CSE 599n1: Network Verification and Synthesis

Abstract Interpretation

Acknowledgements: Aarti Gupta, Ruzica Piskac, Georg Weissenbacher

Abstract Interpretation

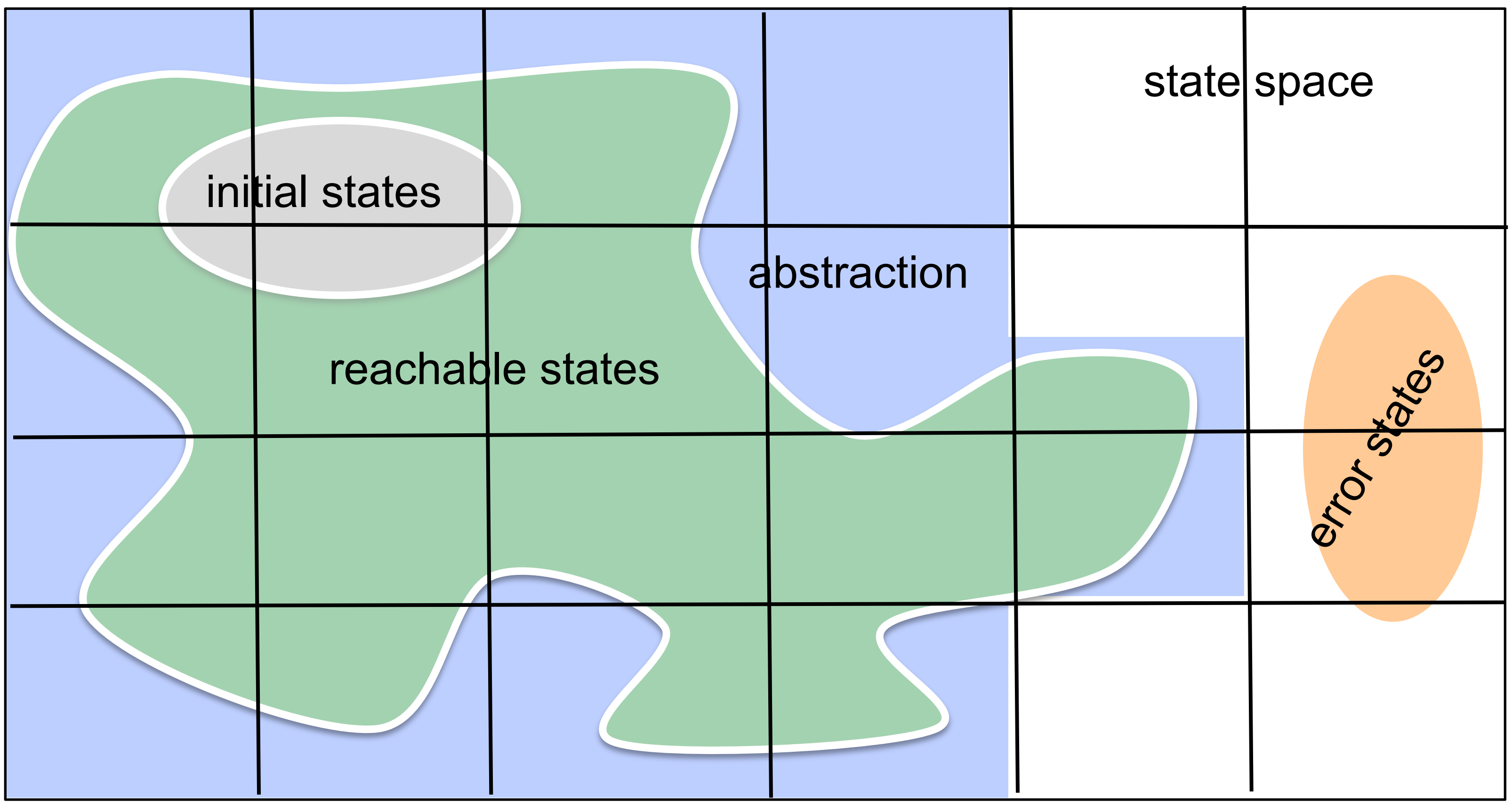
Patrick Cousot, Radhia Cousot [1977]

Approximation is the core idea

Theory of sound program analysis

- How to reason about programs with undecidability?
- Idea: overapproximate the possible program behaviors.
- How do you know you are getting a correct answer?

The big picture



Example: interval abstraction

```
function arrayOutOfBounds(int n, int[10] x)
    a = 0
    if n >= 10 then
        n = n - 5
    else
        a = ++n

    a = math.abs(a – n)
    return x[a]  // safe?
```

Example: interval abstraction

```
function arrayOutOfBounds(int n, int[10] x)
    [0,∞]
    a = 0

    if n >= 10 then

        n = n - 5

    else

        a = ++n


    a = math.abs(a – n)

    return x[a]  // safe?
```

Example: interval abstraction

```
function arrayOutOfBounds(int n, int[10] x)
    [0,∞]
    a = 0
    [0,∞][0,0]
    if n >= 10 then

        n = n - 5

    else

        a = ++n


    a = math.abs(a – n)

    return x[a]  // safe?
```

Example: interval abstraction

```
function arrayOutOfBounds(int n, int[10] x)
    [0,∞]
    a = 0
    [0,∞][0,0]
    if n >= 10 then
        [10,∞][0,0]
        n = n - 5

    else


        a = ++n



    a = math.abs(a – n)

    return x[a]  // safe?
```

Example: interval abstraction

```
function arrayOutOfBounds(int n, int[10] x)
    [0,∞]
    a = 0
    [0,∞][0,0]
    if n >= 10 then
        [10,∞][0,0]
        n = n - 5
        [5,∞][0,0]
    else

        a = ++n


    a = math.abs(a – n)

    return x[a]  // safe?
```

Example: interval abstraction

```
function arrayOutOfBounds(int n, int[10] x)
    [0,∞]
    a = 0
    [0,∞][0,0]
    if n >= 10 then
        [10,∞][0,0]
        n = n - 5
        [5,∞][0,0]
    else
        [0,9][0,0]
        a = ++n


    a = math.abs(a – n)

    return x[a]  // safe?
```

Example: interval abstraction

```
function arrayOutOfBounds(int n, int[10] x)
    [0,∞]
    a = 0
    [0,∞][0,0]
    if n >= 10 then
        [10,∞][0,0]
        n = n - 5
        [5,∞][0,0]
    else
        [0,9][0,0]
        a = ++n
        [1,10][1,10]

    a = math.abs(a – n)

    return x[a]  // safe?
```

Example: interval abstraction

```
function arrayOutOfBounds(int n, int[10] x)
    [0,∞]
    a = 0
    [0,∞][0,0]
    if n >= 10 then
        [10,∞][0,0]
        n = n - 5
        [5,∞][0,0]
    else
        [0,9][0,0]
        a = ++n
        [1,10][1,10]
    [1,∞][0,10]
    a = math.abs(a – n)

    return x[a]  // safe?
```

← Merging branches can lose precision!

Example: interval abstraction

```
function arrayOutOfBounds(int n, int[10] x)
    [0,∞]
    a = 0
    [0,∞][0,0]
    if n >= 10 then
        [10,∞][0,0]
        n = n - 5
        [5,∞][0,0]
    else
        [0,9][0,0]
        a = ++n
        [1,10][1,10]
    [1,∞][0,10]
    a = math.abs(a – n)
    [1,∞][0,9]
    return x[a]  // safe?
```

Abstract Interpretation Idea

Goal: Compute set of values S possible at line of code

But… this is not feasible in general

Compute an overapproximation $S \subseteq S^{\#}$ using abstract values

But… how do we know our abstract operations are sound?

background on lattices

Definition Lattice: $\langle S, \sqsubseteq, \sqcup, \sqcap \rangle$

set of elements — partial order — join — meet

example: naturals lattice

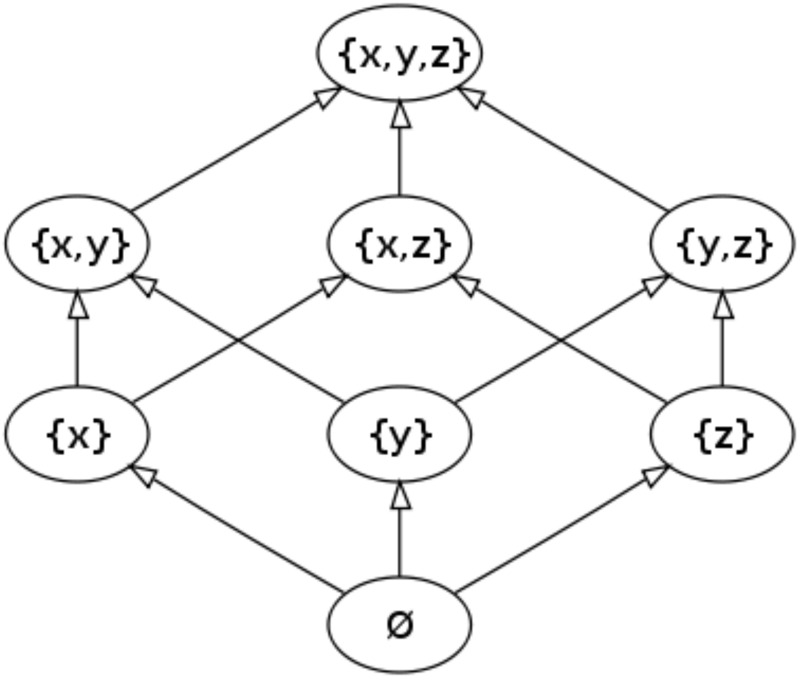
$\langle N, \leq, \max, \min \rangle$ is a lattice.

$1 \sqsubseteq 3$	yes
$2 \sqsubseteq 2$	yes
$2 \sqsubseteq 1$	no
$1 \sqcup 3 = 3$	yes
$3 \sqcap 2 = 2$	yes

example: natural numbers

$\langle P(S), \subseteq, \cup, \cap \rangle$
is the power-set lattice of set S

$S = \{x, y, z\}$



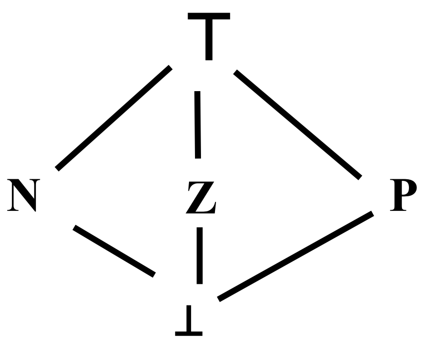
$\emptyset \sqsubseteq \{x, y\}$ yes

$\{x\} \sqcup \{x, y\} = \{x, y, z\}$ no

Abstract Domain

A candidate for abstract domain: Lattice on set {P, N, Z}

- positive numbers (P), negative numbers (N), zero (Z)
- ⊤ = top, “Don't know", represents any value
- ⊥ = represents no value, empty set



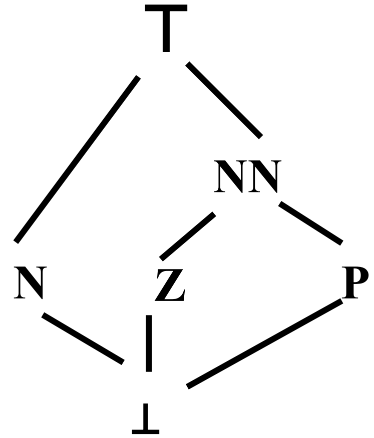
A More Complex Lattice

Better Candidate for the abstract domain

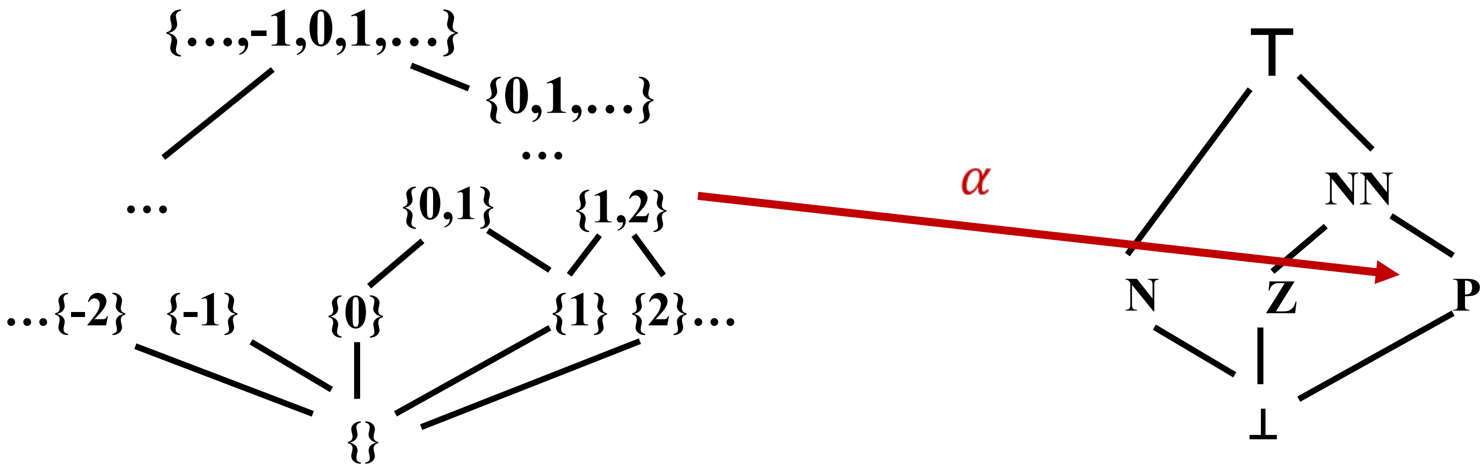
lub(P, Z) = NN (non-negative)

lub(N, P) = ⊤

glb(N, P) = ⊥



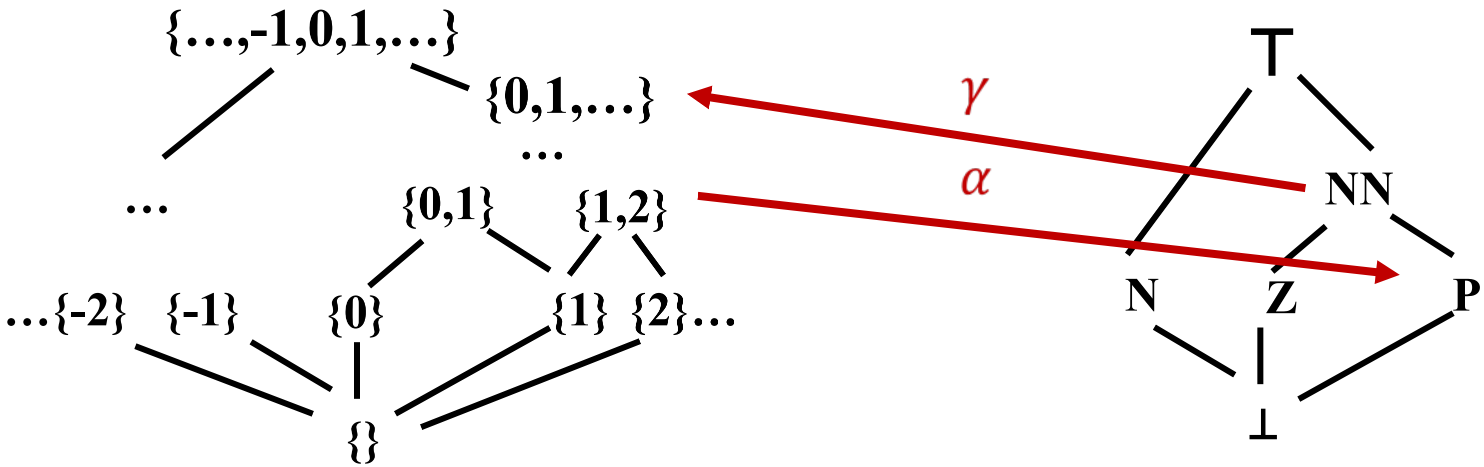
Abstraction and Concretization Functions



Abstraction function α maps sets of concrete elements to the *most precise* value in the abstract domain

- α({2, 10, 0}) = NN
- α({3, 99}) = P
- α({-3, 2}) = ⊤

Abstraction and Concretization Functions



Concretization function γ maps each abstract value to sets of concrete elements

- $\gamma(NN) = \{ x \mid x \in \mathbb{Z} \land x >= 0 \}$

Another example: Interval Abstract Domain

Interval abstract domain: for any set of values: use [lower, upper]

Function α maps concrete values into abstract values that best describe them (abstraction)

$\alpha(\{2, 10\}) = [2,10]$

Function γ maps abstract values into concrete values they represent (concretization)

$\gamma([2,10]) = \{2,3,\ldots,9,10\}$

Abstraction followed by concretization is (usually) an approximation

$\gamma(\alpha(\{2, 10\})) = \gamma([2,10]) = \{2,3,\ldots,9,10\}$

Abstract Interpretation Idea

Goal: Compute set of values S possible at line of code

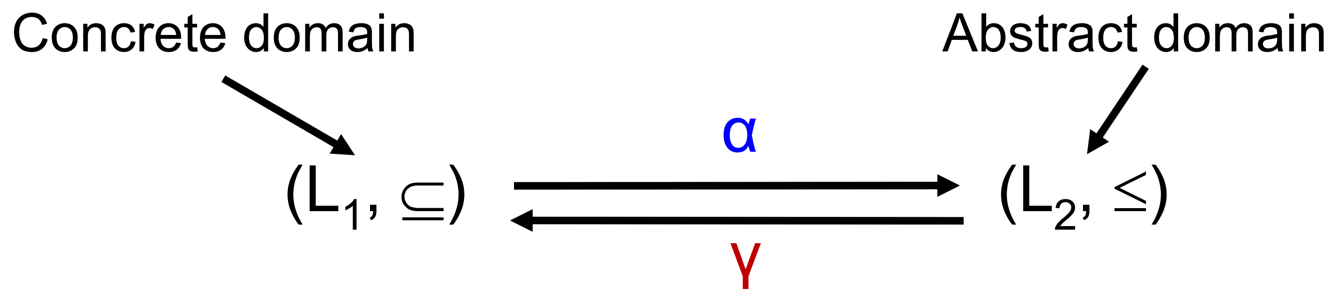
But… this is not feasible in general

Compute an overapproximation $S \subseteq S^{\#}$ using abstract values

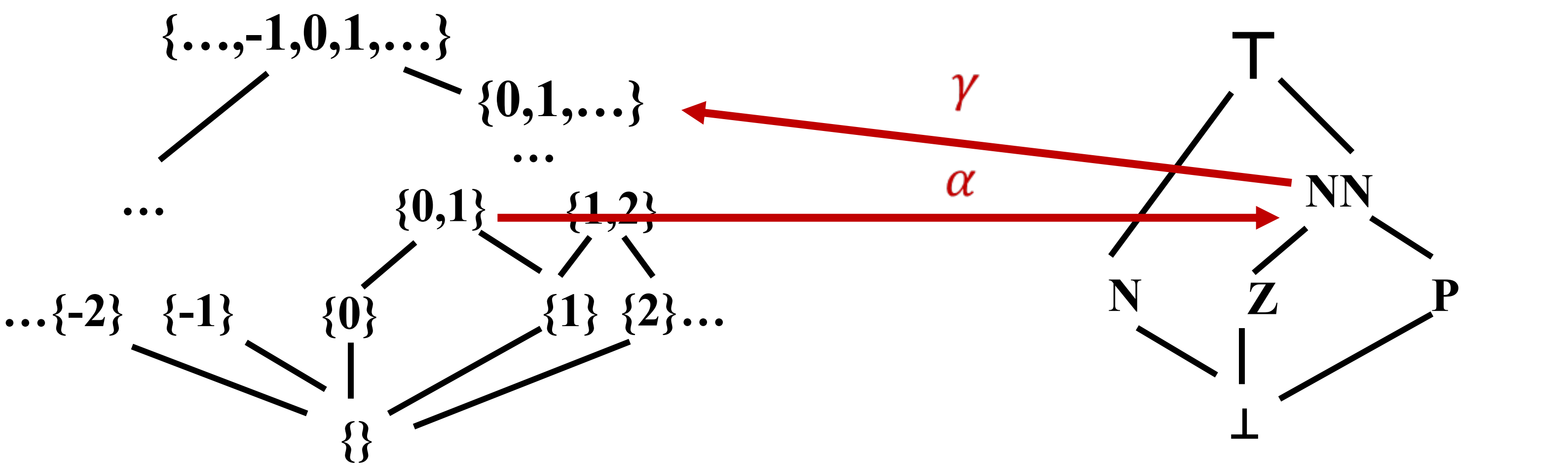
But… how do we know our abstract operations are sound?

Galois Connection

L_1, L_2 are two lattices



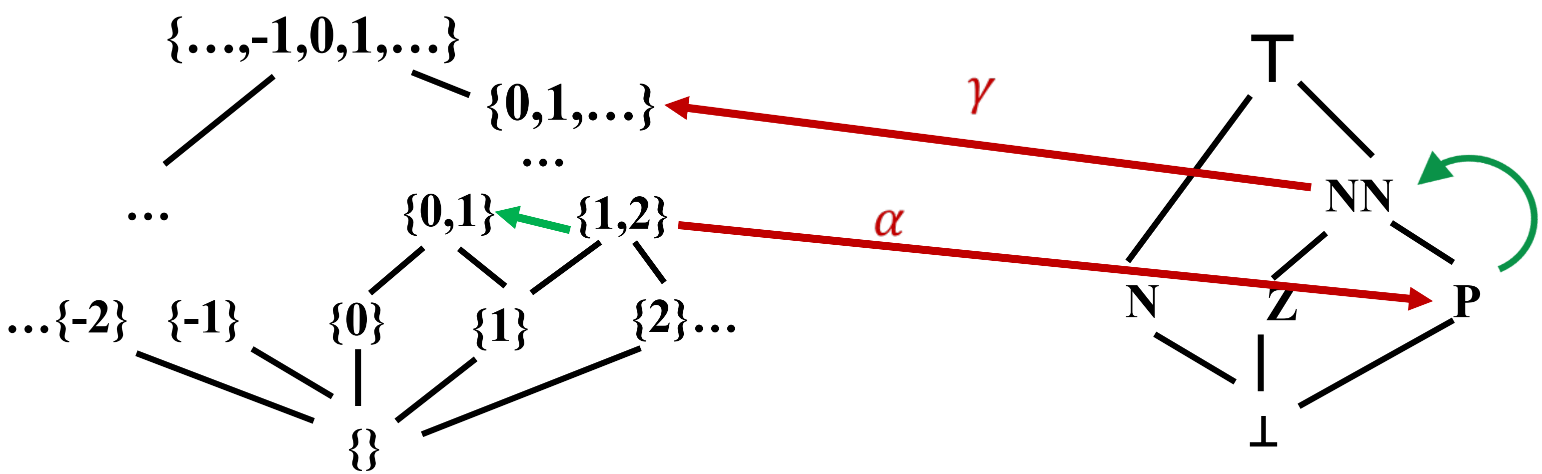
Soundness: $x \sqsubseteq \gamma(\alpha(x))$



Abstract transformers

Given a Galois connection, execute abstract program

Transformer: $x := x - 1$

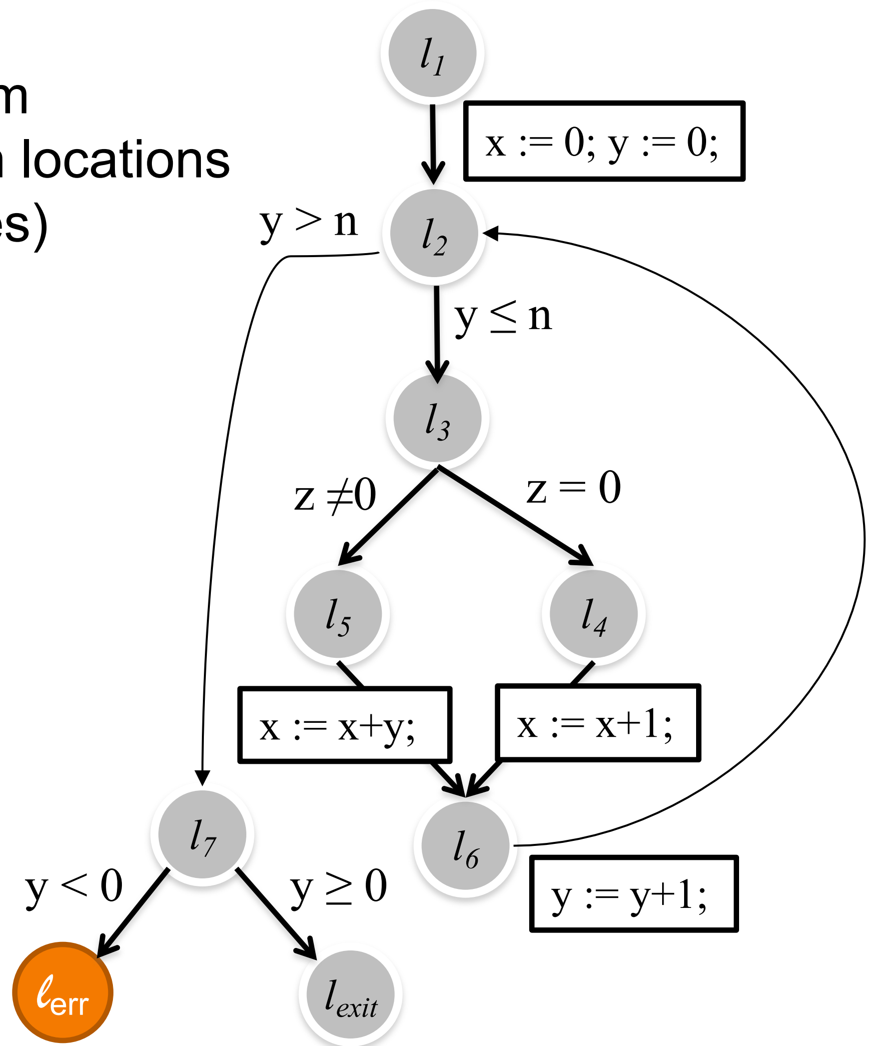


Soundness: $f(\mathrm{x}) \sqsubseteq \gamma\left(f^{\#}(\alpha(x))\right)$

Abstract Interpretation: CFG

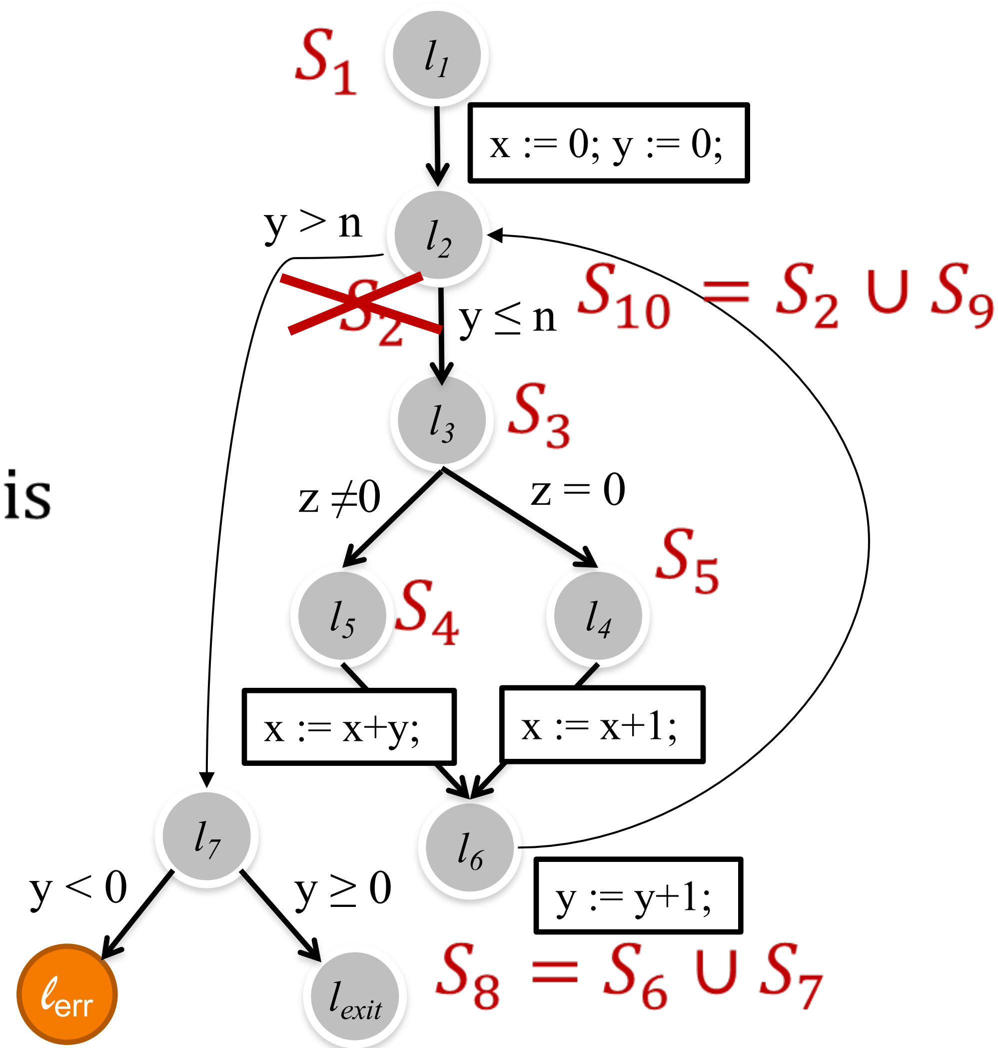
Control Flow Graph
Nodes: locations in program
Edges: transitions between locations
(guards and updates)

```
x := 0;
y := 0;
while (y ≤ n) {
  if (z == 0) {
    x := x+1;
  }  else {
    x := x + y;
  }
  y := y+1
}
assert y ≥ 0
```



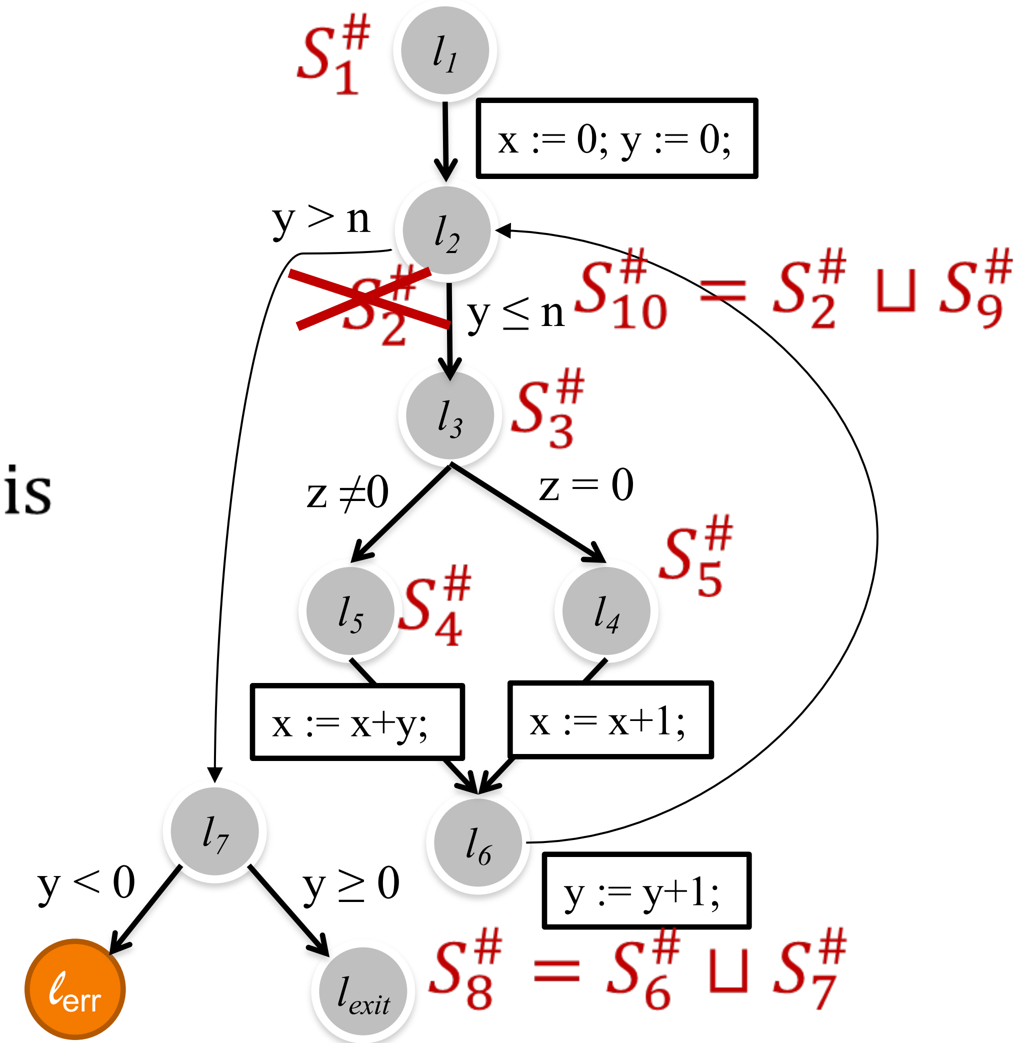
Abstract Interpretation: CFG

Concrete analysis



Abstract Interpretation: CFG

Abstract analysis



Example: interval abstraction

```
function arrayOutOfBounds(int n, int[10] x)
    [0,∞]
    a = 0
    [0,∞][0,0]
    if n >= 10 then
        [10,∞][0,0]
        n = n - 5
        [5,∞][0,0]
    else
        [0,9][0,0]
        a = ++n
        [1,10][1,10]
    [1,∞][0,10]
    a = math.abs(a – n)
    [1,∞][0,9]
    return x[a]  // safe?
```

abstract transformer
for n = n - 10

merge point, compute
[1,∞] = lub([5,∞], [1,10])