Common Domains

Static Analysis section, **Lecture 20**

Pavel Panchekha

CS 6110, U of Utah

24 March 2020

Video Lecture

Online lecture for the rest of the term

Lectures **recorded**, re-watch / stream on Youtube

Attendance cancelled: not taken, everyone marked present

How to **ask questions** during video lectures:

Stay muted; hold space bar to temporarily unmute

Raise hand to ask a question; I'll call on you

Technical support over chat with Manasij

} **Participants** panel

Simplified conditions

Conditions: which variables **known to be non-negative**

Need to know: **which expressions** produce non-negative results

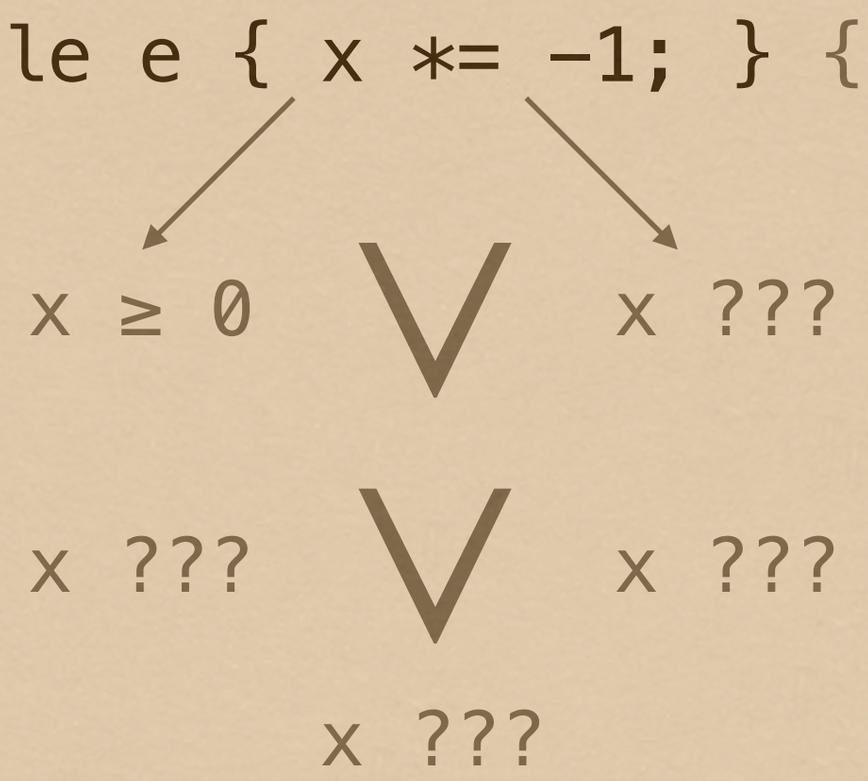
$$(\geq 0) + (\geq 0) \rightsquigarrow (\geq 0) \qquad (\geq 0) \times (\geq 0) \rightsquigarrow (\geq 0) \qquad (\geq 0) \div (\geq 0) \rightsquigarrow (\geq 0)$$

```
{ i ≥ 0 && j ≥ 0 }
m = (i + j) / 2
{ i ≥ 0 && j ≥ 0 && m ≥ 0 }
l[m] ⟵ valid
```

Simple rules instead of solver queries: fast

Loops

If statements require **“or”** for simplified conditions

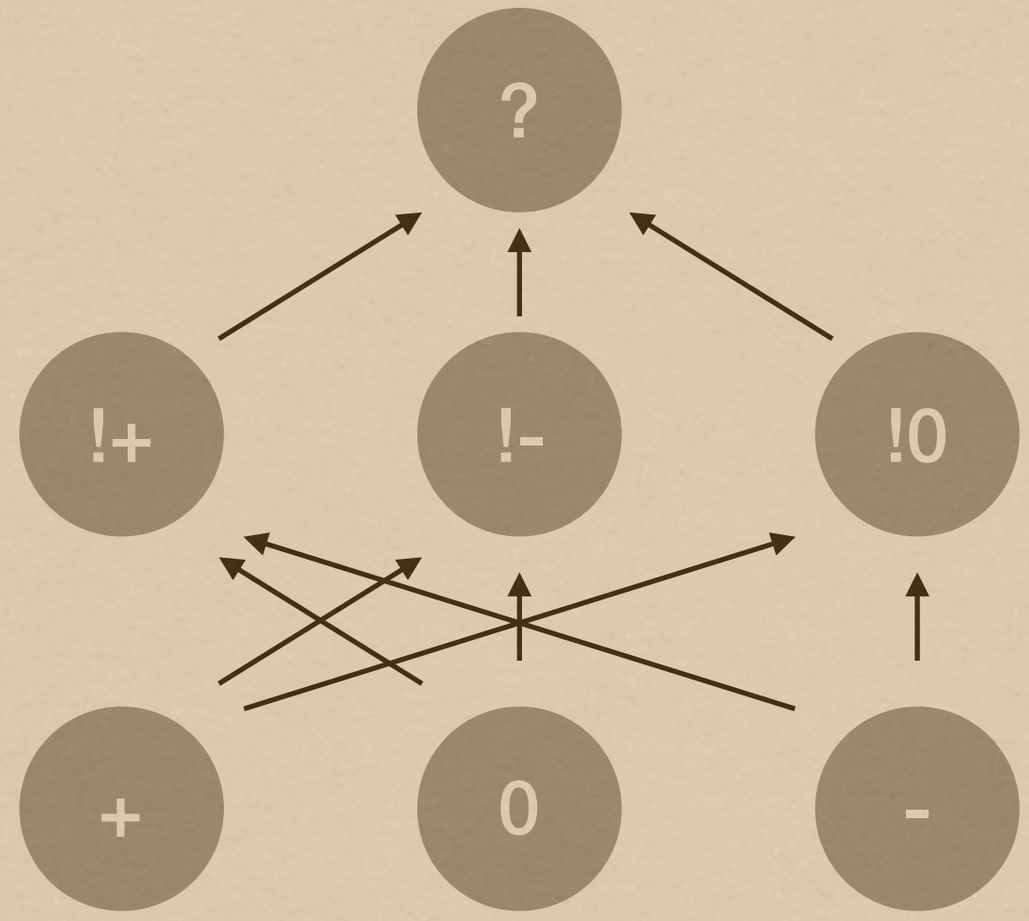


Repeat loop analysis until result stops changing

Example

OR operation produces a **graph structure**

A → B when (A OR B) = B; structure called a **lattice**



Class Progress

Logical reasoning	Program logics	**Static analysis**
Abstract Interpretation	Dependent Types	
Abstract Interpretation	**Common Domains**	Checker Framework

Common Domains

Lay out **complete definition** of a task-specific analysis

Conditions, merge, transfer functions, refinements

Interval domain for integer ranges

And how to deal with infinite-height lattices

Inequality domain for tracking integer order

And how to deal with relational lattices

Refinements

Handling conditionals in an analysis

What's in an Analysis?

Ingredients you need to define a **task-specific** analysis:

— A set of **conditions** for variables and states

Each condition c corresponds to a property $P_c(x)$

— A **merge function** for conditions

If $c = \text{merge}(c_1, c_2)$, then $\forall x, P_{c_1}(x) \vee P_{c_2}(x) \rightarrow P_c(x)$

Constants, $f(x, y)$, ...

— **Transfer functions** for some functions

If $c = \text{trans}_f(c_1)$, then $\forall x, P_{c_1}(x) \rightarrow P_c(f(x))$

Non-Negativity

— A set of **conditions** for variables and states

$$c = nn \mid ?$$

$$P_{nn}(x) = x \geq 0$$

$$P_{?}(x) = \top$$

— A **merge function** for conditions

merge	*nn*	?
nn	*nn*	?
?	?	?

$$P_{?}(x) \vee P_{nn}(x) \rightarrow P_{?}(x)$$

…

— **Transfer functions** for some functions

+	*nn*	?
nn	*nn*	?
?	?	?

$$P_{?}(x) \vee P_{nn}(y) \rightarrow P_{?}(x + y)$$

…

Applying the Analysis

These pieces let you **analyze each function**:

- Initialize parameters to **unknown** condition
- At each step, **over-approximate** results
- **Merge** results of conditional expressions
- **Repeat + merge** until loops stabilize

This leads to a fast, scalable analysis

Non-Negativity

Let's **apply** this analysis step-by-step:

```
lo = 0
hi = len(a)
while …:
    m = (lo + hi) / 2
    if …:
        lo = m
    else:
        hi = m
```

$\text{trans}_0() = nn$

$\text{trans}_{\text{len}}(?) = nn$

$\text{trans}_2() = nn$

$\text{trans}_+(nn, nn) = nn$

$\text{trans}_{\div}, (nn, nn) = nn$

a	lo	hi	m

Flow-sensitivity

Sometimes, **if statements** provide additional info:

```
if i >= 0:
    return a[i]
else:
    return a[len(a) + i]
```

i now known to be non-negative

Add **refinement functions** for the analysis:

If $(c_1', c_2') = \text{refi}_R(c_1, c_2)$, then

$$\forall x, \forall y, P_{c_1}(x) \wedge P_{c_2}(y) \wedge R(x, y) \rightarrow P_{c_1'}(x) \wedge P_{c_2'}(y)$$

Flow-sensitivity

This requires **refinement functions** for the analysis:

If $(c_1', c_2') = \text{refi}_R(c_1, c_2)$, then

$$\forall x, \forall y, P_{c_1}(x) \wedge P_{c_2}(y) \wedge T(x, y) \rightarrow P_{c_1'}(x) \wedge P_{c_2'}(y)$$

For **comparisons** in the non-negativity analysis:

$\geq$	*nn*	?
nn	*nn, nn*	*nn*, ?
?	*nn, nn*	?, ?

$$P_?(x) \wedge P_{nn}(y) \wedge x \geq y \rightarrow P_{nn}(x) \wedge P_{nn}(y)$$

...

Usually add **negations of comparisons** as well

Flow-sensitivity

Sometimes, **if statements** provide additional info:

```
if i >= 0:
    return a[i]
else:
    return a[len(a) - i]
```

$\text{trans}_0() = nn$

$\text{refi}_{\geq}(?, nn) = (nn, nn)$

$i : nn$

Add **refinement functions** for the analysis:

If $(c_1', c_2') = \text{refi}_R(c_1, c_2)$, then

$$\forall x, \forall y, P_{c_1}(x) \wedge P_{c_2}(y) \wedge T(x, y) \rightarrow P_{c_1'}(x) \wedge P_{c_2'}(y)$$

Infinite height lattices

Making sure loop analyses still terminate

Interval analysis

Want to track **valid ranges** for integer variables:

Four kinds of ranges: (a, b), $(-\infty, b)$, (a, ∞), and $?$

$$P_{(a,b)}(x) = a \leq x \leq b \qquad P_{(a,\infty)}(x) = a \leq x$$

$$P_{(-\infty,b)}(x) = x \leq b \qquad P_{?}(x) = \top$$

Merge function computes min and max:

$$\textbf{merge}((a, b), (c, d)) = (\textbf{min}(a, c), \textbf{max}(b, d))$$

Interval analysis

Transfer functions for constants, addition, negation:

$$\textbf{trans}_i\,() = (i, i)$$

$$\textbf{trans}_+((a, b), (c, d)) = (\textbf{min}(a, c), \textbf{max}(b, d))$$

$$\textbf{trans}_-((a, b)) = (-b, -a)$$

Refinement functions for comparison:

$$\textbf{refi}_{\leq}((a, b), (c, d)) = (a, \textbf{min}(b, d)), (\textbf{max}(a, c), d)$$

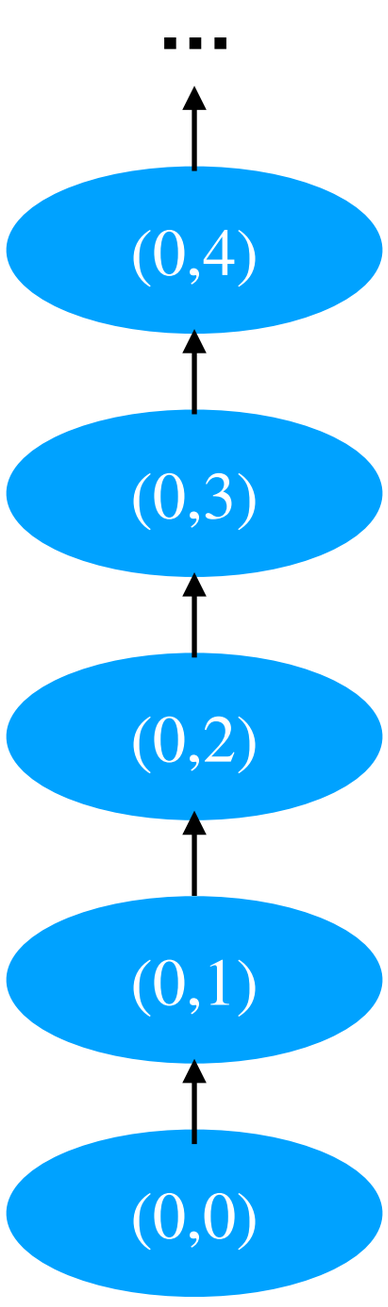
Example

Let's **apply** this analysis step-by-step:

```
output = ["0", "2^x", "???"]

i = popcount(x)

if i >= len(output):

  i = len(output) - 1



return output[i]
```

i	len(output)

Infinite Height



Lattice of intervals has infinitely-long paths

Creates a problem for analyzing **loops**

```
i = 0; while … { i++ }
```

(0, 0) ⋁ (1, 1)

(0, 1) ⋁ (1, 2)

(0, 2) ⋁ (1, 3)

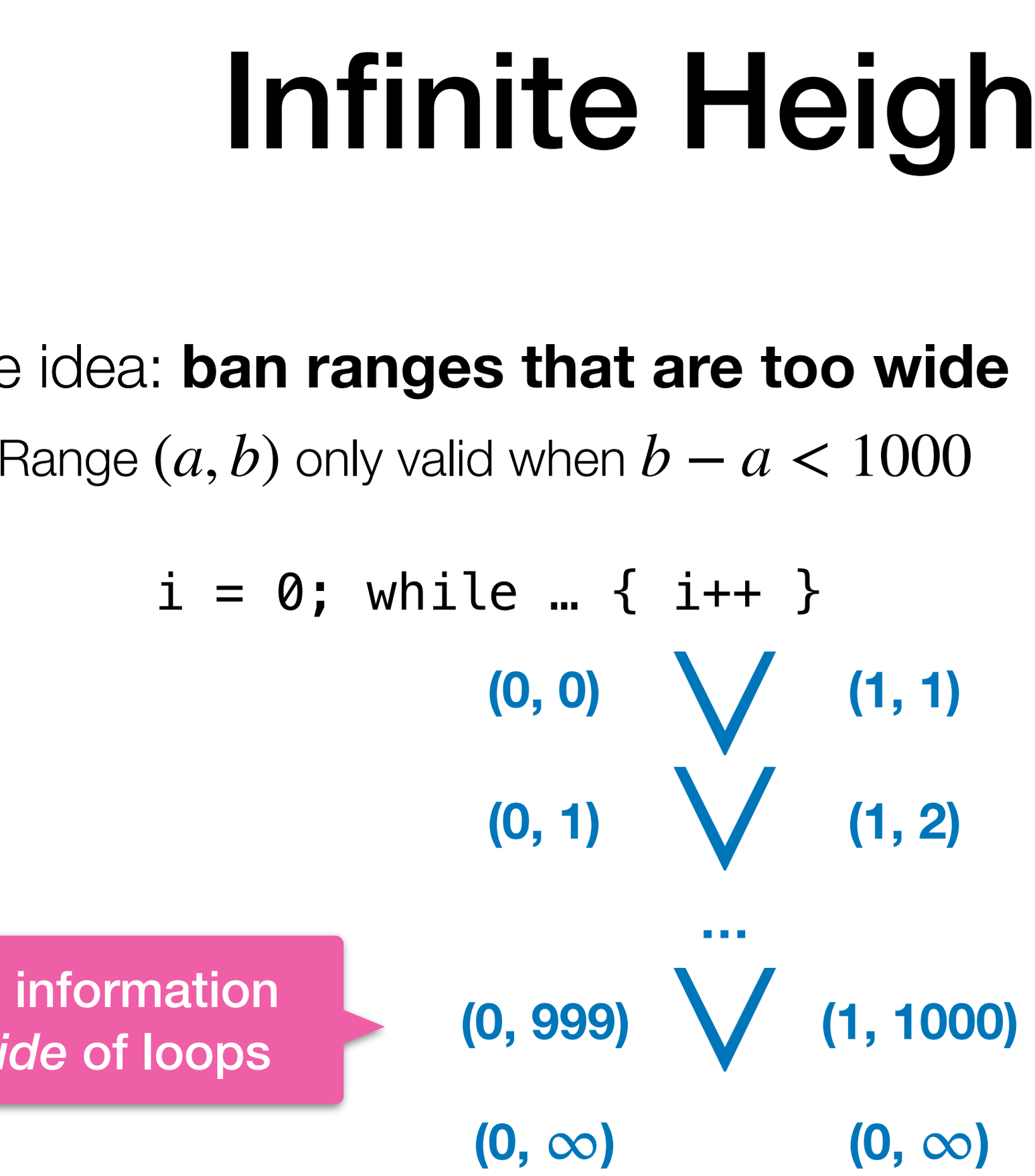
...

Infinite Height

One idea: **ban ranges that are too wide**

Range (a, b) only valid when $b - a < 1000$

```
i = 0; while … { i++ }
```



Lose information *outside* of loops

Infinite Height

One idea: **ban ranges that are too wide**

Range (a, b) only valid when $b - a < 1000$

Only inside loops!

Wide lattice for loops, **narrow** lattice outside loops

Widening and narrowing operators to go between them

narrow$((a, b)) = (a, b)$

wide$((a, b)) = (a, b)$ **if** $b - a < 1000$ **else** (a, ∞)

Relational conditions

What connects two variables?

Inequality analysis

Want to track **which variable is biggest**.

Ranges not enough: each variable **independent**

```
{ x : (−∞, ∞) }
y = x - 1
{ x : (−∞, ∞), y : (−∞, ∞) }
```

Need **relational** conditions on **multiple** values:

Normal	**Relational**
$P_{nn}(x) = x \geq 0$	$P_{<}(x, y) = x < y$

Inequality analysis

Need **more care** when handling statements:

```
{ x < y }
y = 1000000
{ ??? }
```

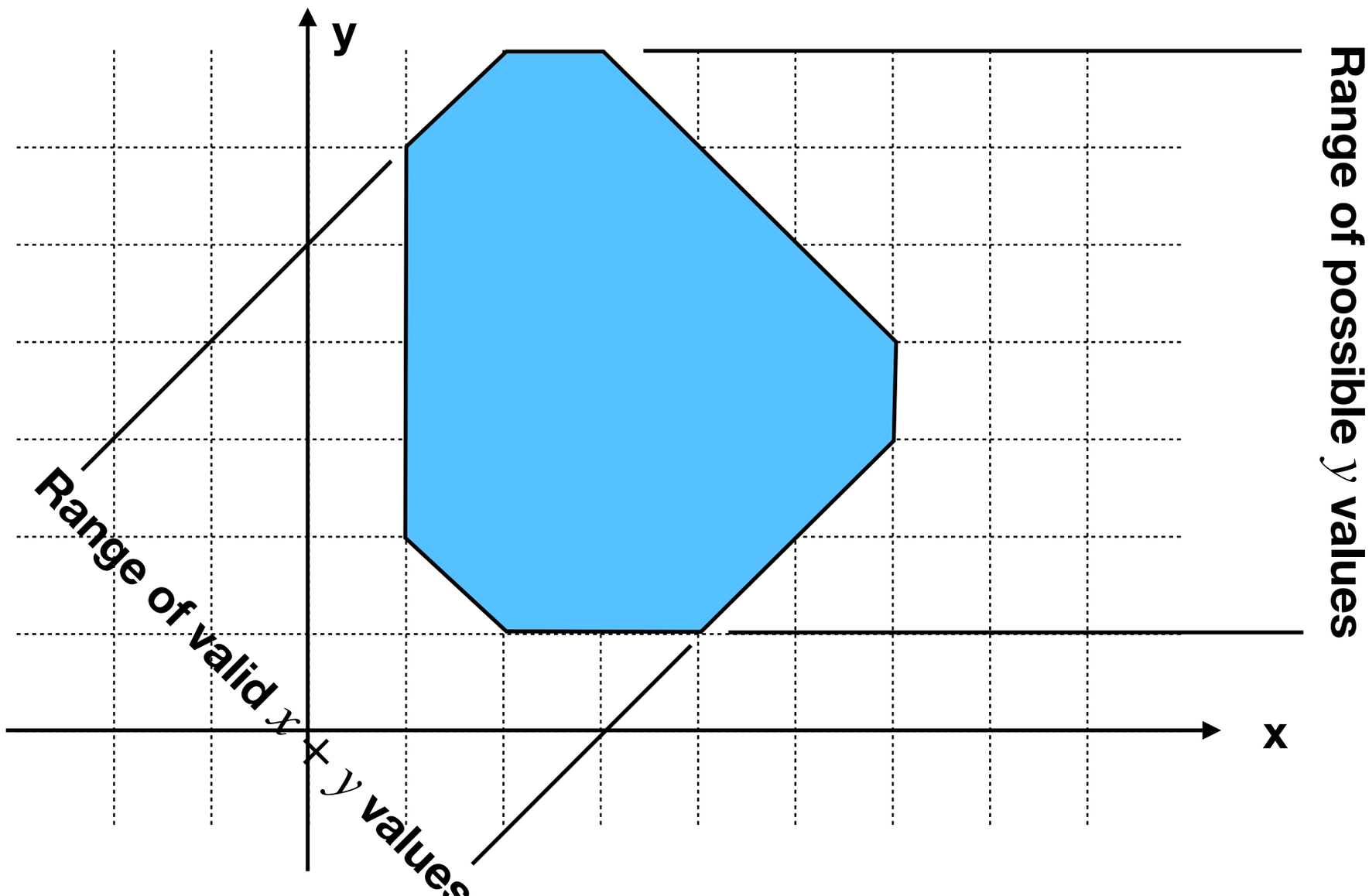
Also need **more care** when writing merge functions:

`x < y, y < z, x < z` `y < z, z < w, y < w`

`y < z`

Inequality analysis

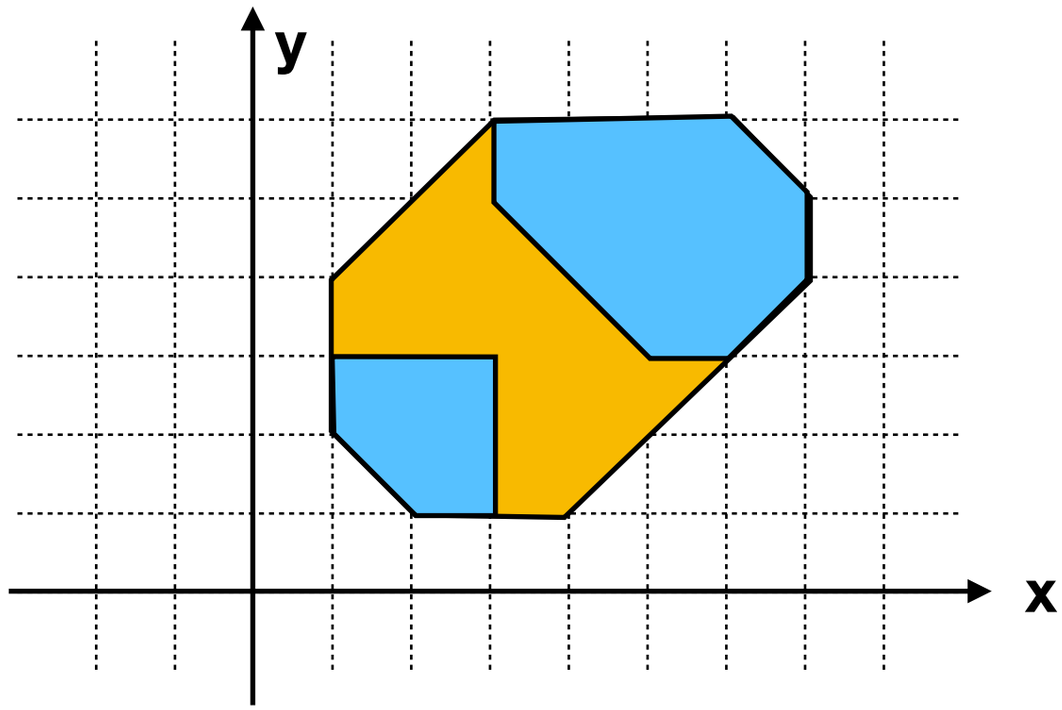
Octagon domain: x, y, $x + y$, and $x - y$ ranges



Inequality analysis

Octagon domain: x, y, $x + y$, and $x - y$ ranges

Octagon union handles the four ranges independently



When $x : (a, b)$ and $y : (c, d)$, bounds on $x \pm y$

Inequality analysis

Octagon domain: x, y, $x + y$, and $x - y$ ranges

Octagon union handles the four ranges independently

Transfer functions for addition, subtraction easy

Handles common idioms like $a[i + 1]$

Refinement functions also simple

$$x \leq y \rightarrow x - y \leq 0$$

Using Octagons

Want to check **array indices in bounds**

- Ranges on integer variables
- Relations over integers and arrays
- Same-length arrays
- Right-hand exclusive indices
- Special support for indexOf operator
- Java-style negative indices for insertions

Solution: **octagons** on array lengths + integer variables

Other Relations

Map has key

Element in array

String matches regex

Lock guards reference

Next class:

Common Domains

To do:

- ☐ Course feedback
- ☐ Milestone II

Common Domains

Lay out **complete definition** of a task-specific analysis

Conditions, merge, transfer functions, refinements

Interval domain for integer ranges

And how to deal with infinite-height lattices

Inequality domain for tracking integer order

And how to deal with relational lattices

DEMO

CHECKER framework

FOR

Next class:

Checker Framework

To do:

- ☐ Course feedback
- ☐ Milestone II