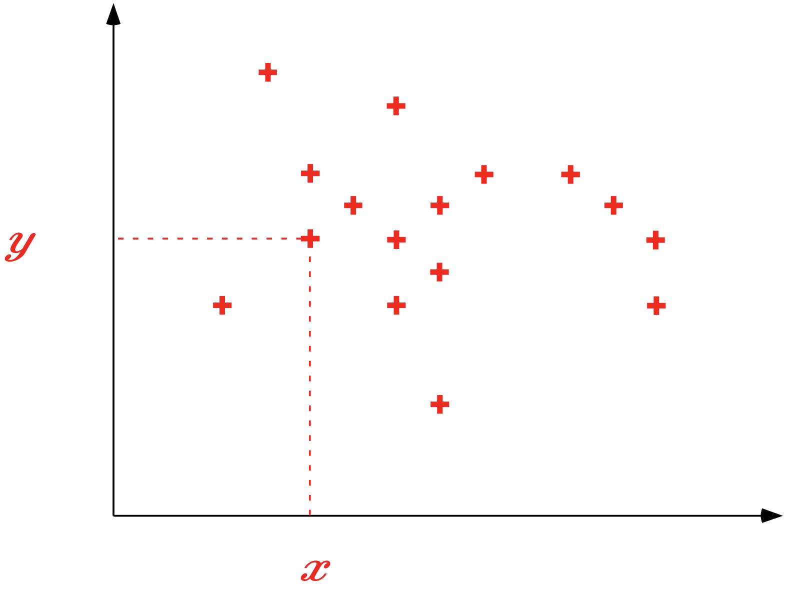
Static Program Analysis

Foundations of Abstract Interpretation

Sebastian Hack, Christian Hammer, Jan Reineke

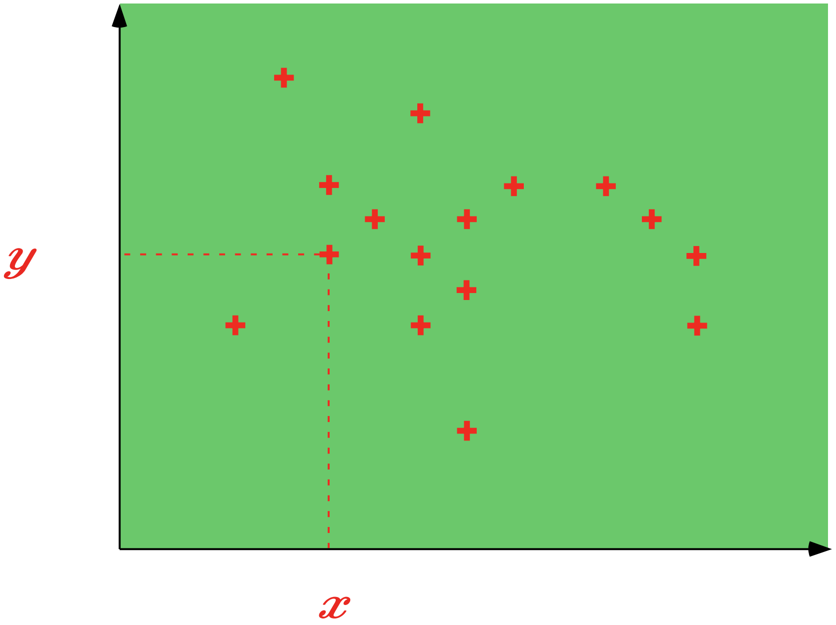
Advanced Lecture, Winter 2014/15

Overview: Numerical Abstractions



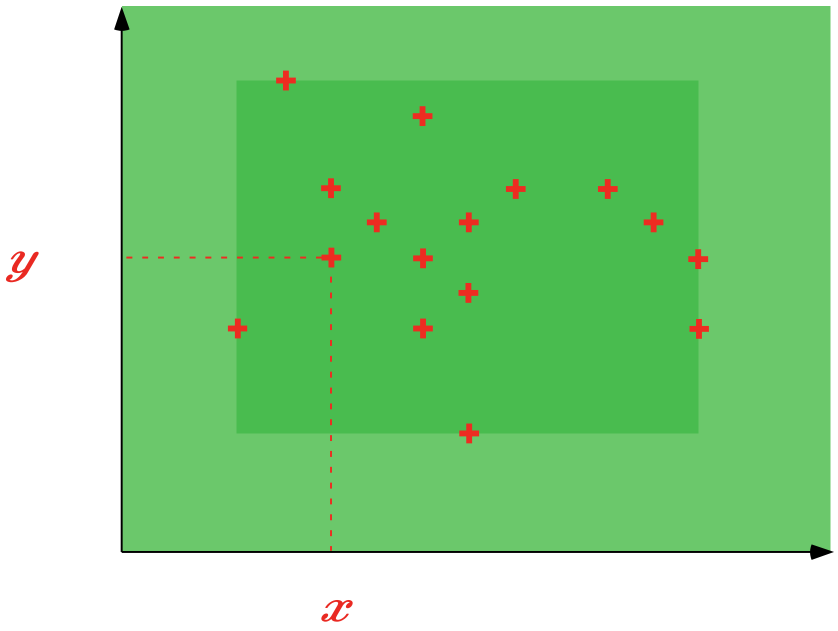
f : : : ; h19; 77i ; : : : ;
h20; 03i ; : : :g

Overview: Numerical Abstractions Signs (Cousot & Cousot, 1979)



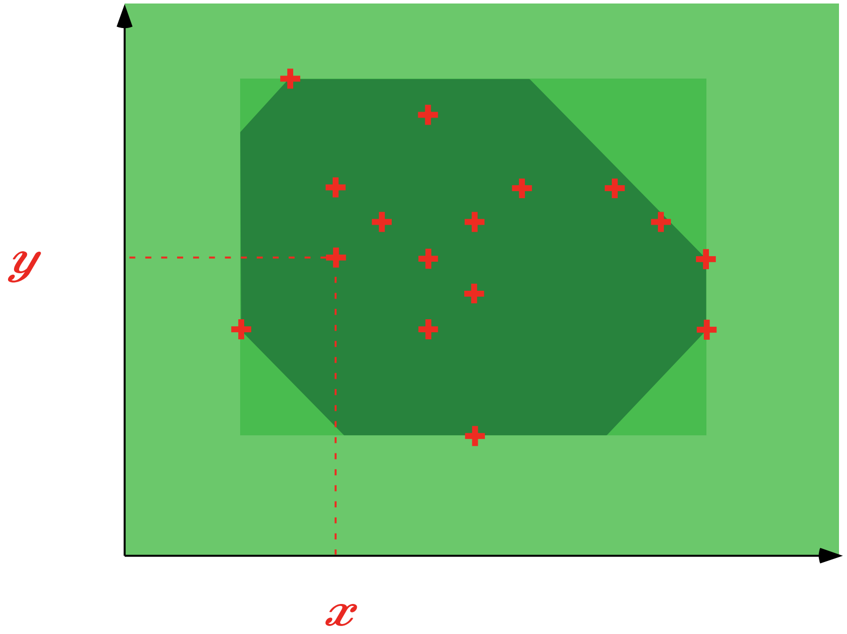
$$\begin{cases} x - 0 \\ y - 0 \end{cases}$$

Overview: Numerical Abstractions
Intervals (Cousot & Cousot, 1976)



$$\begin{cases} \text{x 2 [19; 77]} \\ \text{y 2 [20; 03]} \end{cases}$$

Overview: Numerical Abstractions
Octagons (Mine, 2001)



8

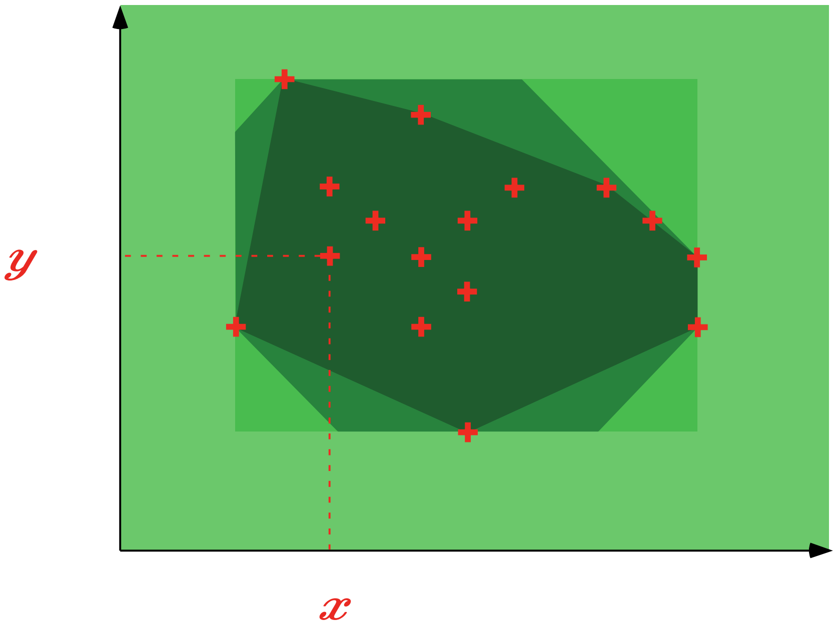
1 » x » 9

x + y » 77

1 » y » 9

x ` y » 99

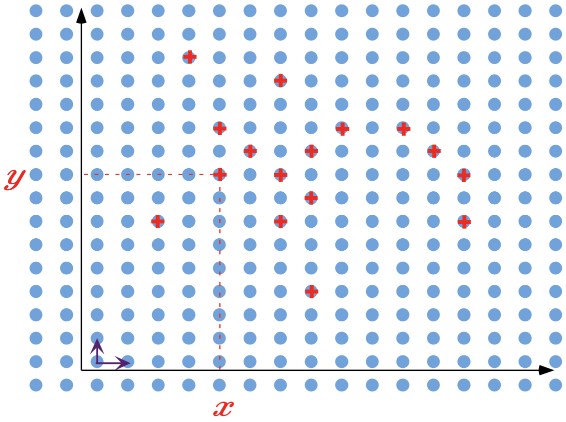
Overview: Numerical Abstractions
Polyhedra (Cousot & Halbwachs, 1978)



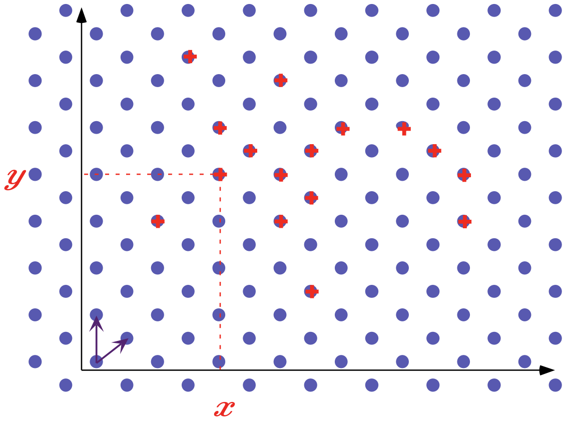
19x + 77y » 2004
20x + 03y – 0

→ *Very Expensive…*

Overview: Numerical Abstractions Simple and Linear Congruences (Granger, 1989+1991)



$$\begin{cases} x = 19 \bmod 77 \\ y = 20 \bmod 99 \end{cases}$$

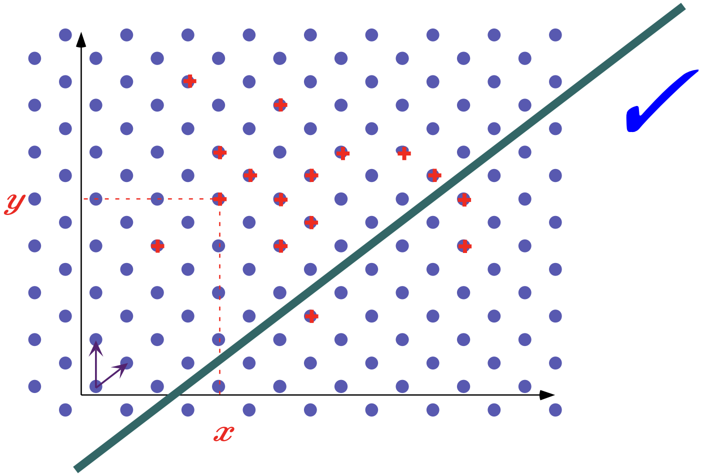


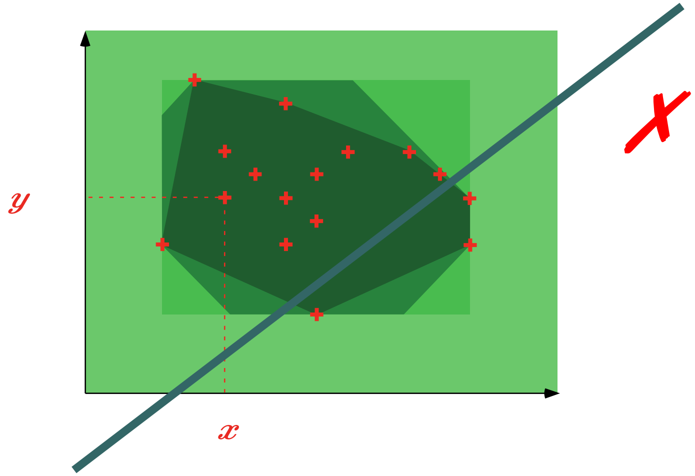
$$\begin{cases} 1x + 9y = 7 \bmod 8 \\ 2x \text{ ` } 1y = 9 \bmod 9 \end{cases}$$

Numerical Abstractions

Which abstraction is the most precise?

Depends on questions you want to answer!

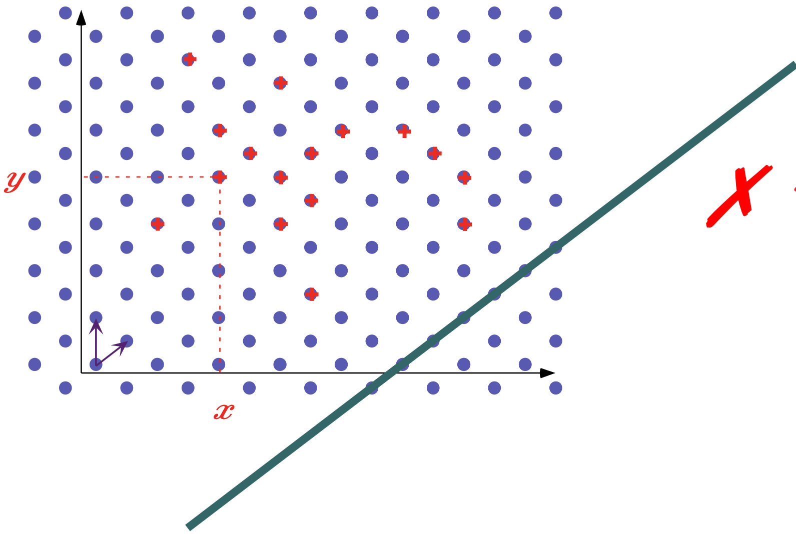


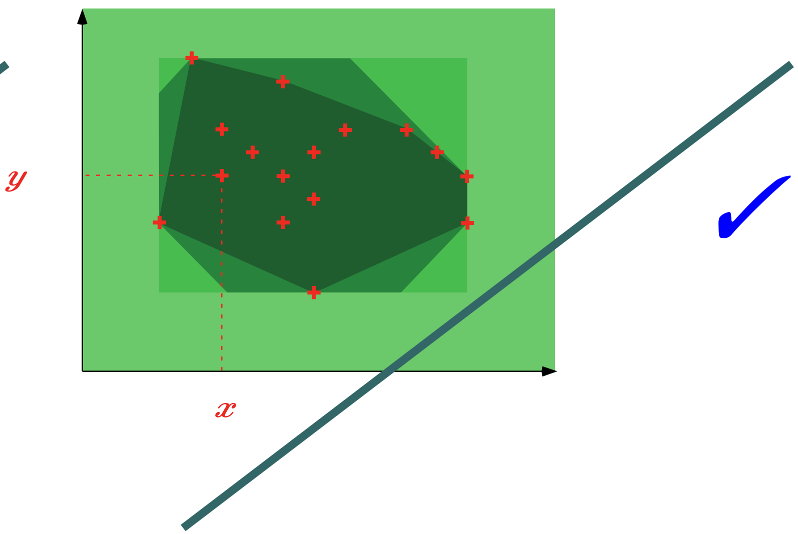


Numerical Abstractions

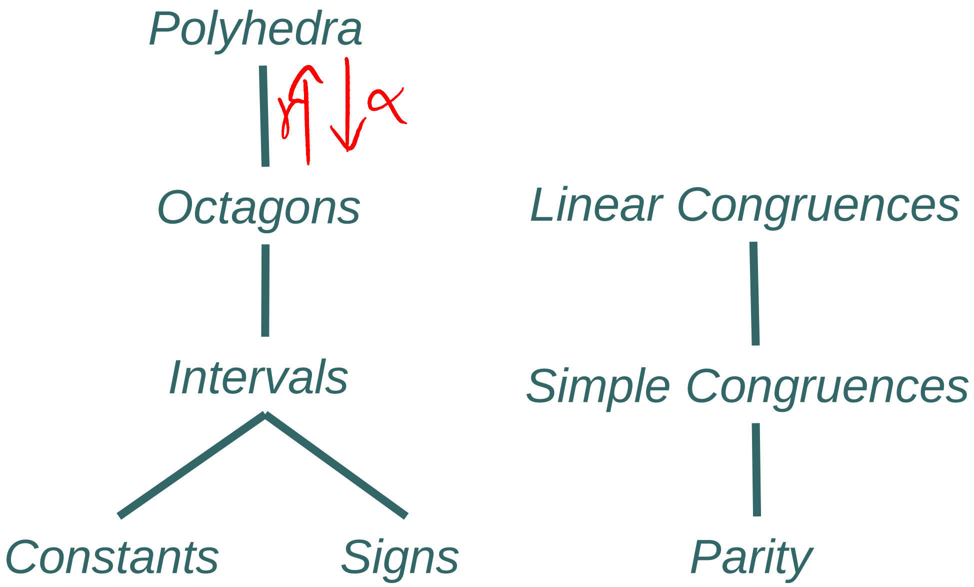
Which abstraction is the most precise?

Depends on questions you want to answer!





Partial Order of Abstractions



Polyhedra

Octagons

Intervals

Constants

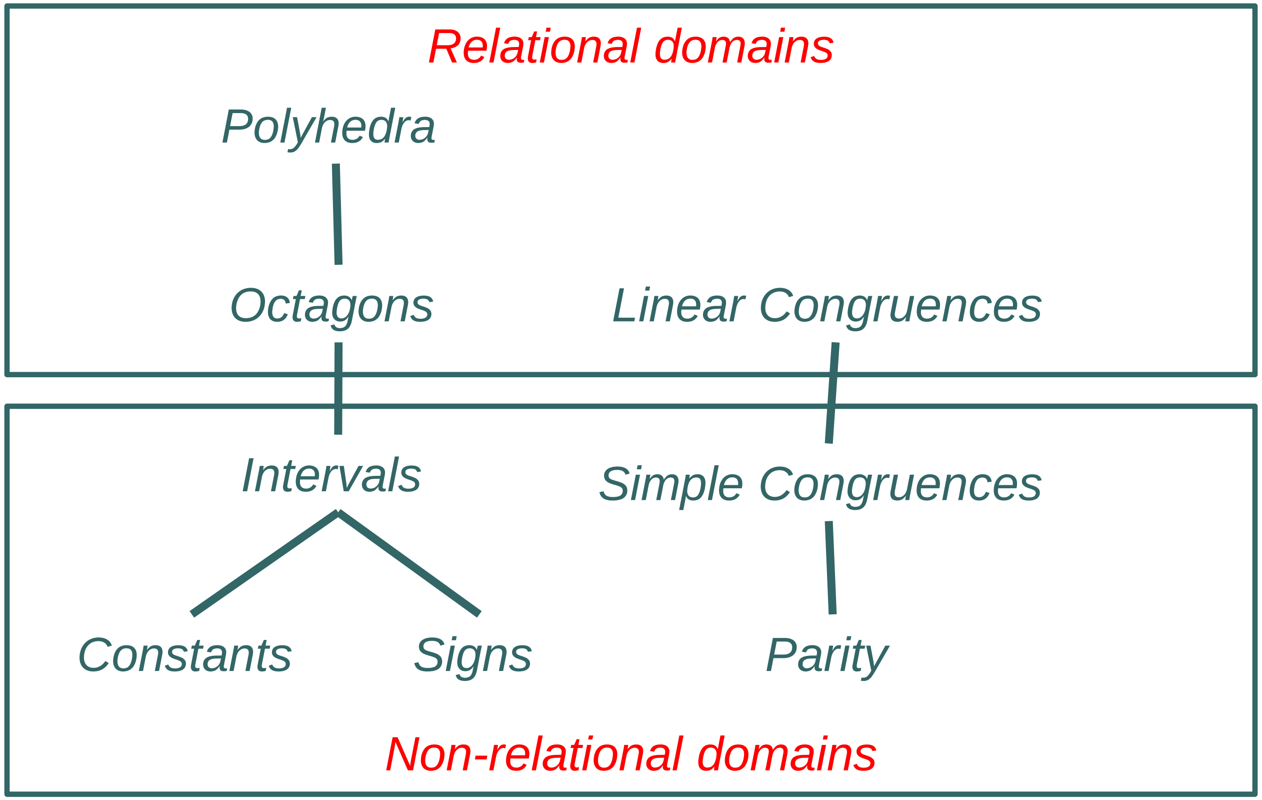
Signs

Linear Congruences

Simple Congruences

Parity

Partial Order of Abstractions



Characteristics of Non-relational Domains

- Non-relational/independent attribute abstraction:
  - Abstract each variable separately

$$(\mathcal{P}(\mathbb{Z}), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\textsc{Numerical}, \sqsubseteq)$$

  - Maintains no relations between variable values
- Can be lifted to an abstraction of valuations of multiple variables in the expected way:

$$(\mathcal{P}(\mathit{Vars} \to \mathbb{Z}), \subseteq) \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} (\mathit{Vars} \to \boxed{\mathcal{P}(\mathbb{Z})}, \leq) \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} (\mathit{Vars} \to \textsc{Numerical}, \sqsubseteq)$$

$$\alpha_2(f) := \lambda x \in \mathit{Vars}.\alpha(f(x)) \qquad \gamma_2(f^{\#}) := \lambda x \in \mathit{Vars}.\gamma(f^{\#}(x))$$

$y \to \emptyset$ $y \to \bot$

The Interval Domain

Abstracts sets of values by enclosing interval

$$\textsc{Interval} = \{[l, u] \mid l \leq u, l \in \mathbb{Z} \cup \underline{\{-\infty\}}, u \in \mathbb{Z} \cup \underline{\{\infty\}}\} \cup \{\bot\}$$

where $\leq$ is appropriately extended from $\mathbb{Z} \times \mathbb{Z}$ to $(\mathbb{Z} \cup \{-\infty\}) \times (\mathbb{Z} \cup \{\infty\})$

Intervals are ordered by inclusion:

$$\bot \sqsubseteq x \quad \forall x \in \textsc{Interval}$$

$$[l, u] \sqsubseteq [l', u'] \ \mathit{if}\ l' \leq l \wedge u \leq u'$$

$(\textsc{Interval}, \sqsubseteq)$ forms a complete lattice.

Concretization and Abstraction of Intervals

- Concretization:

$$\gamma(\bot) = \emptyset$$

$$\gamma([l, u]) = \{n \in \mathbb{Z} \mid l \leq n \leq u\}$$

- Abstraction:

$$\alpha(\emptyset) = \bot$$

$$\alpha(S) = [\inf S, \sup S]$$

min max

They form a Galois connection.

Interval Arithmetic

Calculating with Intervals:

$$
\begin{aligned}
[a,b] &\quad + \quad [c,d] &&= [a+c, b+d] \\
[a,b] &\quad - \quad [c,d] &&= [\underline{a-d,} b-c] \\
[a,b] &\quad * \quad [c,d] &&= [\min(ac,ad,bc,bd), \max(ac,ad,bc,bd)] \\
[a,b] &\quad / \quad [c,d] &&= \underline{[a,b]*[1/d,1/c], 0 \notin [c,d]}
\end{aligned}
$$

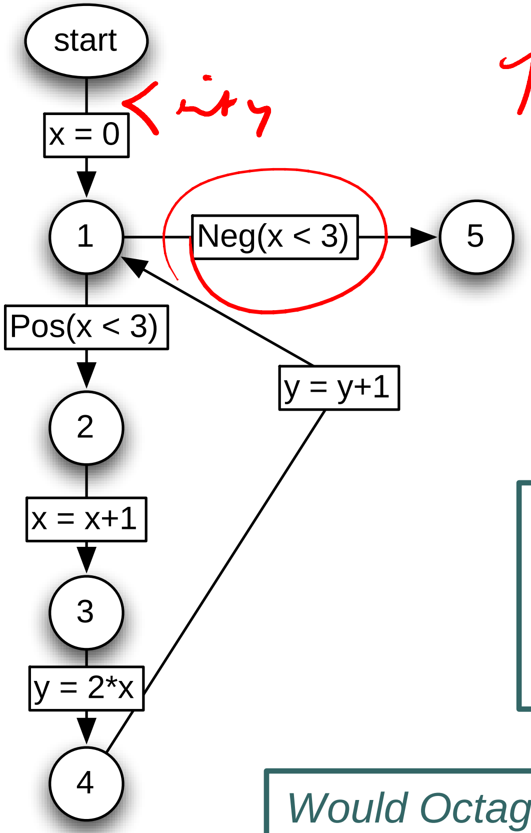
$$x/y = x \cdot \frac{1}{y}$$

Example: Interval Analysis

y = -x

y = x + 3

y = 2x + 7



< x+y

x → [0,3] y → [3,7]	x → [0,2] y → [3,5]	x → [0,1] y → [3,3]	x → [0,0] y → ~~top~~ ⊥
	x → [0,2] y → [3,5]	x → [0,1] y → [3,3]	x → [0,0] y → ~~top~~ ⊥
	x → [1,3] y → [3,5]	x → [1,2] y → [3,3]	x → [1,1] y → ~~top~~ ⊥
	x → [1,3] y → [2,6]	x → [1,2] y → [2,4]	x → [1,1] y → [2,2]

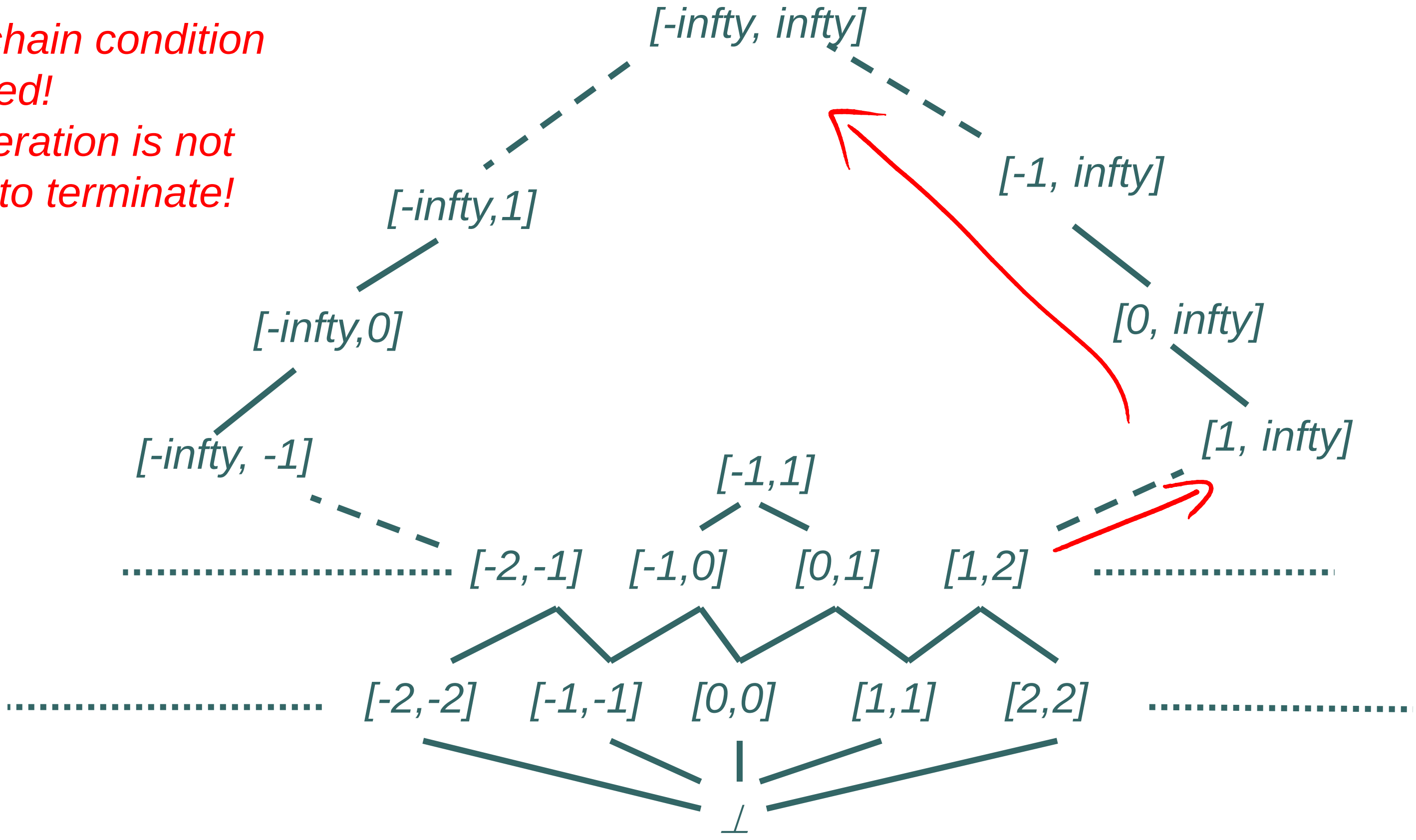
x → [3,3]
y → [3,7]

Imprecise due to non-relational analysis

Would Octagons determine that y must be 7 at program point 5?

Intervals, Hasse diagram

Ascending chain condition is not satisfied!
→ Kleene iteration is not guaranteed to terminate!



[-infty, infty]

[-1, infty]

[-infty,1]

[0, infty]

[-infty,0]

[1, infty]

[-infty, -1]

[-1,1]

[-2,-1] *[-1,0]* *[0,1]* *[1,2]*

[-2,-2] *[-1,-1]* *[0,0]* *[1,1]* *[2,2]*

⊥

Example: Interval Analysis

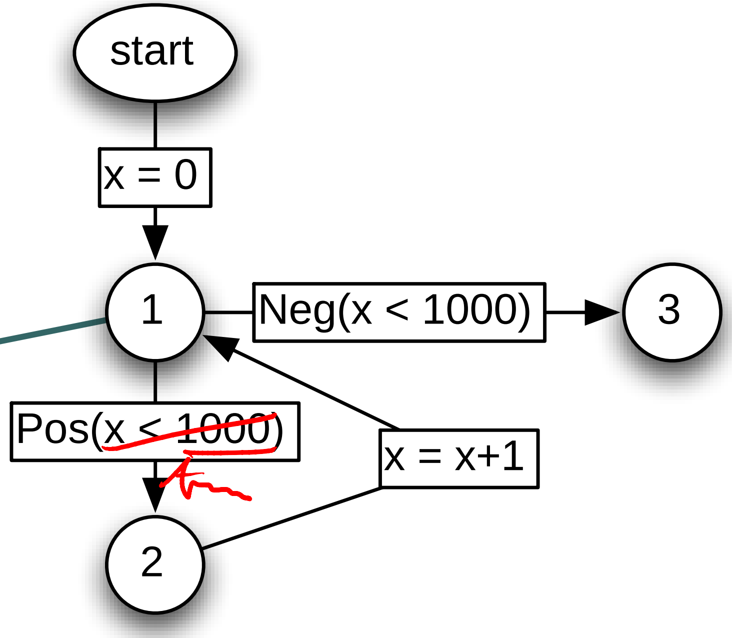
$x \mapsto \perp$

$x \mapsto [0,0]$

$x \mapsto [0,1]$

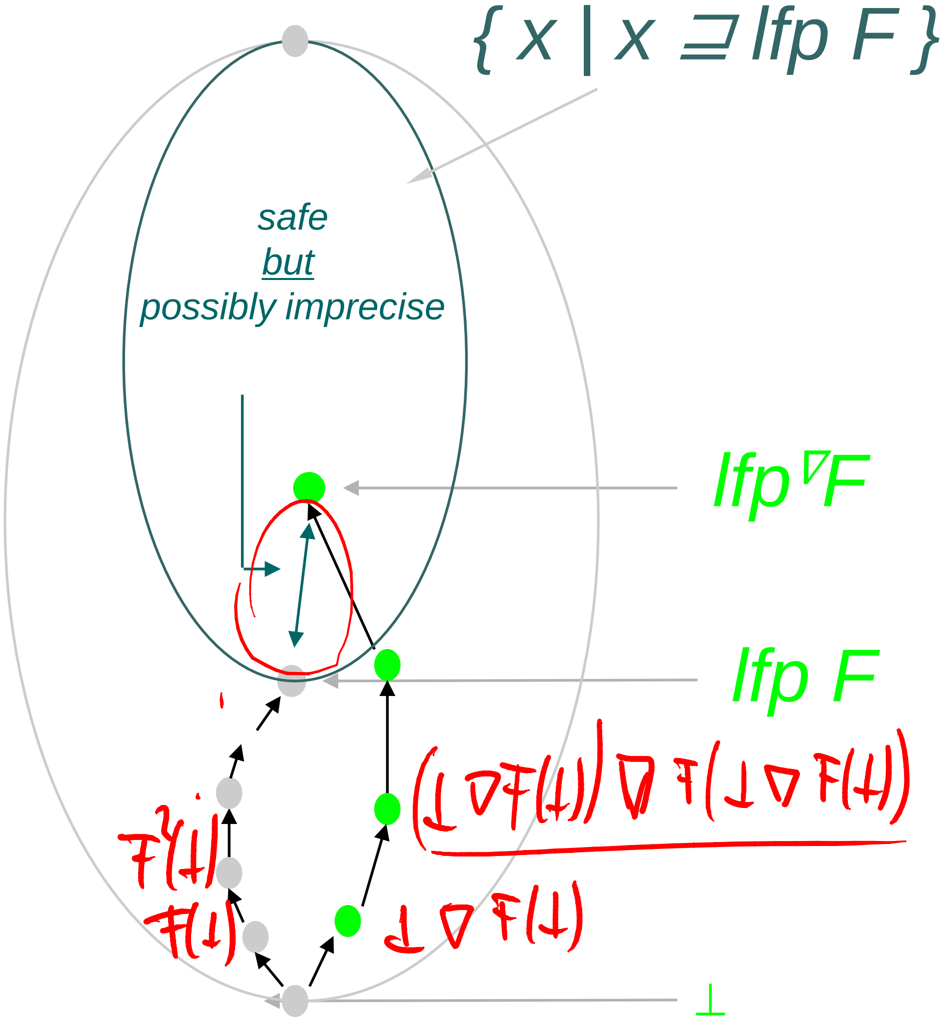
$\dots$

1000 iterations later → $x \mapsto [0,1000]$



Solution: Widening
“Enforce Ascending Chain Condition”

- Widening enforces the ascending chain condition during analysis.
- Accelerates termination by moving up the lattice more quickly.
- May yield imprecise results…



Widening: Formal Requirement

A widening ∇ is an operator $\nabla : D \times D \rightarrow D$ such that

1. Safety: $x \sqsubseteq (x \nabla y)$ and $y \sqsubseteq (x \nabla y)$
2. Termination:

   forall ascending chains $x_0 \sqsubseteq x_1 \sqsubseteq \ldots$ the chain

   $$\begin{aligned} y_0 &= x_0 \\ y_{i+1} &= y_i \nabla x_{i+1} \end{aligned}$$

   is finite.

$$x_0 \,,\, x_0 \nabla x_1 \,,\, (x_0 \nabla x_1) \nabla x_2 \,,\, \ldots$$

Widening Operator for Intervals

Simplest solution:

$$\bot \nabla x = \cancel{x \nabla \bot} = x$$

$$[l, u] \nabla [l', u'] = \left[\begin{cases} l & : l' \geq l \\ -\infty & : l' < l \end{cases}, \begin{cases} u & : u' \leq u \\ \infty & : u' > u \end{cases} \right]$$

Example:

$$[3, 5] \nabla [\underline{2}, 5] = [\boxed{-\infty,} 5]$$

$$[3, 5] \nabla [4, 5] = [3, 5]$$

$$[3, 5] \nabla [4, 6] = [3, \infty]$$

$$[3, 5] \nabla [2, \boxed{6}] = [-\infty, \boxed{\infty}]$$

Example Revisited: Interval Analysis with Simple Widening

Standard Kleene Iteration:

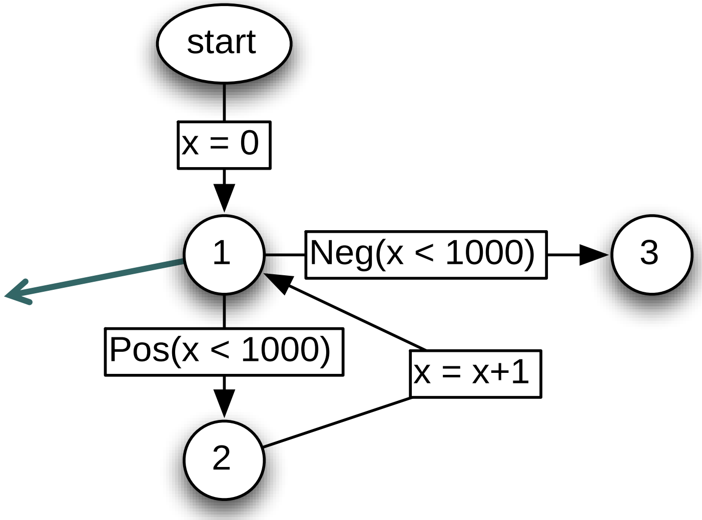
$$\bot \leq F(\bot) \leq F^2(\bot) \leq F^3(\bot) \leq \dots$$

Kleene Iteration with Widening: $F_\nabla(x) := x \nabla F(x)$

$$\bot \leq F_\nabla(\bot) \leq F_\nabla^2(\bot) \leq F_\nabla^3(\bot) \leq \dots$$

$x \mapsto [0, 0]$

$x \mapsto [0, \infty]$



Do we need to apply widening at all program points?

→ Quick termination but imprecise result!

More Sophisticated Widening for Intervals

Define set of jump points (barriers) based on constants appearing in program, e.g.:

$$\mathcal{J} = \{-\infty, 0, 1, 1000, \infty\}$$

Intuition: “Don’t jump to –infty, +infty immediately but only to next jump point.”

$$[l,u] \nabla [l',u'] = \left[\begin{cases} l & : l' \geq l \\ \max\{x \in \mathcal{J} \mid x \leq l'\} & : l' < l \end{cases}, \right.$$

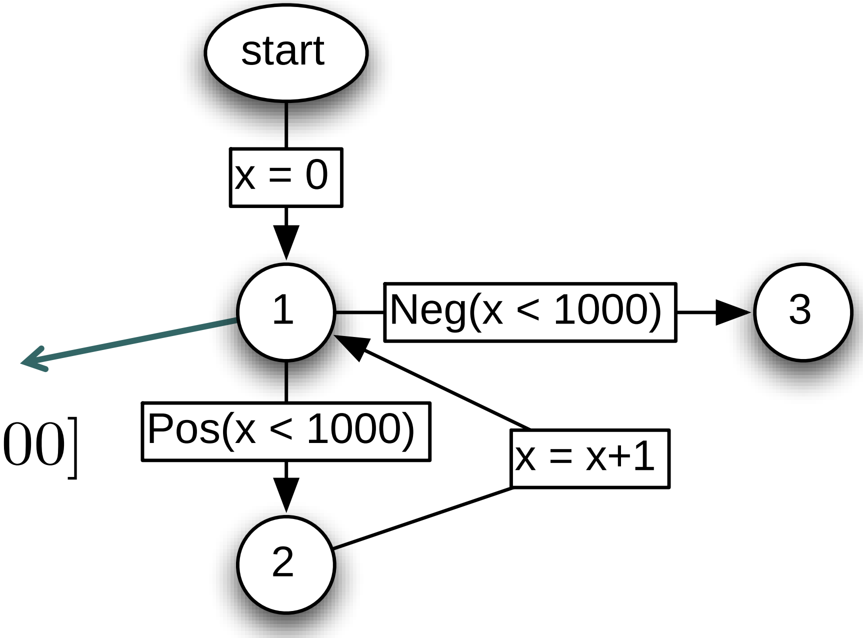
$$\left. \begin{cases} u & : u' \leq u \\ \min\{x \in \mathcal{J} \mid x \geq u'\} & : u' > u \end{cases} \right]$$

Example Revisited:
Interval Analysis with Sophisticated Widening

$$x \mapsto [0, 0]$$

$$x \mapsto [0, 1]$$

$$x \mapsto [0, 1000]$$



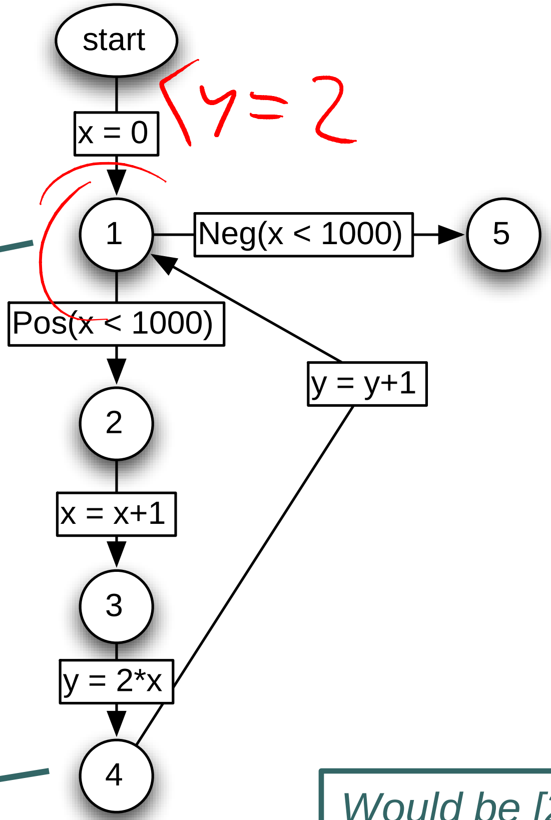
→ More precise, potentially terminates more slowly.

Another Example:
Interval Analysis with Sophisticated Widening

$x \mapsto [0, 0]$

$x \mapsto [0, 1]$

$x \mapsto [0, 1000]$



$y \mapsto [2, 2]$

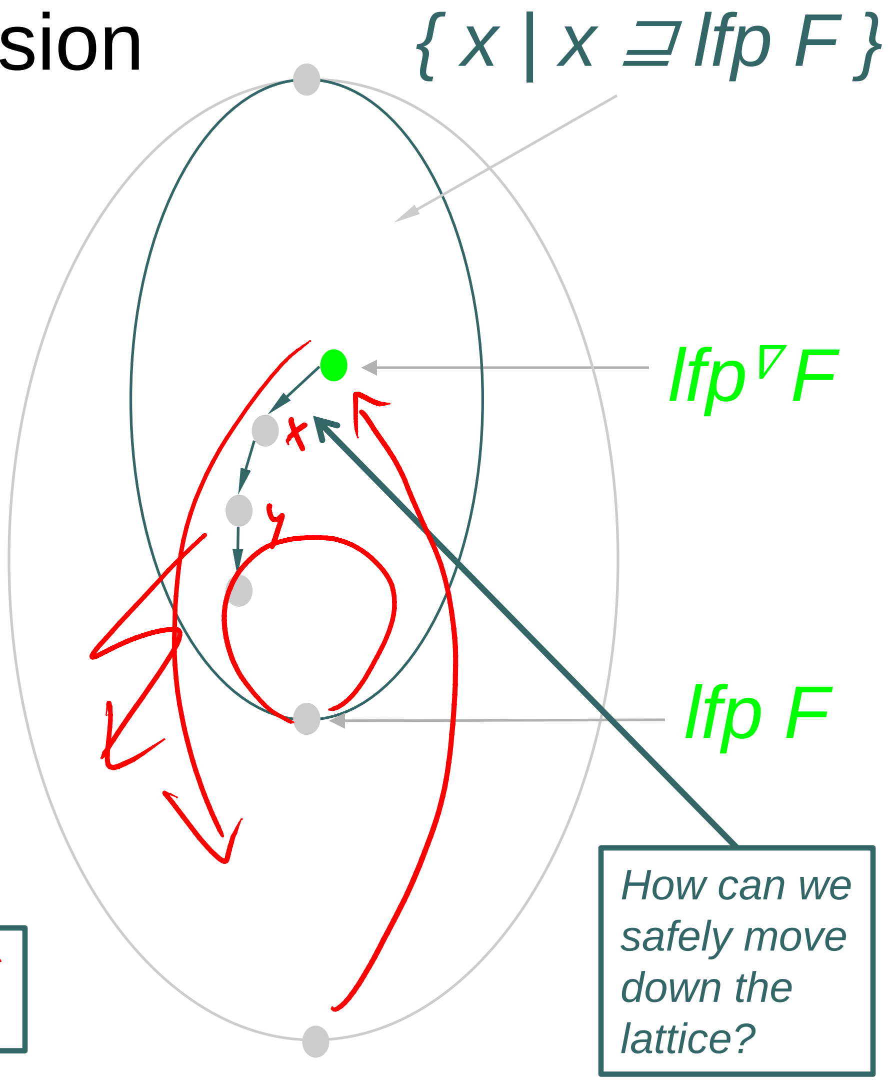
$y \mapsto [2, 1000]$

$y \mapsto [2, \infty]$

Would be [2, 2000] in least fixed point, but 2000 does not appear in the program…

Narrowing: Recovering Precision

- Widening may yield imprecise results by overshooting the least fixed point.
- Narrowing is used to approach the least fixed point from above.



How can we safely move down the lattice?

Possible problem: infinite descending chains
Is it really a problem?

Narrowing: Recovering Precision

Widening terminates at a point $x \sqsupseteq \text{lfp } F$.

We can iterate:

$$x_0 = x$$
$$x_{i+1} = F(x_i) \left(\sqcap x_i\right) = F(x_i)$$

Safety:

By monotonicity we know $F(x) \sqsupseteq F(\text{lfp } F) = \text{lfp } F$.
By induction we can easily show that $x_i \sqsupseteq \text{lfp } F$ for all i.

Termination:

Depends on existence of infinite descending chains.

Narrowing: Formal Requirement

A narrowing Δ is an operator $\Delta : D \times D \rightarrow D$ such that

1. **Safety:** $I \sqsubseteq x$ and $I \sqsubseteq y \;\;\rightarrow\;\; \underline{I \sqsubseteq (x \,\Delta\, y)} \boxed{\sqsubseteq x}$
2. **Termination:**

   for all descending chains $x_0 \sqsupseteq x_1 \sqsupseteq \ldots$ the chain

   $$y_0 = x_0$$
   $$y_{i+1} = y_i \,\Delta\, x_{i+1}$$

   is finite.

Is $\sqcap$ (“meet”) a narrowing operator on intervals?

Narrowing Operator for Intervals

Simplest solution:

$$x \Delta \bot = \bot$$

$$[l,u] \Delta [l',u'] = \left[\begin{cases} l' & : l = -\infty \\ l & : else \end{cases}, \begin{cases} u' & : u = \infty \\ u & : else \end{cases} \right]$$

Example:

$$\begin{aligned} \boxed{[2,5]} \Delta [4,5] &= [2,5] \\ [-\infty,5] \Delta [4,5] &= [4,5] \\ [-\infty,\infty] \Delta [4,6] &= [4,6] \\ [2,\infty] \Delta [3,5] &= [2,5] \end{aligned}$$

Another Example Revisited:
Interval Analysis with Widening and Narrowing

Result after Widening:

$x \mapsto [0, 0]$

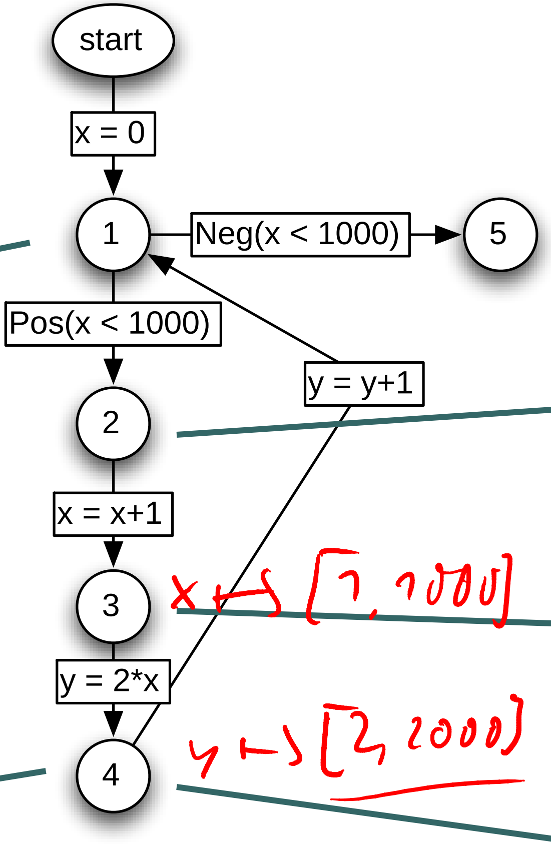
$x \mapsto [0, 1]$

$x \mapsto [0, 1000]$

$y \mapsto [2, 2]$

$y \mapsto [2, 1000]$

$y \mapsto [2, \infty]$



x ↦ [1, 1000]

y ↦ [2, 2000]

Result after Narrowing:

$x \mapsto [1000, 1000]$

$y \mapsto [3, 2001]$

$x \mapsto [0, 999]$

$x \mapsto [1, 1000]$

$y \mapsto [2, 2000]$

→ *Precisely the least fixed point!*

Some Applications of Numerical Domains

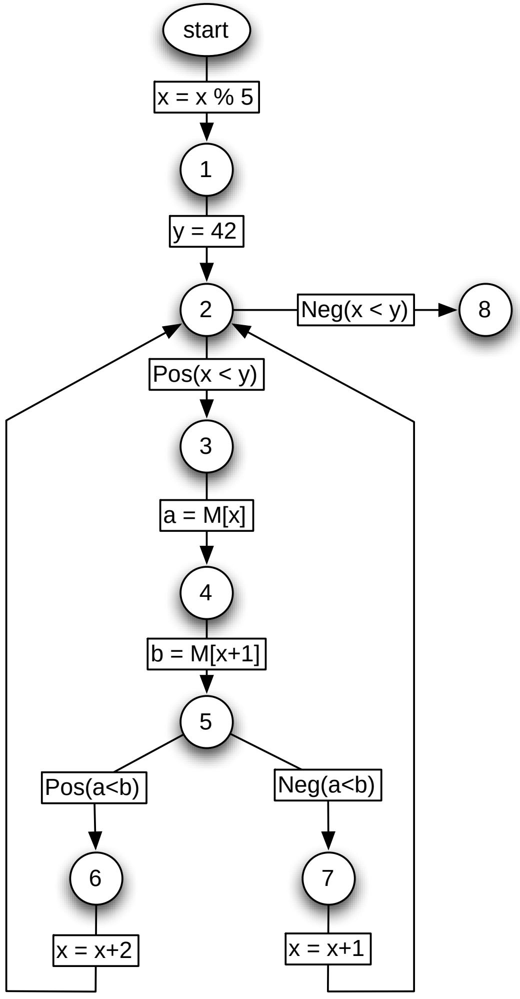
Immediate applications:

- To rule out runtime errors, such as division by zero, buffer overflows, exceeding upper or lower bounds of data types

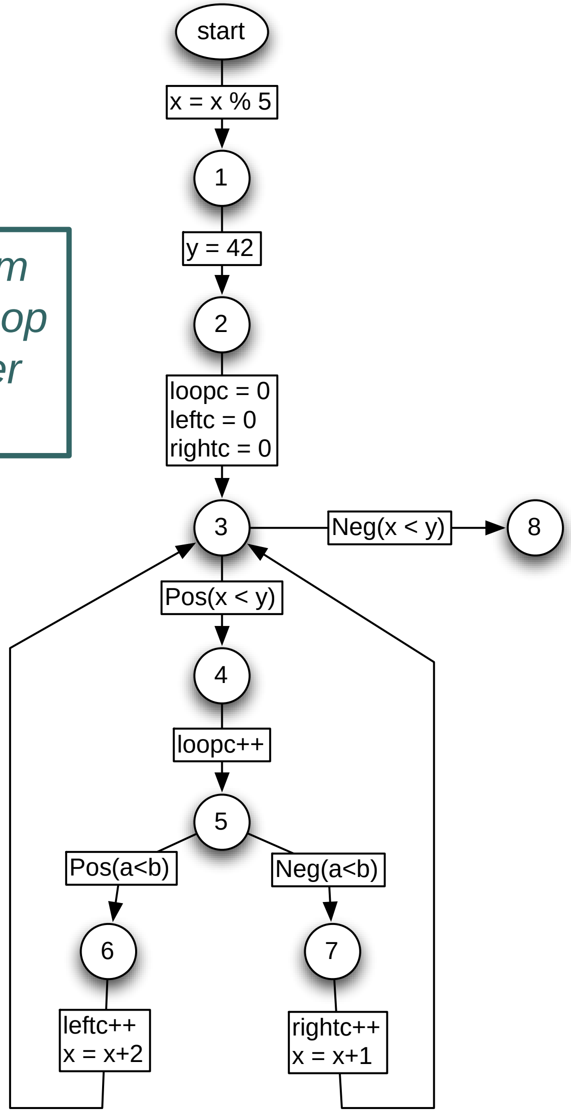
Within other analyses:

- Cache Analysis
- Loop Bound Analysis

Reduction:
Loop Bound Analysis to Value Analysis



Instrument program with counters of loop iterations and other interesting events



Summary

- Interval Analysis:

  A non-relational value analysis
- Widenings for termination in the presence of Infinite Ascending Chains
- Narrowings to recover precision
- Basic Approach to Loop Bound Analysis based on Value Analysis

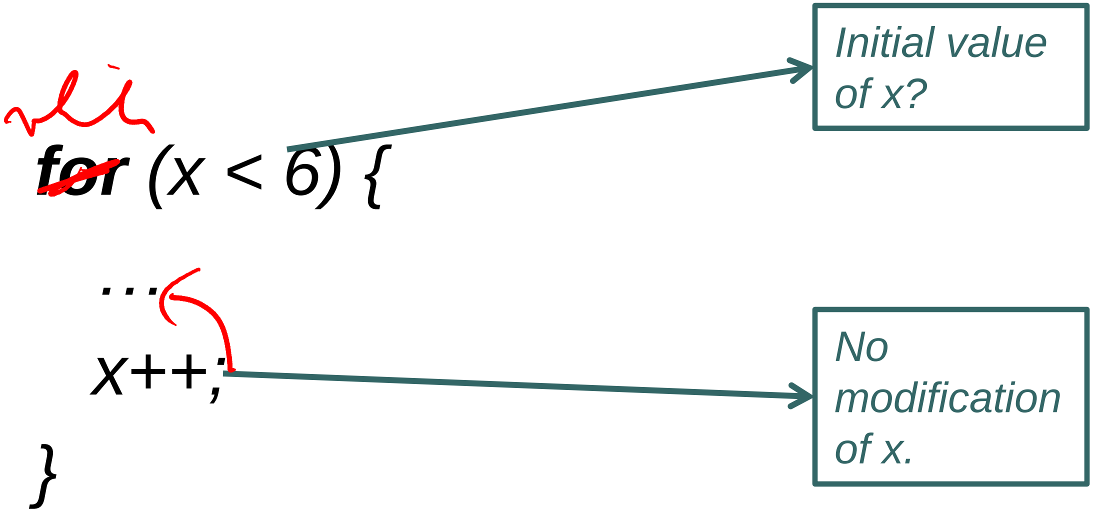
State of the Art in Loop Bound Analysis

Multiple approaches of varying sophistication

- Pattern-based approach
- Slicing + Value Analysis + Invariant Analysis
- Reduction to Value Analysis

Loop Bound Analysis: Pattern-based Approach

Identify common loop patterns; derive loop bounds for pattern once manually



while

~~for~~ (x < 6) {
  …
  x++;
}

Initial value of x?

No modification of x.

→ Loop bound: 6-minimal value of x

Slicing + Value Analysis + Invariant Analysis [Ermedahl et al., WCET 2007]

Combination of multiple analyses:

1. Slicing: eliminate code that is irrelevant for loop termination
2. Value analysis: determine possible values of all variables in slice
3. Invariant analysis: determine variables that do not change during loop execution
4. Loop bound = set of possible valuations of non-invariant variables

***Program slicing** is the computation of the set of programs statements, the program slice, that may affect the values at some point of interest, referred to as a **slicing criterion**.*

Slicing + Value Analysis + Invariant Analysis [Ermedahl et al., WCET 2007]

Step 1: Slicing with slicing criterion (i <= INPUT)

```
int OUTPUT = 0;
int i = 1;
while (i <= INPUT) {
        OUTPUT += ~~2;~~ f(i)
        i += 2;
}
```

```
int i = 1;
while (i <= INPUT) {
        i += 2;
}
```

Slicing + Value Analysis + Invariant Analysis [Ermedahl et al., WCET 2007]

Step 2: Value Analysis

Observation:

If the loop terminates, the program can only be in any particular state once.

→ Determine number of states the program can be in at the loop header.

```
int i = 1;
while (i <= INPUT) {
        i += 2;
}
```

Value Analysis:

INPUT in [10, 20] (assumption)

i in [1, 20], i % 2 = 1

→ *11 * 10 states*

→ *Loop bound 110!*

Slicing + Value Analysis + Invariant Analysis [Ermedahl et al., WCET 2007]

Step 3: Invariant Analysis

Observation:

Value of INPUT is not completely known, but INPUT does not change during loop.

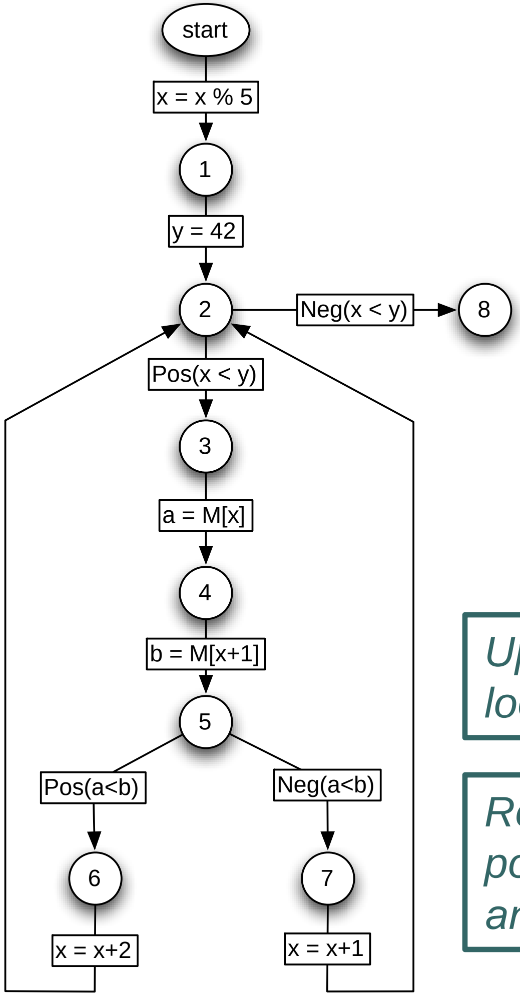
→ Determine variables that are invariant during loop.

```
int i = 1;
while (i <= INPUT) {
        i += 2;
}
```

Value Analysis:
INPUT in [10, 20] (assumption)
i in [1, 20] , i % 2 = 1

→ *INPUT is invariant!*
→ *Loop bound 10!*

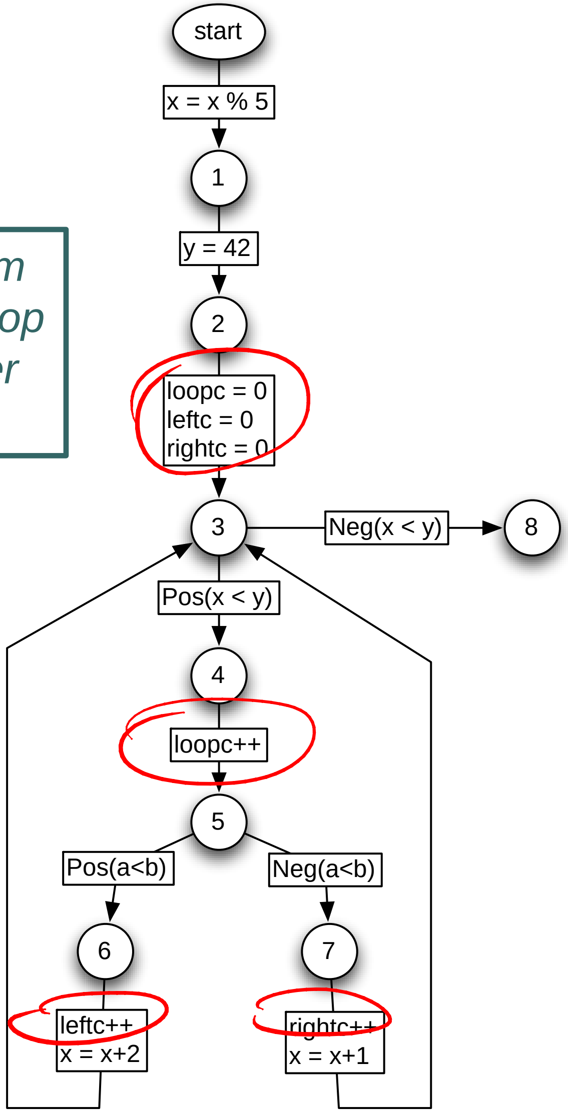
Reduction:
Loop Bound Analysis to Value Analysis



Instrument program with counters of loop iterations and other interesting events

Upper bound for loopc is loop bound!

Requires very powerful relational analysis…


start
x = x % 5
1
y = 42
2
loopc = 0
leftc = 0
rightc = 0
3
Neg(x < y)
8
Pos(x < y)
4
loopc++
5
Pos(a<b)
Neg(a<b)
6
7
leftc++
x = x+2
rightc++
x = x+1