...$)

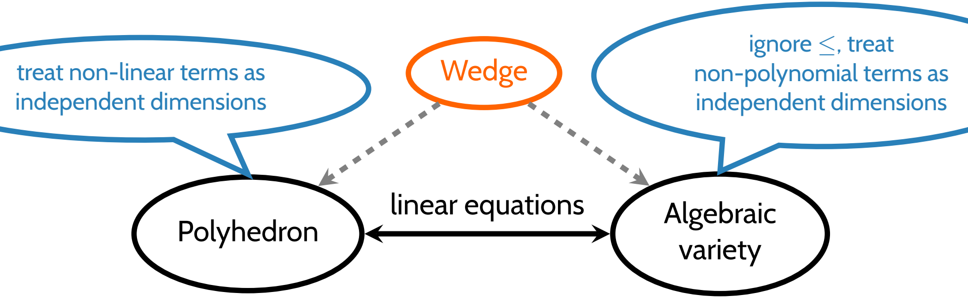
The *wedge* abstract domain

- 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 ($x \times y, x/y, x^y, \log_x(y), x$ *mod* $y, \ldots$)



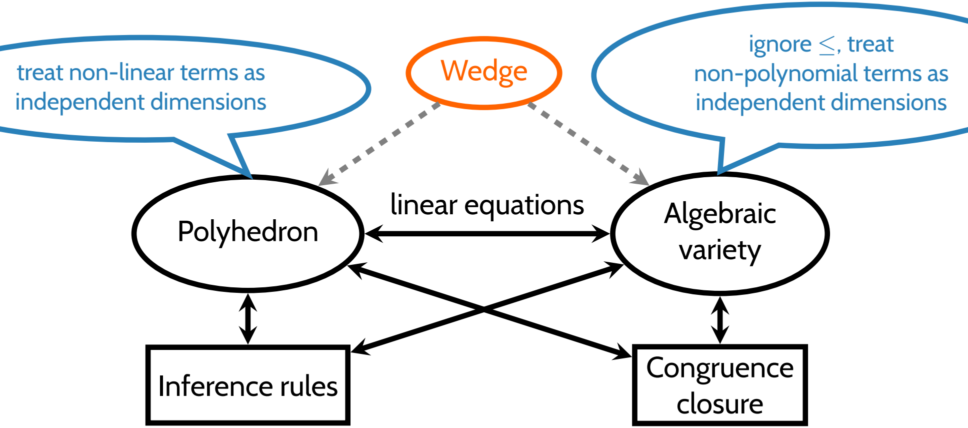
The *wedge* abstract domain

- 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 ($x \times y, x/y, x^y, \log_x(y), x$ *mod* $y, \ldots$)



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 ($x \times y, x/y, x^y, \log_x(y), x$ *mod* $y, \ldots$)



Symbolic abstraction

$$
\begin{array}{l}
\mathtt{i}' = \mathtt{i} + 1 \\
\land \mathtt{i} < \mathtt{n} \\
\land \mathtt{n}' = \mathtt{n} \\
\land \left(\begin{array}{l}
 \left(\begin{array}{l} \mathtt{ticks}' = \mathtt{ticks} \\ \land \mathtt{j}' = \mathtt{j} \\ \land \mathtt{k}' = \mathtt{k} \end{array} \right) \\
\lor \left(\begin{array}{l} \exists y \geq 0. \\ \left(\begin{array}{l} \mathtt{ticks}' = \mathtt{ticks} + y \times n \\ \land \mathtt{j}' = y = \mathtt{n} \\ \land \mathtt{k}' = \mathtt{n} \end{array} \right) \end{array} \right)
\end{array} \right)
\end{array}
$$

Symbolic abstraction

$$\begin{pmatrix} \;\; \mathtt{i}' = \mathtt{i} + 1 \\ \wedge \mathtt{i} < \mathsf{n} \\ \wedge \mathsf{n}' = \mathsf{n} \\ \wedge \mathtt{ticks}' = \mathtt{ticks} \\ \wedge \mathtt{j}' = \mathtt{j} \end{pmatrix} \vee \begin{pmatrix} \;\; \mathtt{i}' = \mathtt{i} + 1 \\ \wedge \mathtt{i} < \mathsf{n} \\ \wedge \mathsf{n}' = \mathsf{n} \\ \wedge \mathtt{ticks}' = \mathtt{ticks} + \mathsf{sk}_y \times \mathsf{n} \\ \wedge \mathtt{j}' = \mathsf{sk}_y = \mathsf{n} \\ \wedge \mathsf{k}' = \mathsf{n} \end{pmatrix}$$

Symbolic abstraction

$$\begin{pmatrix} \quad \mathtt{i}' = \mathtt{i} + 1 \\ \wedge \mathtt{i} < \mathtt{n} \\ \wedge \mathtt{n}' = \mathtt{n} \\ \wedge \mathtt{ticks}' = \mathtt{ticks} \\ \wedge \mathtt{j}' = \mathtt{j} \\ \wedge 0 \leq \mathtt{n} \times \mathtt{n} \end{pmatrix} \vee \begin{pmatrix} \quad \mathtt{i}' = \mathtt{i} + 1 \\ \wedge \mathtt{i} < \mathtt{n} \\ \wedge \mathtt{n}' = \mathtt{n} \\ \wedge \mathtt{ticks}' = \mathtt{ticks} + \mathtt{n} \times \mathtt{n} \\ \wedge \mathtt{j}' = \mathtt{n} \\ \wedge \mathtt{k}' = \mathtt{n} \\ \wedge 0 \leq \mathtt{n} \times \mathtt{n} \end{pmatrix}$$

Symbolic abstraction

$$\begin{pmatrix} \ \ \mathtt{i}' = \mathtt{i} + 1 \\ \wedge \mathtt{i} < \mathtt{n} \\ \wedge \mathtt{n}' = \mathtt{n} \\ \wedge \mathtt{ticks} \leq \mathtt{ticks}' \leq \mathtt{ticks} + \mathtt{n} \times \mathtt{n} \\ \wedge \mathtt{j}' = \mathtt{j} \\ \wedge 0 \leq \mathtt{n} \times \mathtt{n} \end{pmatrix}$$

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
   - $\underline{x} = \underline{x_0} + \dfrac{\underline{1}}{q - \underline{1}}$
3. Translate solution back
   - $x^{(k)} = x^{(0)} + k$

Operational Calculus

Field of operators:

- Operator is a sequence with finitely many negative positions

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

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

- 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

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

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

- 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 $q = (1 \parallel 1, 1, 1, \ldots)$

$$qa = (a_{-2} a_{-1} a_0 \parallel a_1, a_2, a_3, \ldots)$$

Recurrence $\to$ 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 $(\| \; x_0, x_1, x_2, \ldots)$

Recurrence $\to$ 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 $(\| \, x_0, x_1, x_2, \ldots)$
- Use left-shift operator to write recurrence as an equation

$$\begin{aligned} qx &= (x_0 \, \| \, x_1, x_2, x_3, \ldots) \\ (q-1)\underline{x_0} &= (x_0 \, \| \, 0, 0, 0, \ldots) \\ qx - (q-1)\underline{x_0} &= (\| \, x_1, x_2, x_3, \ldots) \end{aligned}$$

Recurrence $\to$ 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 $(\| \; x_0, x_1, x_2, \dots)$
- Use left-shift operator to write recurrence as an equation

$$\begin{aligned} qx &= (x_0 \; \| \; x_1, x_2, x_3, \dots) \\ (q-1)\underline{x_0} &= (x_0 \; \| \; 0, 0, 0, \dots) \\ qx - (q-1)\underline{x_0} &= (\| \; x_1, x_2, x_3, \dots) \end{aligned}$$

Can translate any expression in the grammar

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

$$\begin{aligned} \mathcal{T}_k(c) &= \underline{c} \\ \mathcal{T}_k(ct) &= \underline{c}\mathcal{T}_k(t) \\ &\vdots \end{aligned} \qquad \begin{aligned} \mathcal{T}_k(s+t) &= \mathcal{T}_k(s) + \mathcal{T}_k(t) \\ \mathcal{T}_k(k) &= \frac{\underline{1}}{q - \underline{1}} \\ &\vdots \end{aligned}$$

Operational Calculus $\rightarrow$ classical algebra

$$\begin{aligned}
\mathcal{T}_k(c) &= \underline{c} \\
\mathcal{T}_k(ct) &= \underline{c}\mathcal{T}_k(t) \\
\mathcal{T}_k(s+t) &= \mathcal{T}_k(s) + \mathcal{T}_k(t) \\
\mathcal{T}_k(k) &= \frac{\underline{1}}{q - \underline{1}} \\
&\vdots
\end{aligned}$$

$$\begin{aligned}
\mathcal{T}_k^{-1}(\underline{c}) &= c \\
\mathcal{T}_k^{-1}(\underline{c}t) &= c\mathcal{T}_k^{-1}(t) \\
\mathcal{T}_k^{-1}(s+t) &= \mathcal{T}_k^{-1}(s) + \mathcal{T}_k^{-1}(t) \\
\mathcal{T}_k^{-1}\left(\frac{\underline{1}}{q - \underline{1}}\right) &= k \\
&\vdots
\end{aligned}$$

Operational Calculus $\to$ classical algebra

$$\begin{aligned}
\mathcal{T}_k(c) &= \underline{c} \\
\mathcal{T}_k(ct) &= \underline{c}\mathcal{T}_k(t) \\
\mathcal{T}_k(s+t) &= \mathcal{T}_k(s) + \mathcal{T}_k(t) \\
\mathcal{T}_k(k) &= \frac{\underline{1}}{q-\underline{1}} \\
&\vdots
\end{aligned}$$

$$\begin{aligned}
\mathcal{T}_k^{-1}(\underline{c}) &= c \\
\mathcal{T}_k^{-1}(\underline{c}t) &= c\mathcal{T}_k^{-1}(t) \\
\mathcal{T}_k^{-1}(s+t) &= \mathcal{T}_k^{-1}(s) + \mathcal{T}_k^{-1}(t) \\
\mathcal{T}_k^{-1}(\frac{\underline{1}}{q-\underline{1}}) &= k \\
&\vdots \\
\mathcal{T}_k^{-1}(t) &= ?
\end{aligned}$$

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

Operational Calculus $\rightarrow$ classical algebra

$$
\begin{aligned}
\mathcal{T}_k(c) &= \underline{c} \\
\mathcal{T}_k(ct) &= \underline{c}\,\mathcal{T}_k(t) \\
\mathcal{T}_k(s+t) &= \mathcal{T}_k(s) + \mathcal{T}_k(t) \\
\mathcal{T}_k(k) &= \frac{\underline{1}}{q-\underline{1}} \\
&\vdots
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{T}_k^{-1}(\underline{c}) &= c \\
\mathcal{T}_k^{-1}(\underline{c}t) &= c\,\mathcal{T}_k^{-1}(t) \\
\mathcal{T}_k^{-1}(s+t) &= \mathcal{T}_k^{-1}(s) + \mathcal{T}_k^{-1}(t) \\
\mathcal{T}_k^{-1}\left(\frac{\underline{1}}{q-\underline{1}}\right) &= k \\
&\vdots \\
\mathcal{T}_k^{-1}(t) &= f_t(k)
\end{aligned}
$$

Implicitly interpreted function

Operational Calcu... complete!

Operational Calculus $\rightarrow$ classical algebra

$$\mathcal{T}_k(c) = \underline{c}$$
$$\mathcal{T}_k(ct) = \underline{c}\mathcal{T}_k(t)$$
$$\mathcal{T}_k(s+t) = \mathcal{T}_k(s) + \mathcal{T}_k(t)$$
$$\mathcal{T}_k(k) = \frac{\underline{1}}{q - \underline{1}}$$
$$\vdots$$
$$\mathcal{T}_k(f_t(k)) = t$$

$$\mathcal{T}_k^{-1}(\underline{c}) = c$$
$$\mathcal{T}_k^{-1}(\underline{c}t) = c\mathcal{T}_k^{-1}(t)$$
$$\mathcal{T}_k^{-1}(s+t) = \mathcal{T}_k^{-1}(s) + \mathcal{T}_k^{-1}(t)$$
$$\mathcal{T}_k^{-1}\left(\frac{\underline{1}}{q - \underline{1}}\right) = k$$
$$\vdots$$
$$\mathcal{T}_k^{-1}(t) = f_t(k)$$

Implicitly interpreted function

Operational Calcul ... complete!

Experiments

ICRA: built on top of Z3, Apron.
Analyzes recursive procedures via [Kincaid, Breck, Boroujeni, Reps PLDI 2017]

Benchmark Suite	Total #A	ICRA Time	ICRA #A	UAut. Time	UAut. #A	CPA Time	CPA #A	SEA Time	SEA #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