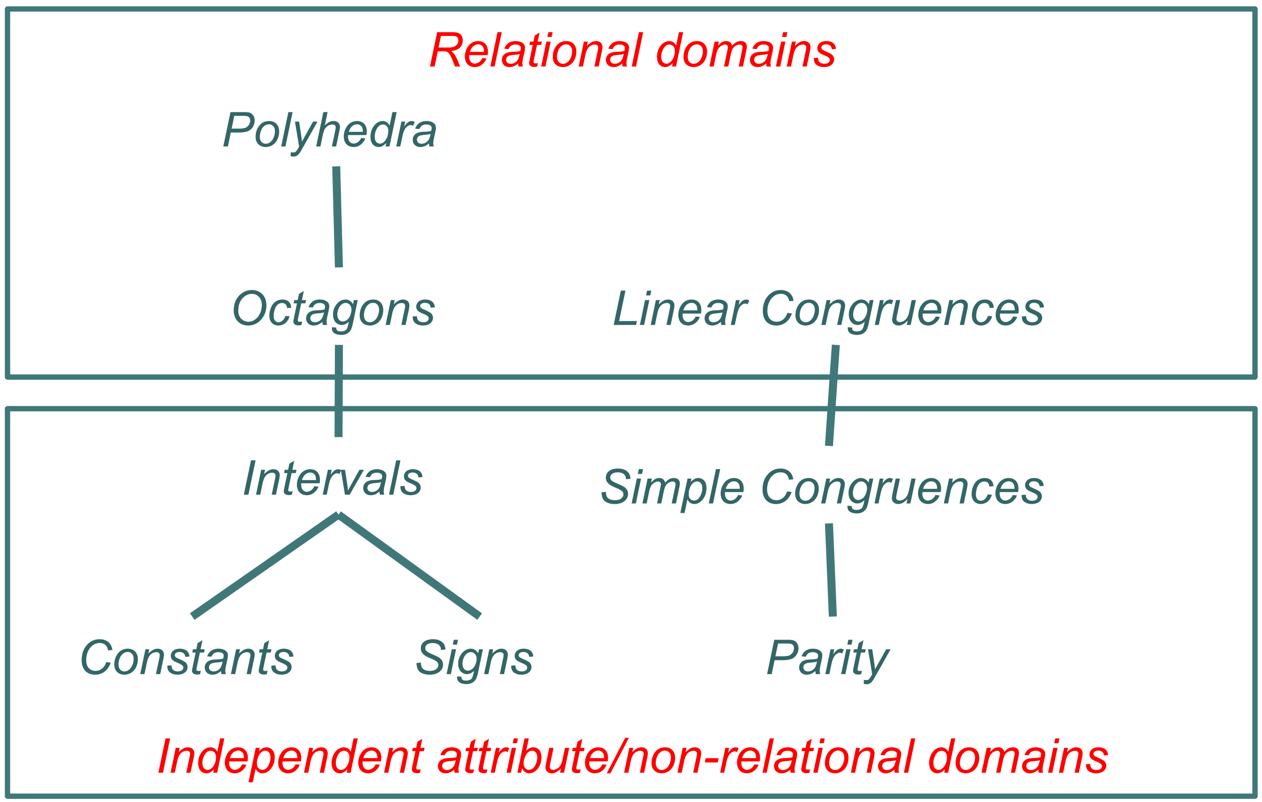
Design and Analysis of Real-Time Systems

Foundations of Abstract Interpretation and Numerical Abstractions

Jan Reineke

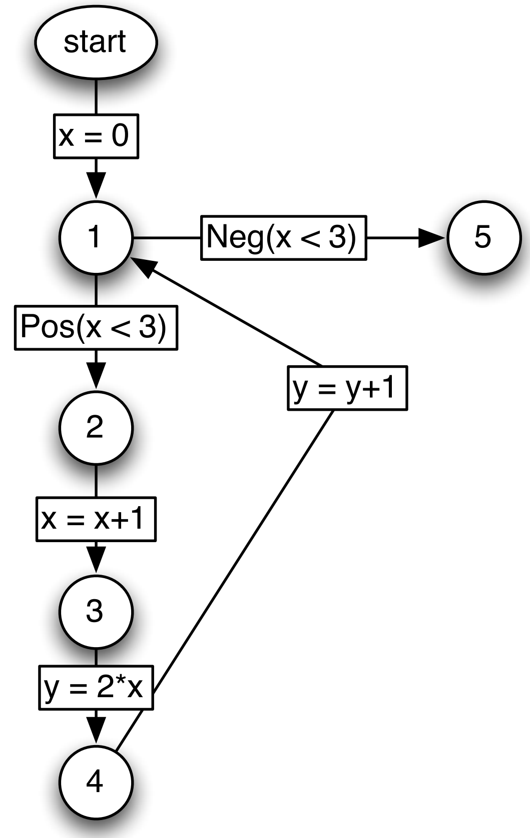
Advanced Lecture, Summer 2013

Partial Order of Abstractions

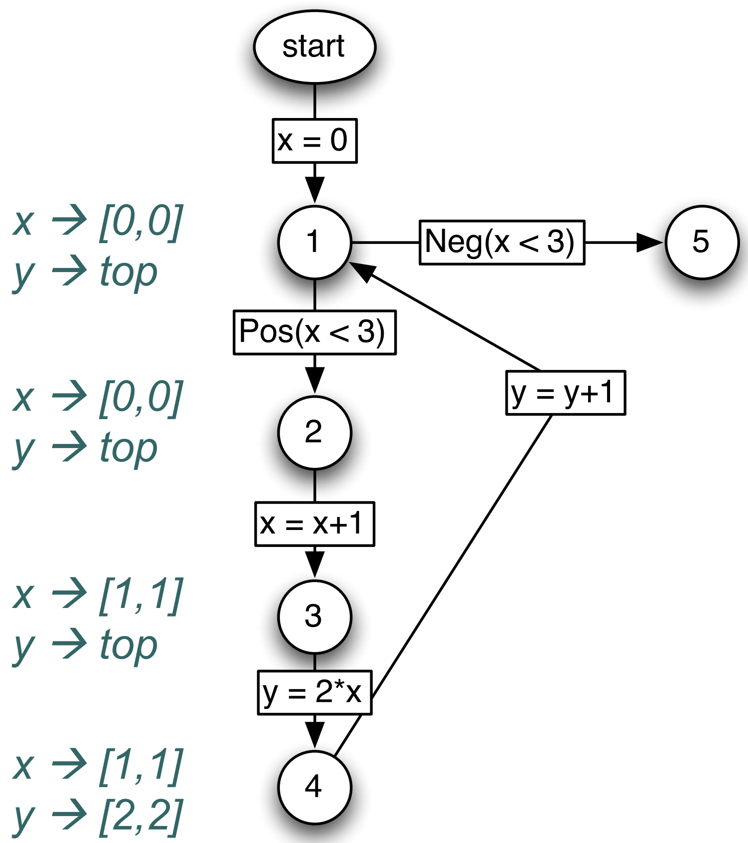


More domains are described at: http://bugseng.com/products/ppl/abstractions

Example: Interval Analysis

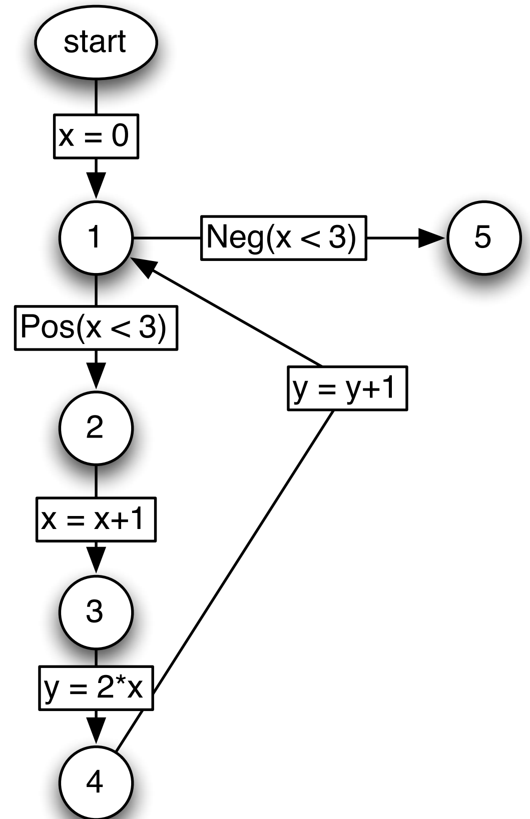


Example: Interval Analysis



Example: Interval Analysis

x → [0,1] *y → [3,3]*	*x → [0,0]* *y → top*
x → [0,1] *y → [3,3]*	*x → [0,0]* *y → top*
x → [1,2] *y → [3,3]*	*x → [1,1]* *y → top*
x → [1,2] *y → [2,4]*	*x → [1,1]* *y → [2,2]*



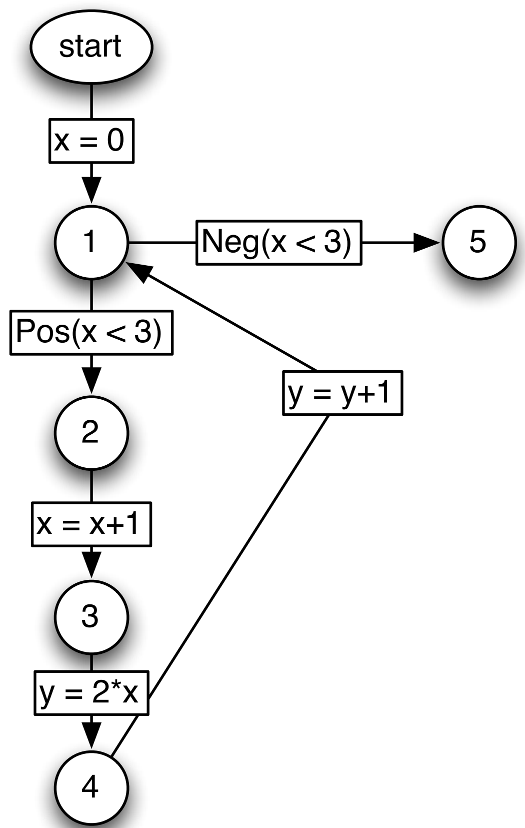
Example: Interval Analysis

x → [0,2] x → [0,1] x → [0,0]
y → [3,5] y → [3,3] y → top

x → [0,2] x → [0,1] x → [0,0]
y → [3,5] y → [3,3] y → top

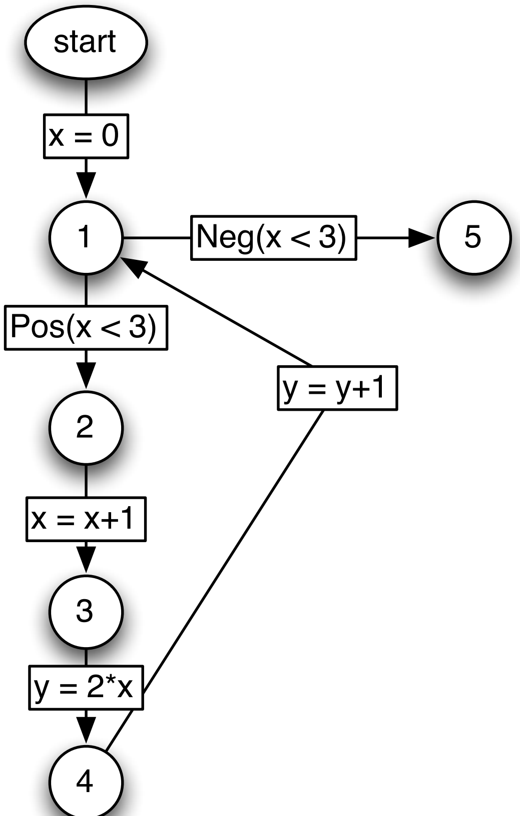
x → [1,3] x → [1,2] x → [1,1]
y → [3,5] y → [3,3] y → top

x → [1,3] x → [1,2] x → [1,1]
y → [2,6] y → [2,4] y → [2,2]



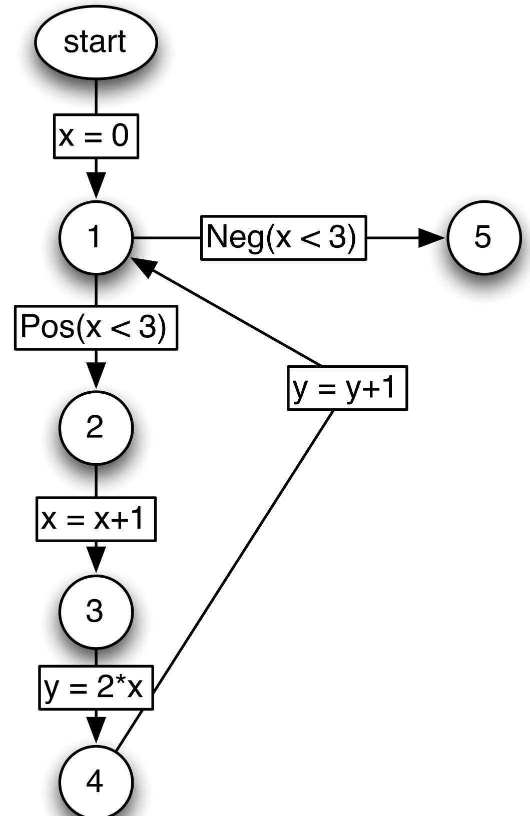
Example: Interval Analysis

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]



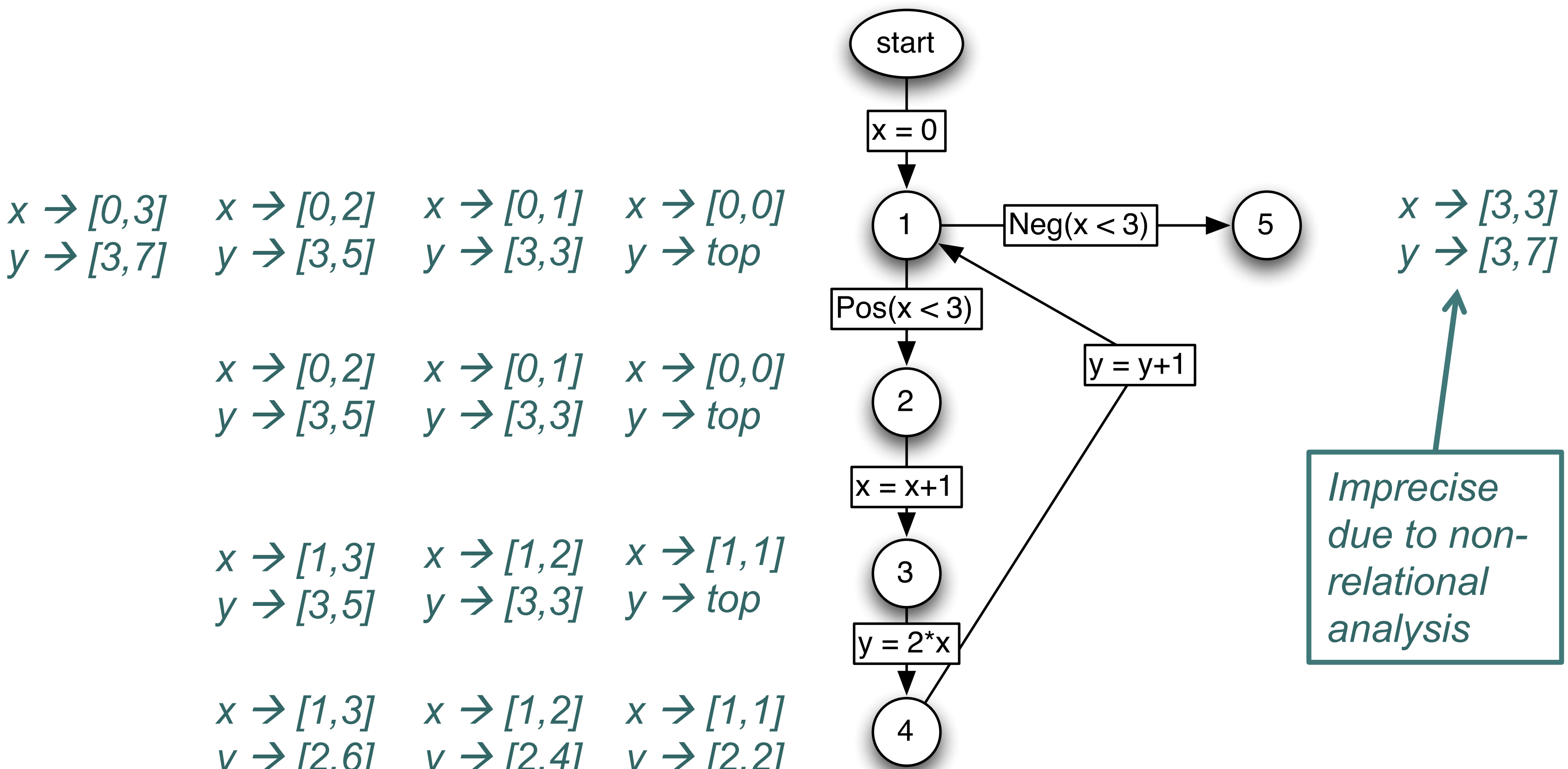
Example: Interval Analysis

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]



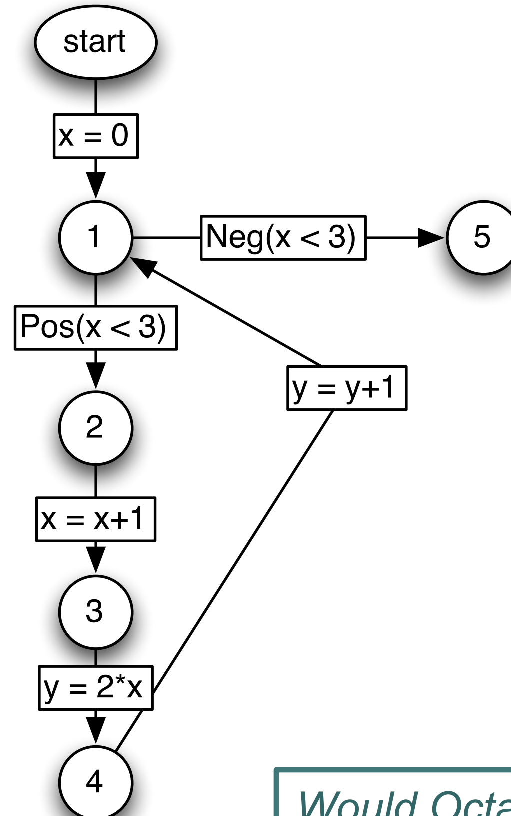
x → [3,3]
y → [3,7]

Example: Interval Analysis



Example: Interval Analysis

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]



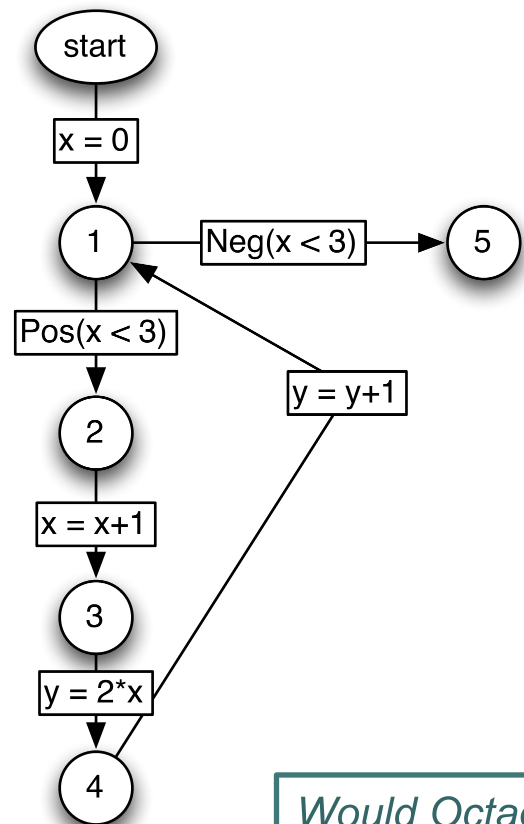
x → [3,3]
y → [3,7]

Imprecise due to non-relational analysis

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

Example: Interval Analysis

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]

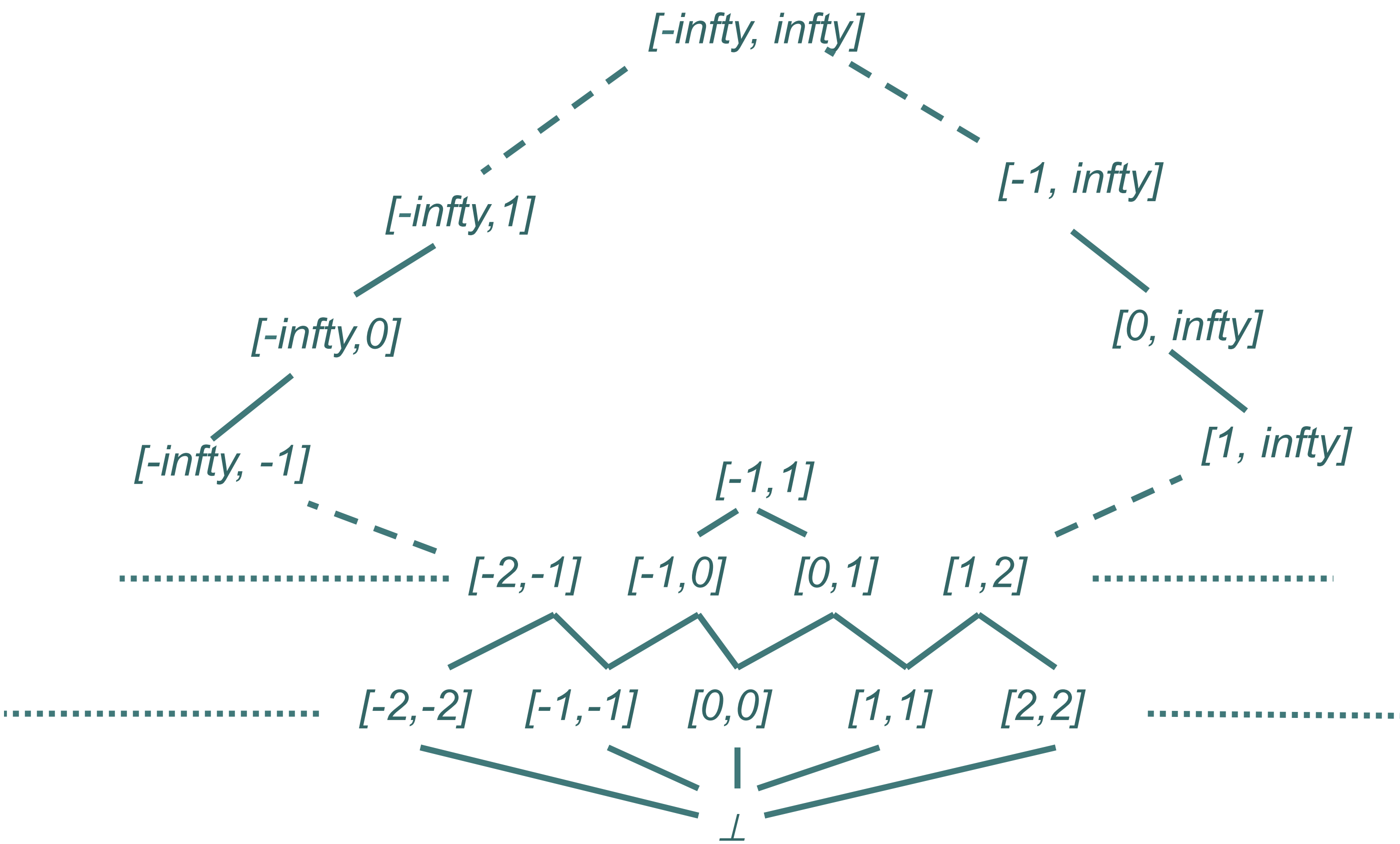


x → [3,3]
y → [3,7]

Imprecise due to non-relational analysis

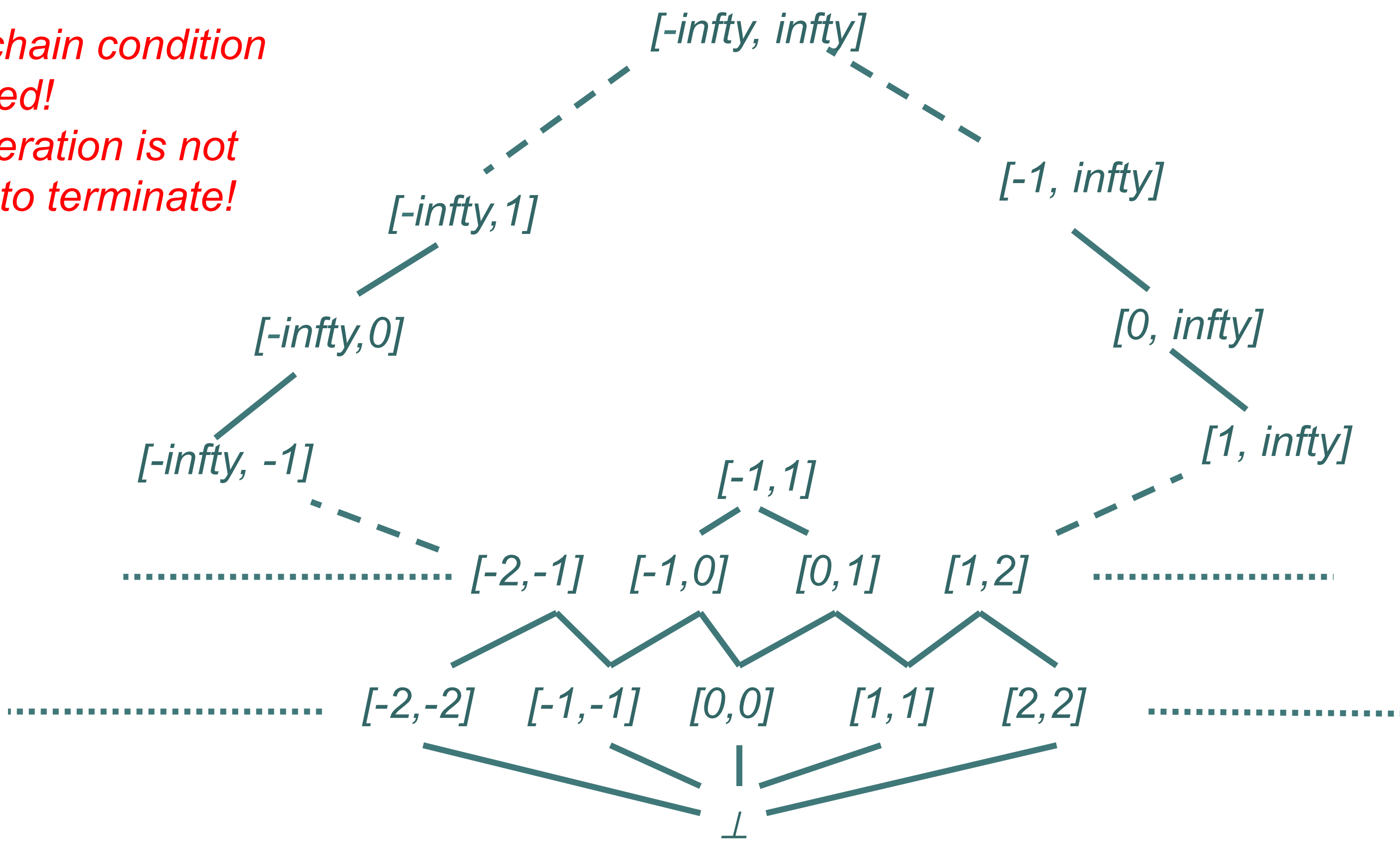
Would Octagons determine that y must be 7 at program point 5?

Intervals, Hasse diagram



Intervals, Hasse diagram

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



Example: Interval Analysis

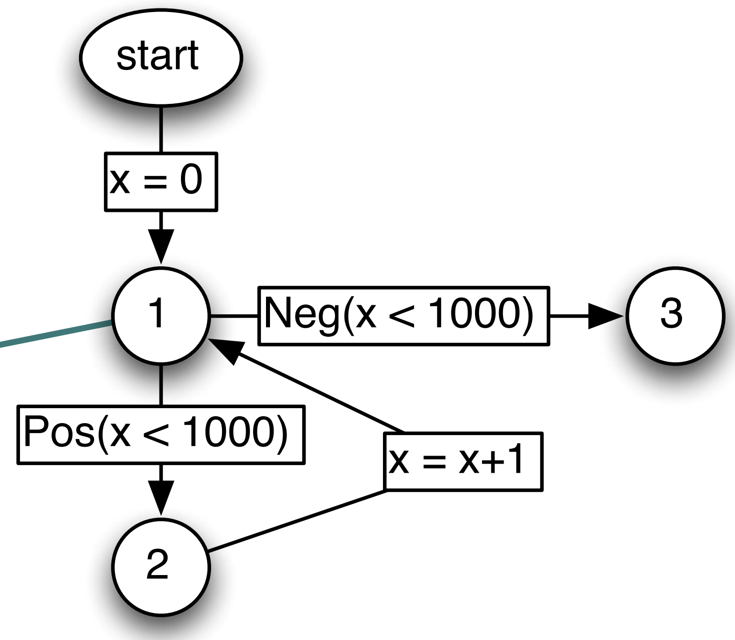
$$x \mapsto \bot$$

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

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

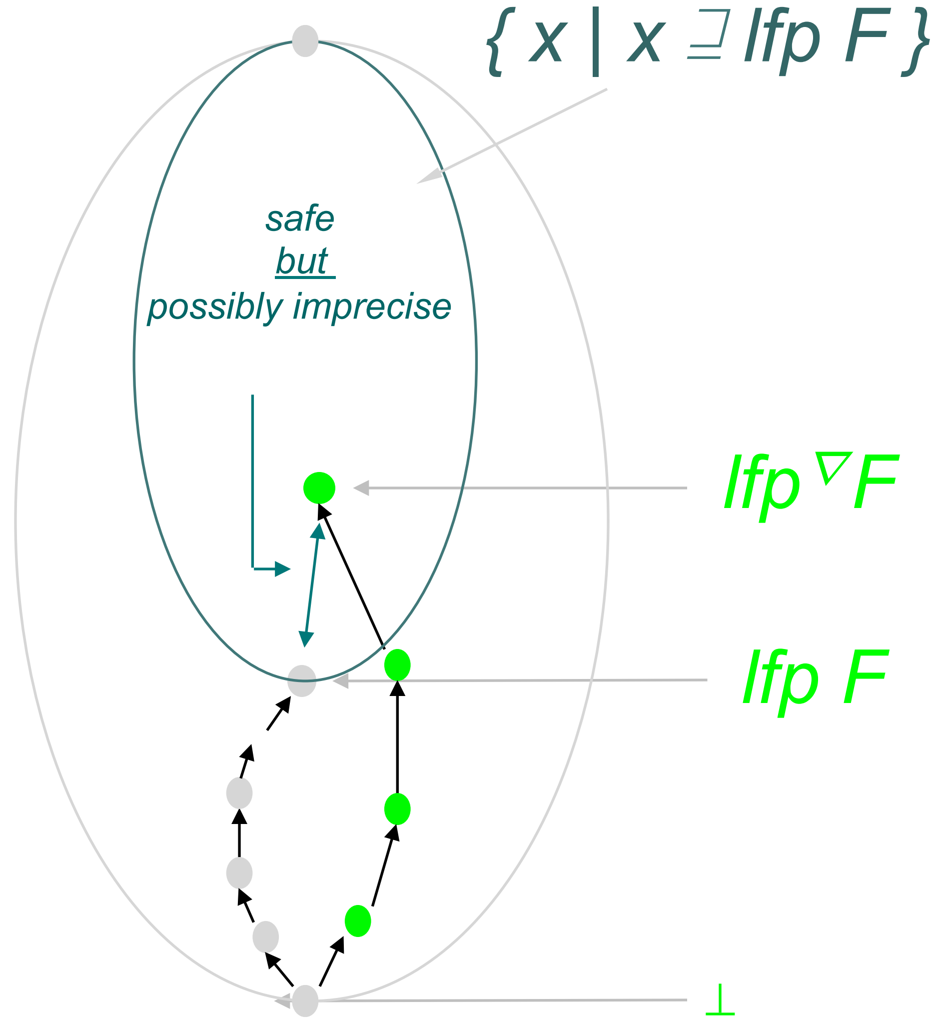
$$\dots$$

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:

   for all ascending chains $x_0 \sqsubseteq x_1 \sqsubseteq \ldots$ the chain

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

   is finite.

Widening Operator for Intervals

Simplest solution:

$$\bot \nabla x = x \nabla \bot = x$$

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

Example:

$$[3, 5] \nabla [2, 5] = [-\infty, 5]$$

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

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

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

Example Revisited: Interval Analysis with Simple Widening

Standard Kleene Iteration:

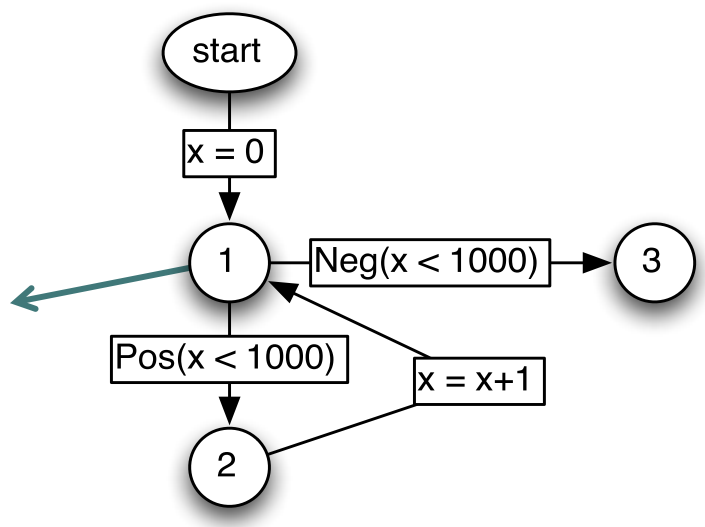
$$\bot \leq F(\bot) \leq F^2(\bot) \leq F^3(\bot) \leq \dots$$

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

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

$x \mapsto [0, 0]$

$x \mapsto [0, \infty]$



Example Revisited: Interval Analysis with Simple Widening

Standard Kleene Iteration:

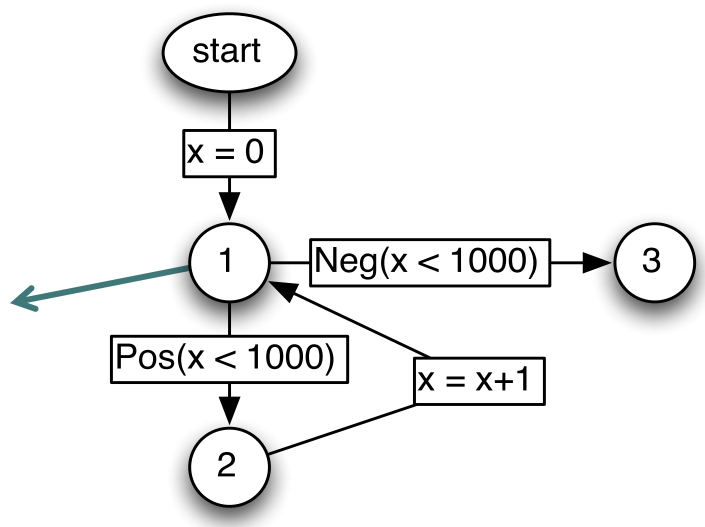
$$\bot \leq F(\bot) \leq F^2(\bot) \leq F^3(\bot) \leq \dots$$

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

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

$x \mapsto [0, 0]$

$x \mapsto [0, \infty]$



→ 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."

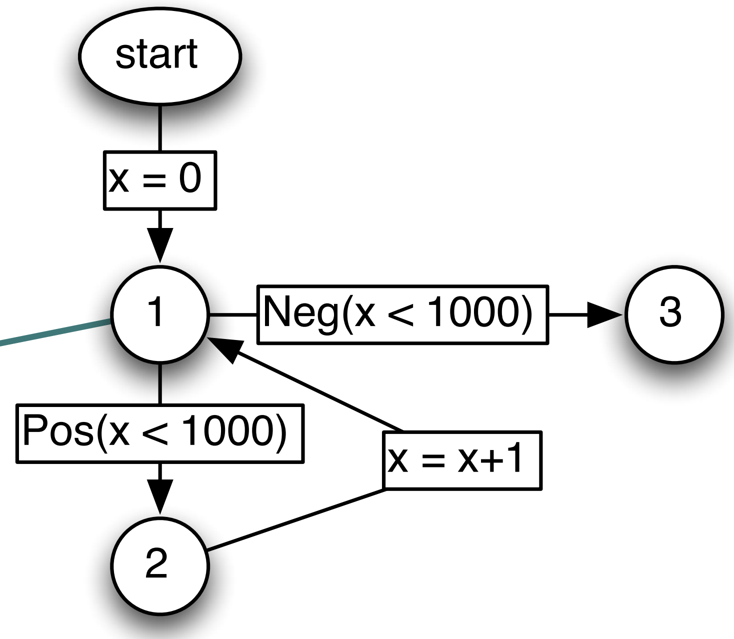
$$[l, u] \nabla [l', u'] = \left[\begin{cases} l & : l' \geq l \\ \max\{x \in \mathcal{J} \mid x \leq l'\} & : l' < l \end{cases}, \begin{cases} u & : u' \leq u \\ \min\{x \in \mathcal{J} \mid x \geq u'\} & : u' > u \end{cases} \right]$$

Example Revisited: Interval Analysis with Sophisticated Widening

$x \mapsto [0, 0]$

$x \mapsto [0, 1]$

$x \mapsto [0, 1000]$

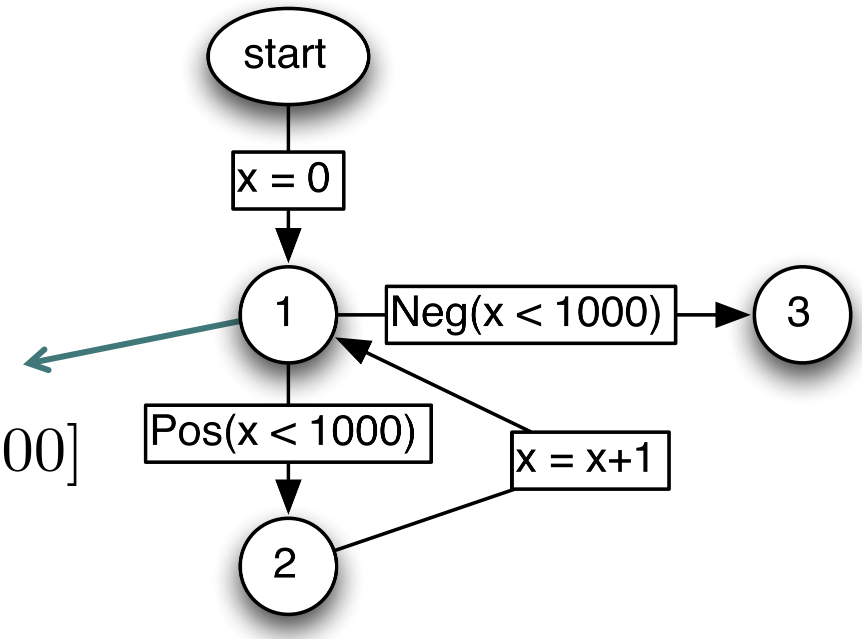


Example Revisited: Interval Analysis with Sophisticated Widening

$x \mapsto [0, 0]$

$x \mapsto [0, 1]$

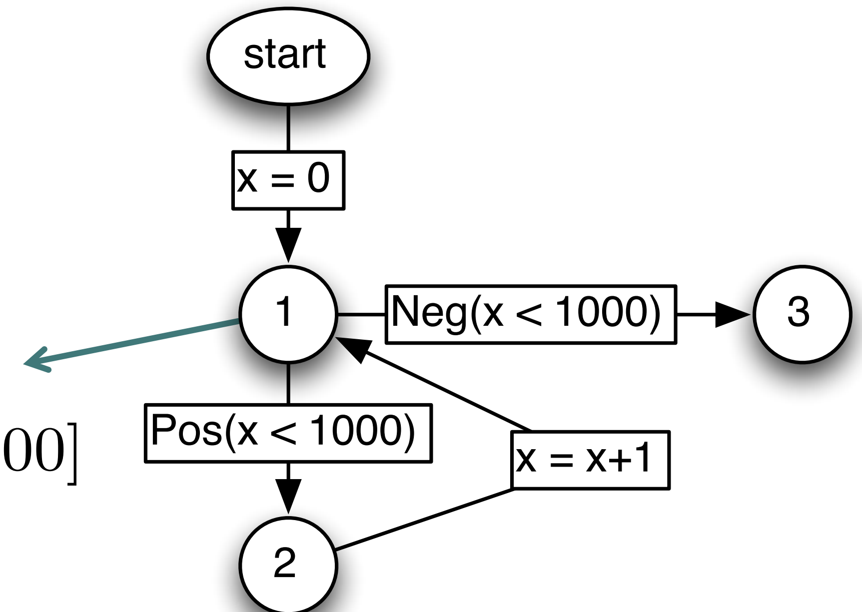
$x \mapsto [0, 1000]$



→ *More precise, potentially terminates more slowly.*

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.*

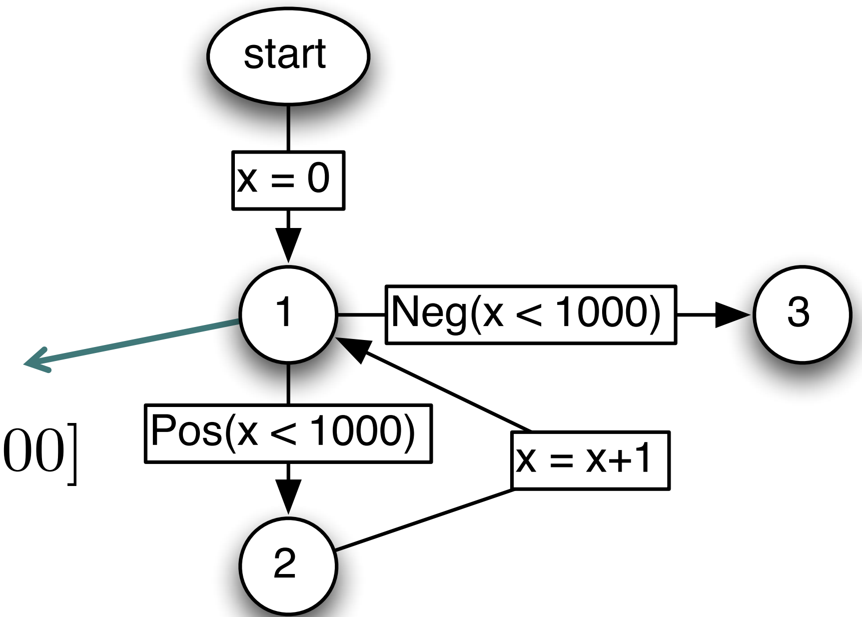
Do we need to apply widening "everywhere"?
Do we need to apply widening "immediately"?

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.*

Do we need to apply widening "everywhere"?
Do we need to apply widening "immediately"?

Selective Application of Widening

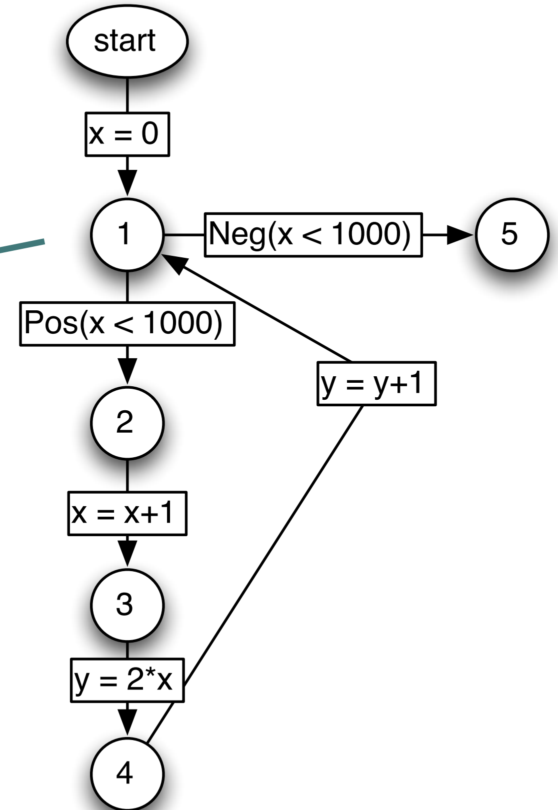
- To ensure convergence it is sufficient to apply widening at cut points.
  - Cut points = set of locations that cut each loop (in the control-flow graph)
- Delayed widening: apply a fixed number of rounds of standard Kleene iteration before starting to apply widening operator.

Another Example: Interval Analysis with Sophisticated Widening

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

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

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

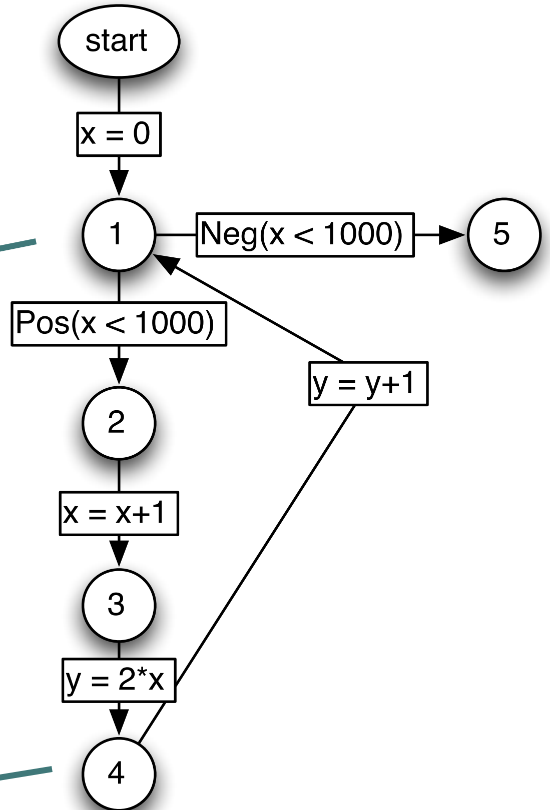


Another Example: Interval Analysis with Sophisticated Widening

$x \mapsto [0, 0]$

$x \mapsto [0, 1]$

$x \mapsto [0, 1000]$



$y \mapsto [2, 2]$

$y \mapsto [2, 1000]$

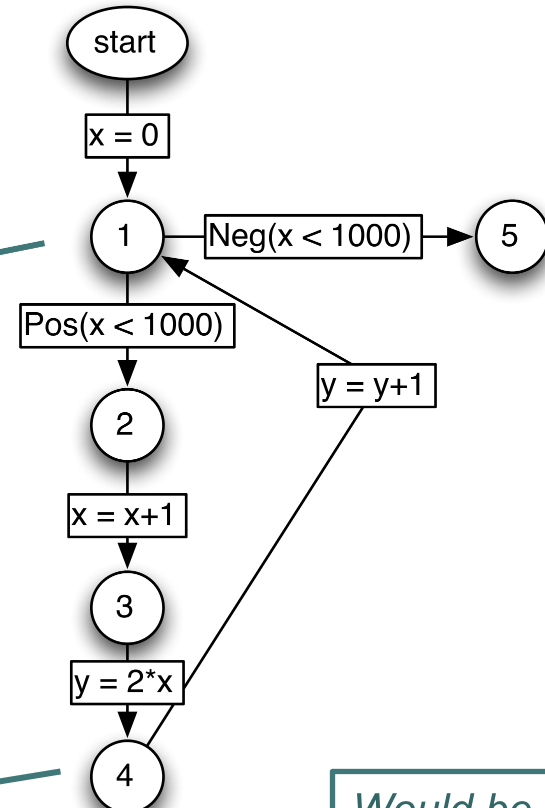
$y \mapsto [2, \infty]$

Another Example: Interval Analysis with Sophisticated Widening

$x \mapsto [0, 0]$

$x \mapsto [0, 1]$

$x \mapsto [0, 1000]$



$y \mapsto [2, 2]$

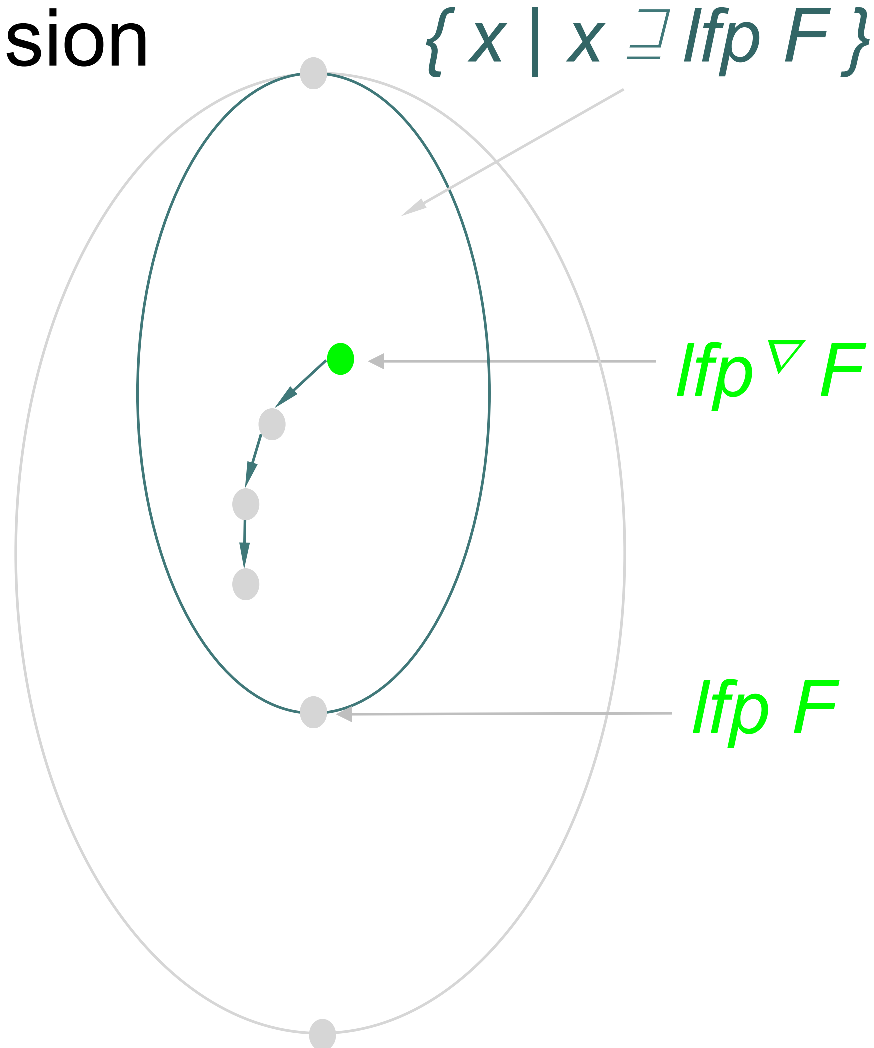
$y \mapsto [2, 1000]$

$y \mapsto [2, \infty]$

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

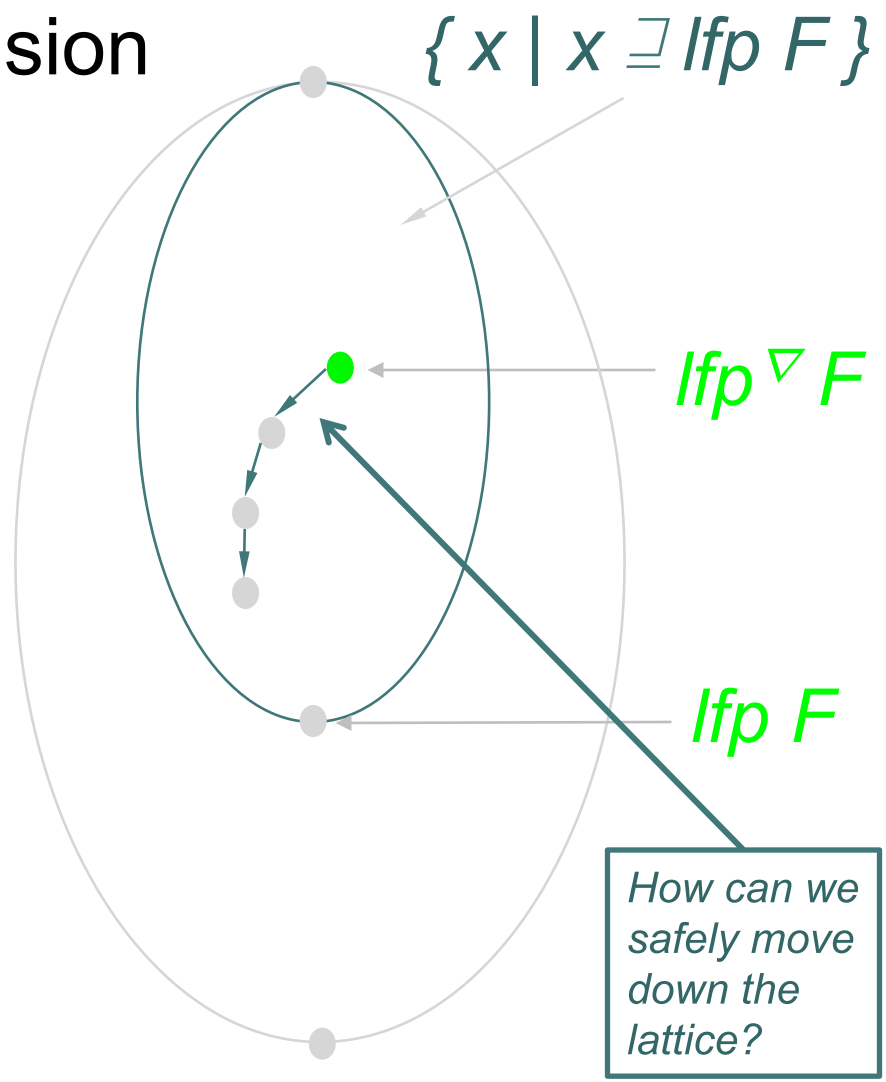
Narrowing: Recovering Precision

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



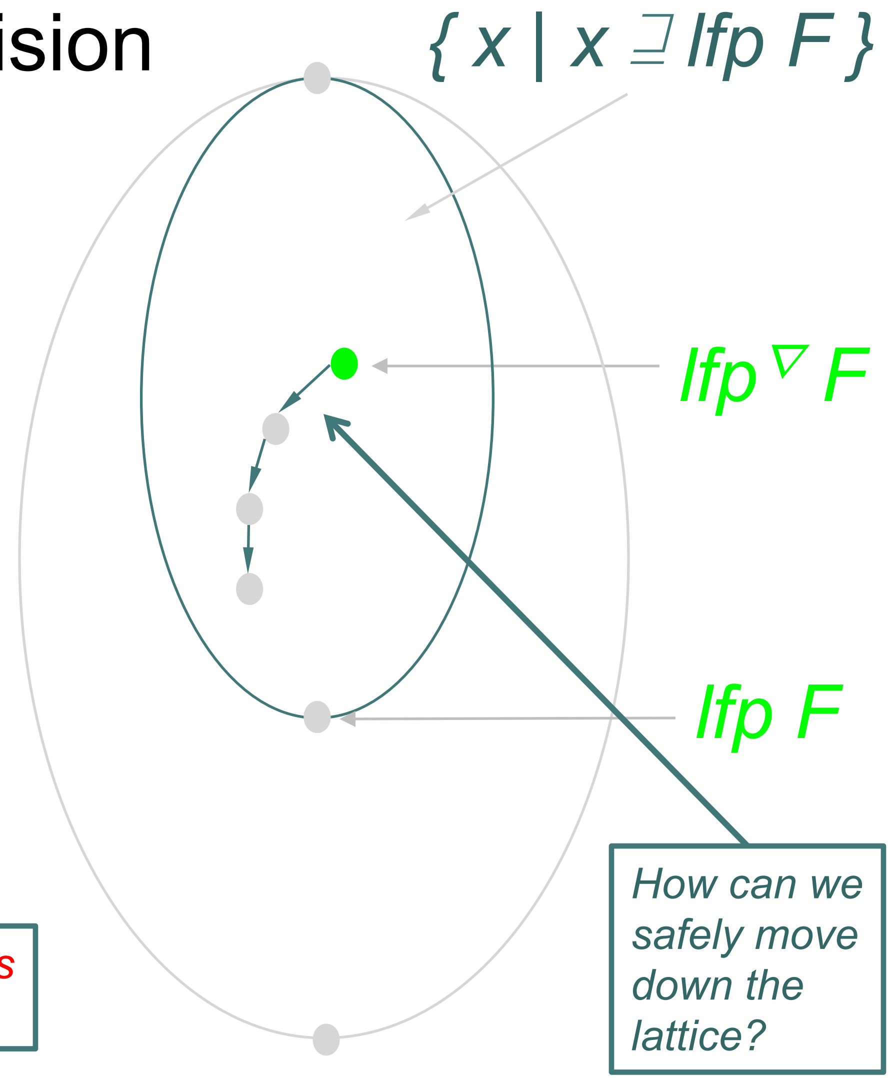
Narrowing: Recovering Precision

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



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.



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:

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

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: $l \sqsubseteq x$ and $l \sqsubseteq y \rightarrow l \sqsubseteq (x \Delta y) \sqsubseteq x$
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.

Narrowing: Formal Requirement

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

1. Safety: $l \sqsubseteq x$ and $l \sqsubseteq y \;\rightarrow\; l \sqsubseteq (x \,\Delta\, y) \sqsubseteq x$
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?

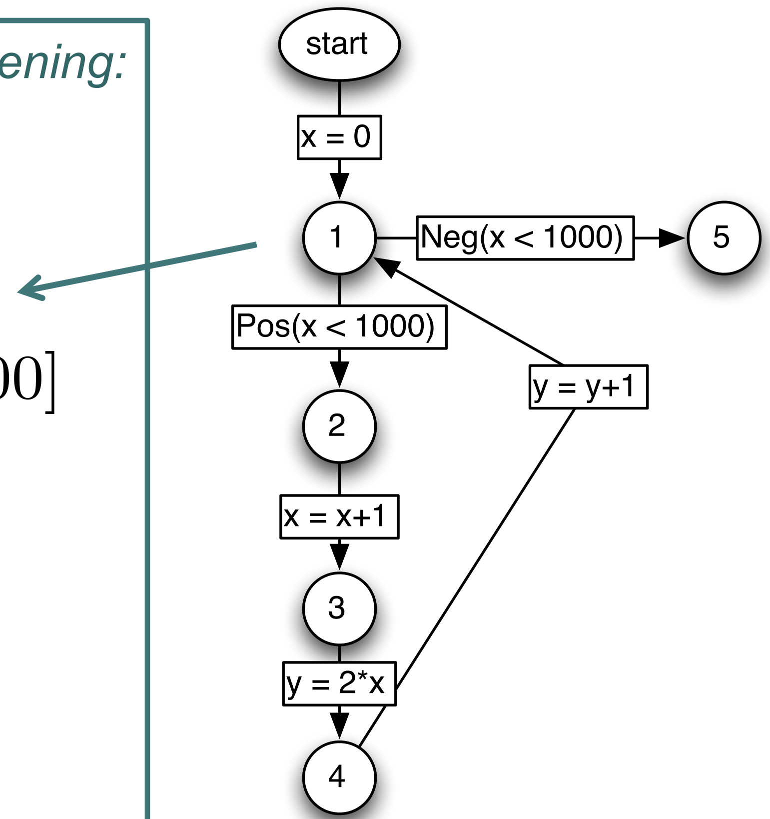
Another Example Revisited: Interval Analysis with Widening and Narrowing

Result after Widening:

$x \mapsto [0, 0]$

$x \mapsto [0, 1]$

$x \mapsto [0, 1000]$



Result after Narrowing:

Another Example Revisited: Interval Analysis with Widening and Narrowing

Result after Widening:

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

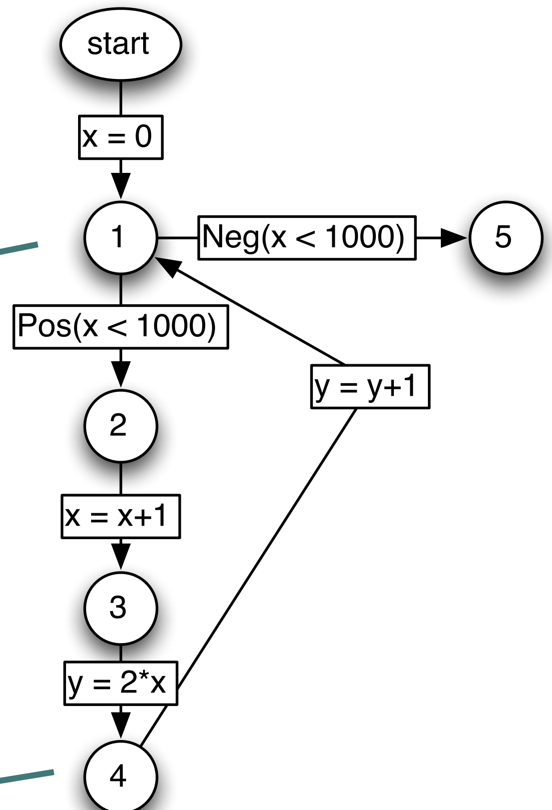
$$x \mapsto [0,1]$$

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

$$y \mapsto [2,2]$$

$$y \mapsto [2,1000]$$

$$y \mapsto [2,\infty]$$



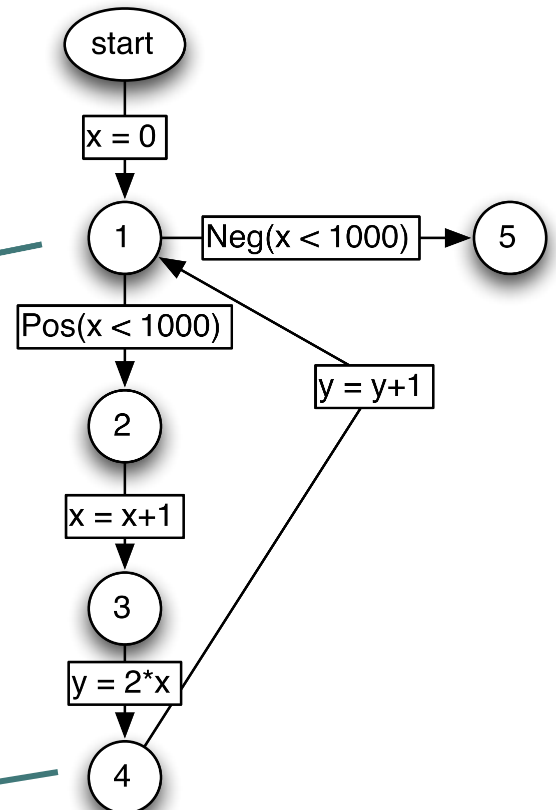
Result after Narrowing:

Another Example Revisited: Interval Analysis with Widening and Narrowing

Result after Widening:

$$x \mapsto [0,0]$$
$$x \mapsto [0,1]$$
$$x \mapsto [0,1000]$$

$$y \mapsto [2,2]$$
$$y \mapsto [2,1000]$$
$$y \mapsto [2,\infty]$$



Result after Narrowing:

→ *Precisely the least fixed point!*

Another Example Revisited: Interval Analysis with Widening and Narrowing

Result after Widening:

$x \mapsto [0, 0]$

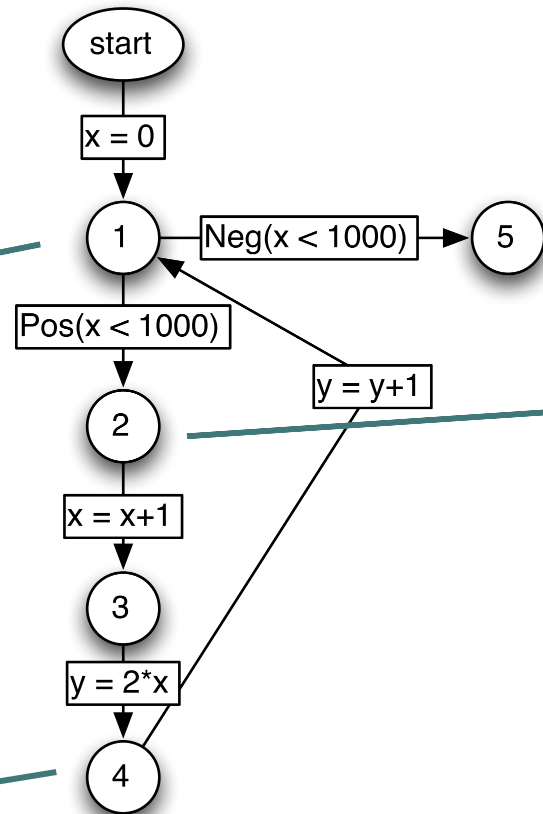
$x \mapsto [0, 1]$

$x \mapsto [0, 1000]$

$y \mapsto [2, 2]$

$y \mapsto [2, 1000]$

$y \mapsto [2, \infty]$



Result after Narrowing:

$x \mapsto [0, 999]$

→ *Precisely the least fixed point!*

Another Example Revisited: Interval Analysis with Widening and Narrowing

Result after Widening:

$x \mapsto [0, 0]$

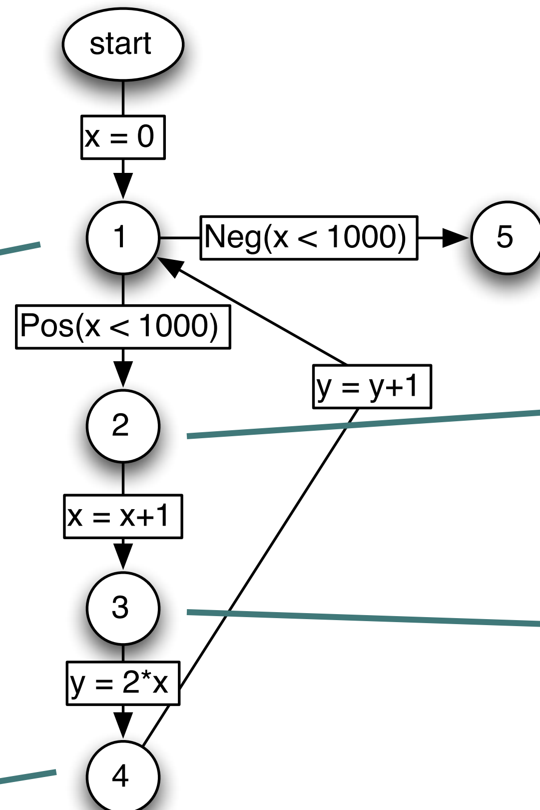
$x \mapsto [0, 1]$

$x \mapsto [0, 1000]$

$y \mapsto [2, 2]$

$y \mapsto [2, 1000]$

$y \mapsto [2, \infty]$



Result after Narrowing:

$x \mapsto [0, 999]$

$x \mapsto [1, 1000]$

→ *Precisely the least fixed point!*

Another Example Revisited: Interval Analysis with Widening and Narrowing

Result after Widening:

$x \mapsto [0, 0]$

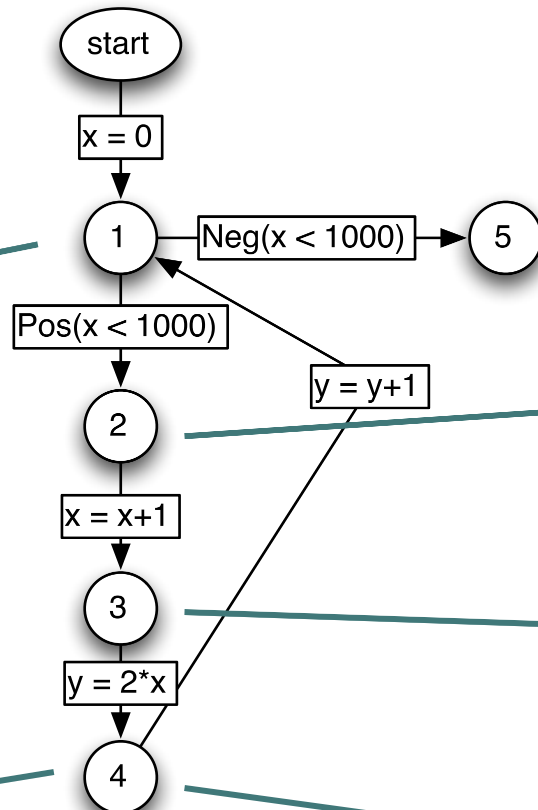
$x \mapsto [0, 1]$

$x \mapsto [0, 1000]$

$y \mapsto [2, 2]$

$y \mapsto [2, 1000]$

$y \mapsto [2, \infty]$



Result after Narrowing:

$x \mapsto [0, 999]$

$x \mapsto [1, 1000]$

$y \mapsto [2, 2000]$

→ *Precisely the least fixed point!*

Another Example Revisited: Interval Analysis with Widening and Narrowing

Result after Widening:

$x \mapsto [0, 0]$

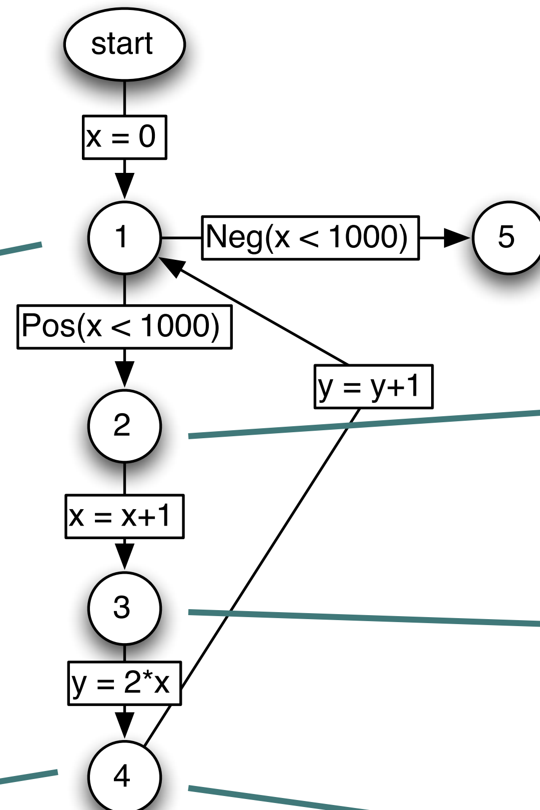
$x \mapsto [0, 1]$

$x \mapsto [0, 1000]$

$y \mapsto [2, 2]$

$y \mapsto [2, 1000]$

$y \mapsto [2, \infty]$



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!*

Applications of Numerical Domains

As input to other analyses:

- Cache Analysis
- To detect dependencies between memory accesses in pipeline
- Loop Bound Analysis:
  - Instrument program with loop iteration counters
  - Determine maximal value of counter
  - Requires relational analysis

State of the Art in Loop Bound Analysis

Multiple approaches of varying sophistication

- Pattern-based approach
- Data-flow 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

```
for (x < 6)
{
    …
    x++;
}
```

Loop Bound Analysis: Pattern-based Approach

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

```
for (x < 6)
{
    …
    x++;
}
```

No modification of x.

Loop Bound Analysis: Pattern-based Approach

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

```
for (x < 6)
{
    …
    x++;
}
```

Initial value of x?

No modification of x.

→ Loop bound: 6-minimal value of x

Loop Bound Analysis: Data-flow-based Approach [Cullmann and Martin, 2007]

Combination of multiple analyses:

1. Identify possible loop counters
2. “Invariant analysis”: determine how loop counters may change in one loop iteration
3. Bound calculation: combine results from step 2 with branch conditions

Loop Bound Analysis: Data-flow-based Approach [Cullmann and Martin, WCET 2007]

Example:

```
for (x < 6) {
    y++;
    if (y % 2 = 0)
      x++;
    else
      x = x+2;
    z++;
}
```

Loop Bound Analysis: Data-flow-based Approach [Cullmann and Martin, WCET 2007]

Example:

```
for (x < 6) {
    y++;
    if (y % 2 = 0)
      x++;
    else
      x = x+2;
    z++;
}
```

1. x, y, and z are potential loop counters

Loop Bound Analysis: Data-flow-based Approach [Cullmann and Martin, WCET 2007]

Example:

```
for (x < 6) {
    y++;
    if (y % 2 = 0)
      x++;
    else
      x = x+2;
    z++;
}
```

1. x, y, and z are potential loop counters

2. Invariants:
x'-x in [1,2]
y'-y in [1,1]
z'-z in [1,1]

Loop Bound Analysis: Data-flow-based Approach [Cullmann and Martin, WCET 2007]

Example:

```
for (x < 6) {
    y++;
    if (y % 2 = 0)
      x++;
    else
      x = x+2;
    z++;
}
```

1. x, y, and z are potential loop counters

2. Invariants:
x'-x in [1,2]
y'-y in [1,1]
z'-z in [1,1]

3. Loop bound:
6 assuming x >= 0 initially

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;
      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

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

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.

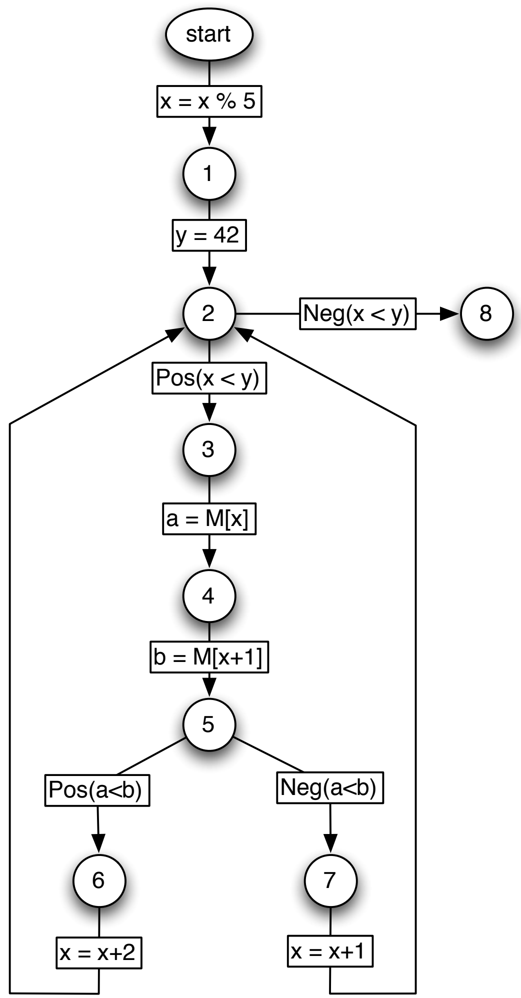
→ Determine variables that are invariant during loop.

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

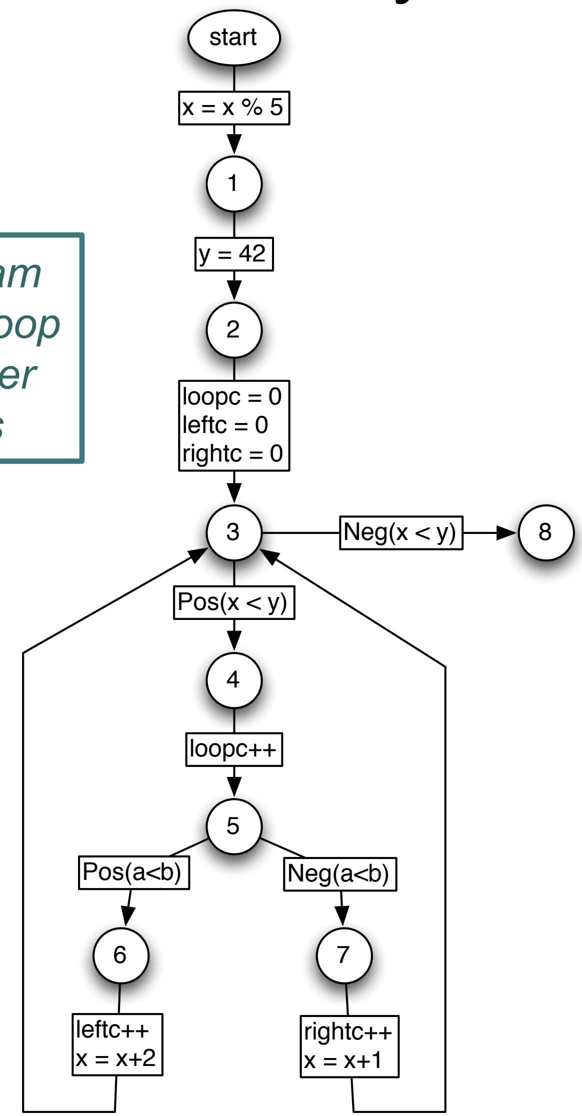
Value Analysis:
INPUT in [10, 20] (assumption)
i in [1, 20] , i % 2 = 1

→ INPUT is invariant!
→ Loop bound 10!

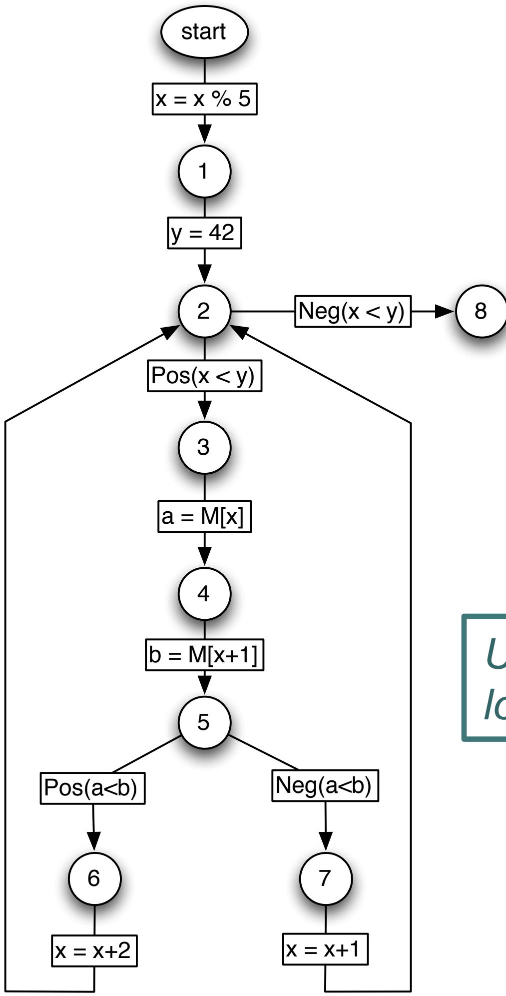
Reduction:
Loop Bound Analysis to Value Analysis



Instrument program with counters of loop iterations and other interesting events

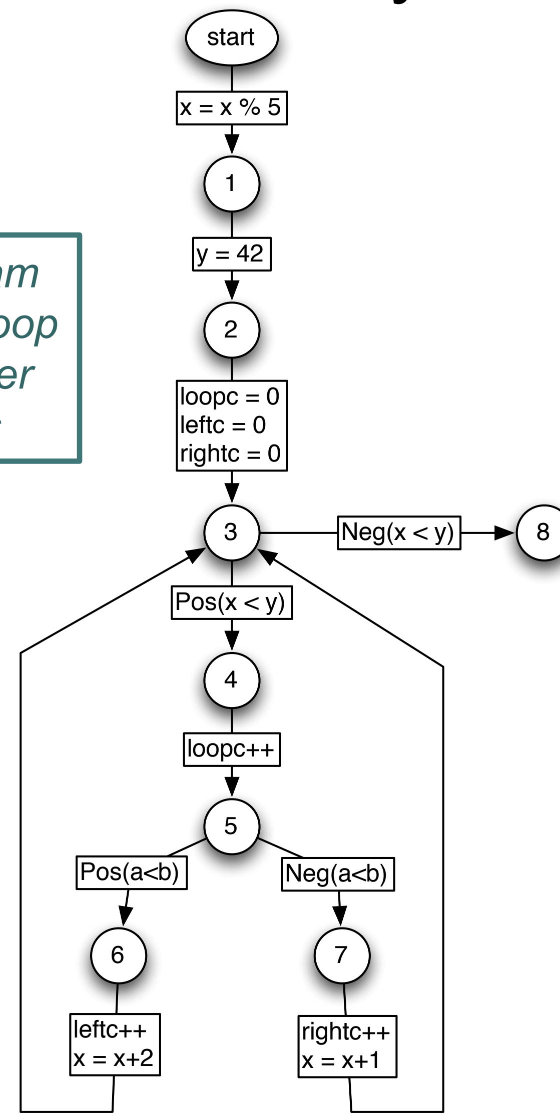


Reduction:
Loop Bound Analysis to Value Analysis

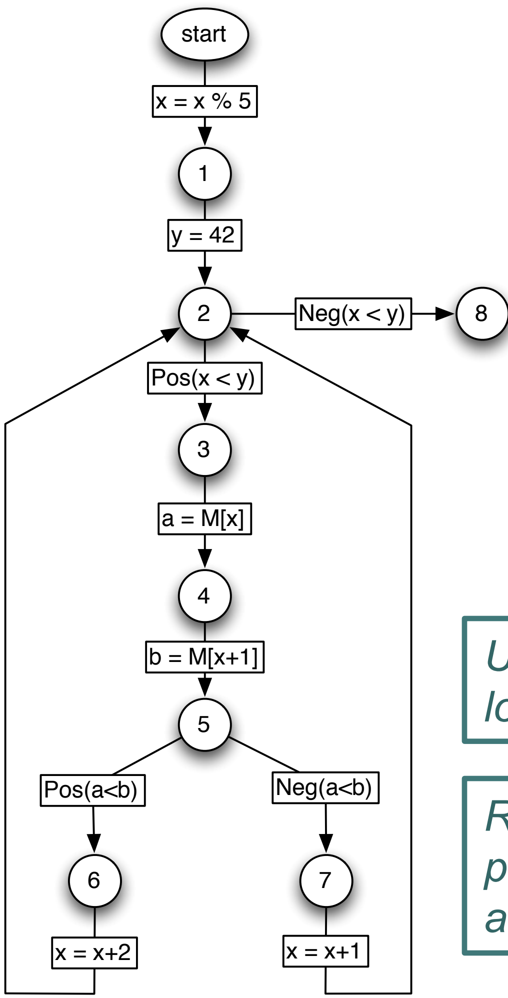


Instrument program with counters of loop iterations and other interesting events

Upper bound for loopc is loop bound!



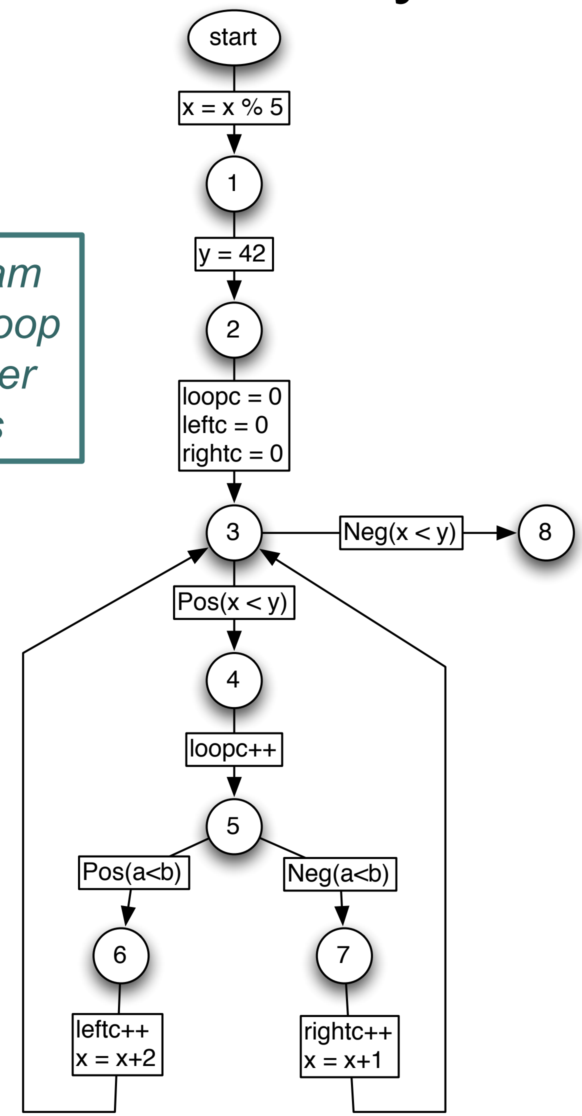
Reduction:
Loop Bound Analysis to Value Analysis



Instrument program with counters of loop iterations and other interesting events

Upper bound for loopc is loop bound!

Requires very powerful relational analysis…



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

Outlook

- Cache Abstractions
- Schedulability Analysis
- Cache-Related Preemption Delay
- Predictable Microarchitectures