Abstract Interpretation

ATSS 2011

How do we verify safety?

1. We define the program semantics with respect to a domain of interest
2. We define validity conditions given the semantics and the domain
3. Give some program P, we want to obtain a guarantee that it will never break the validity conditions

How do we verify safety?

1. We define the program semantics with respect to a domain of interest
2. We define validity conditions given the semantics and the domain
3. Give some program P, we want to obtain a guarantee that it will never break the validity conditions

We want to to **3** automatically for arbitrary programs!

What is Abstract Interpretation?

It is an analysis framework that:

1. Given some defined program semantics under a given domain
2. And specified validity conditions

will do **3** automatically!

What is Abstract Interpretation?

It is an analysis framework that:

1. Given some defined program semantics under a given domain
2. And specified validity conditions

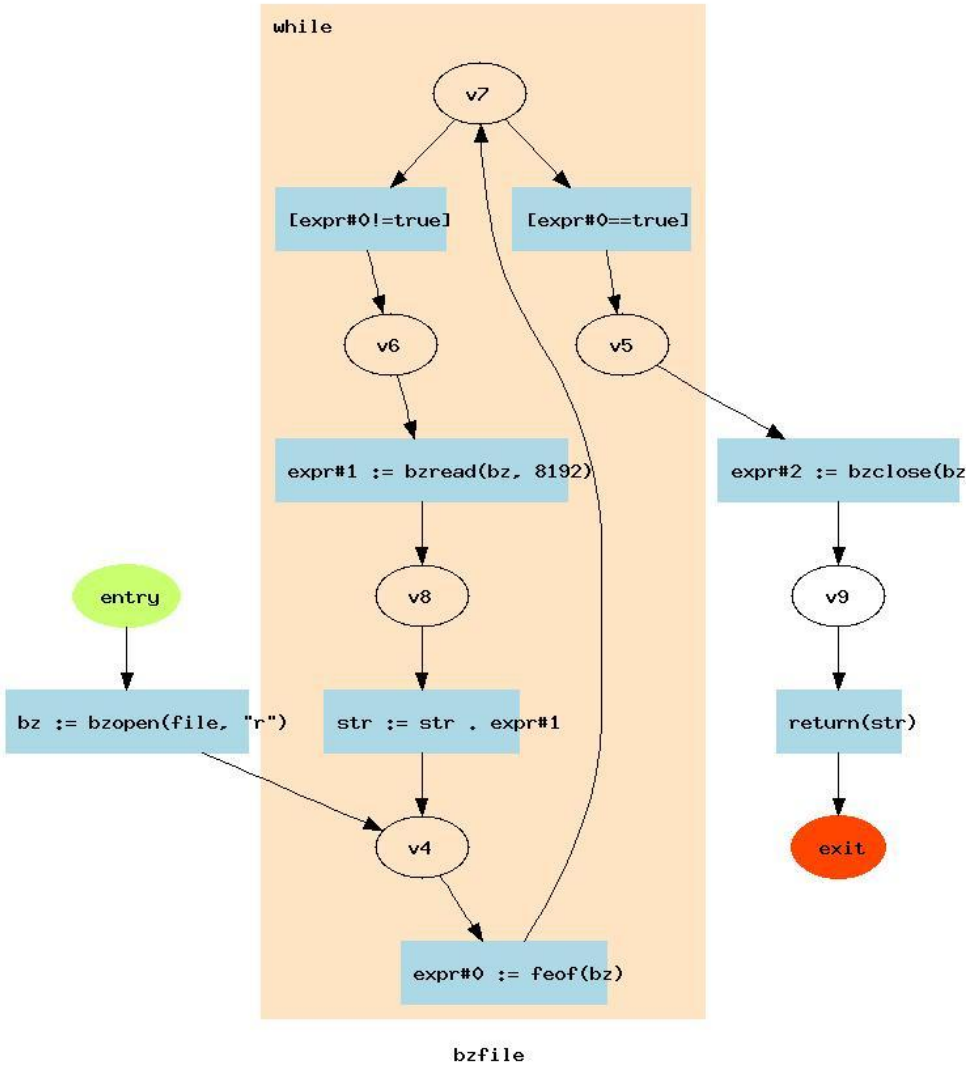
will do **3** automatically!

No free lunch:
It requires specific properties on the domain and on the attached semantics!

Program Representation

CFG: Labeled directed graph representation of the control flow of the program

- Entry and exit points
- Vertices represent program points
- Edges indicate the flow of execution
- Labels contain corresponding statements



Abstract Interpretation Ingredients

Given some concrete domain D_C:

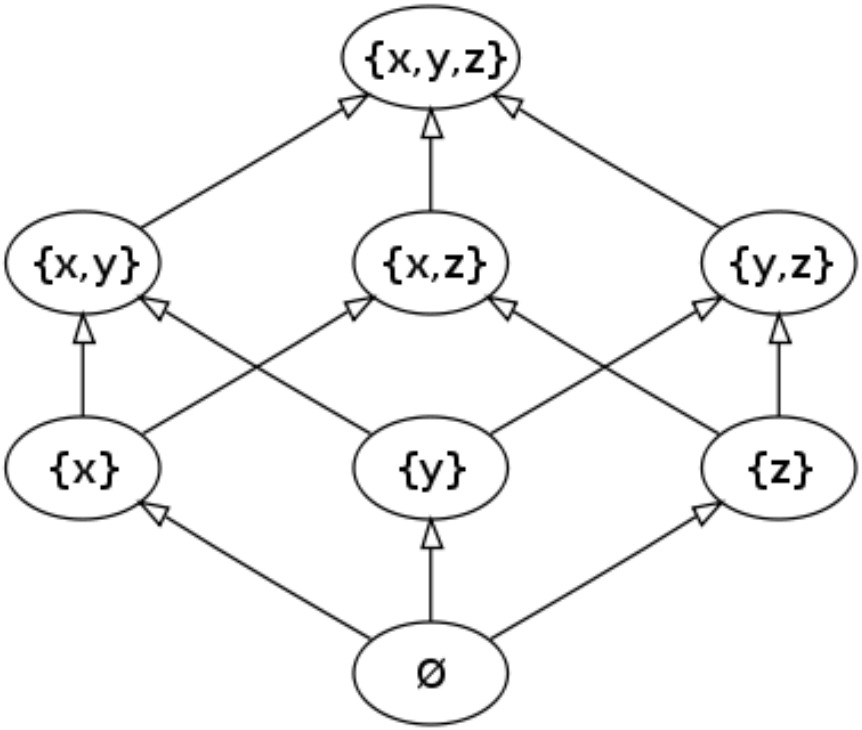
- Abstract Domain D_A forming a lattice
  - An Abstraction Function $D_C \longmapsto D_A$
  - A Concretization Function $D_A \longmapsto D_C$
- The program semantics within D_A:

  A transfer function T_f: $(Stmts \times D_A) \longmapsto D_A$

Lattice

Partially ordered set (D, $\leq$)

- Every $a, b \in D$ there exists a minimal element c s.t. a $\leq c$, b $\leq c$ (lub, join, $\sqcup$)
- It has a top (T) element and a bottom element ($\perp$)



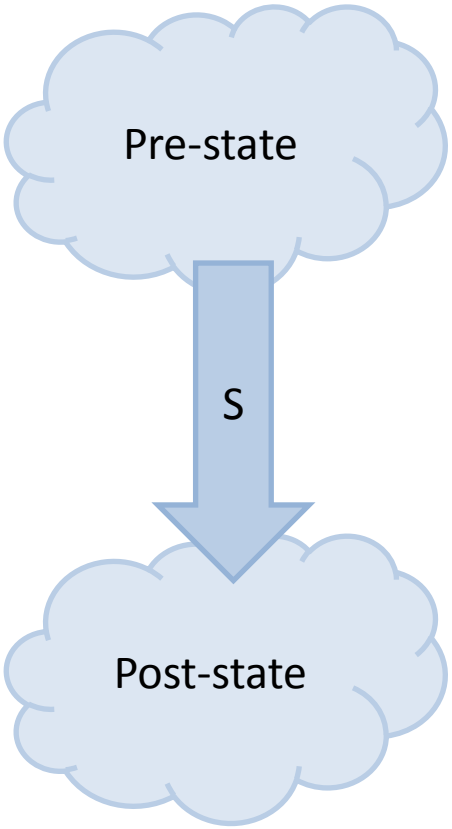
Lattice for $(\wp(\{x, y, z\}), \subseteq)$

Transfer Function

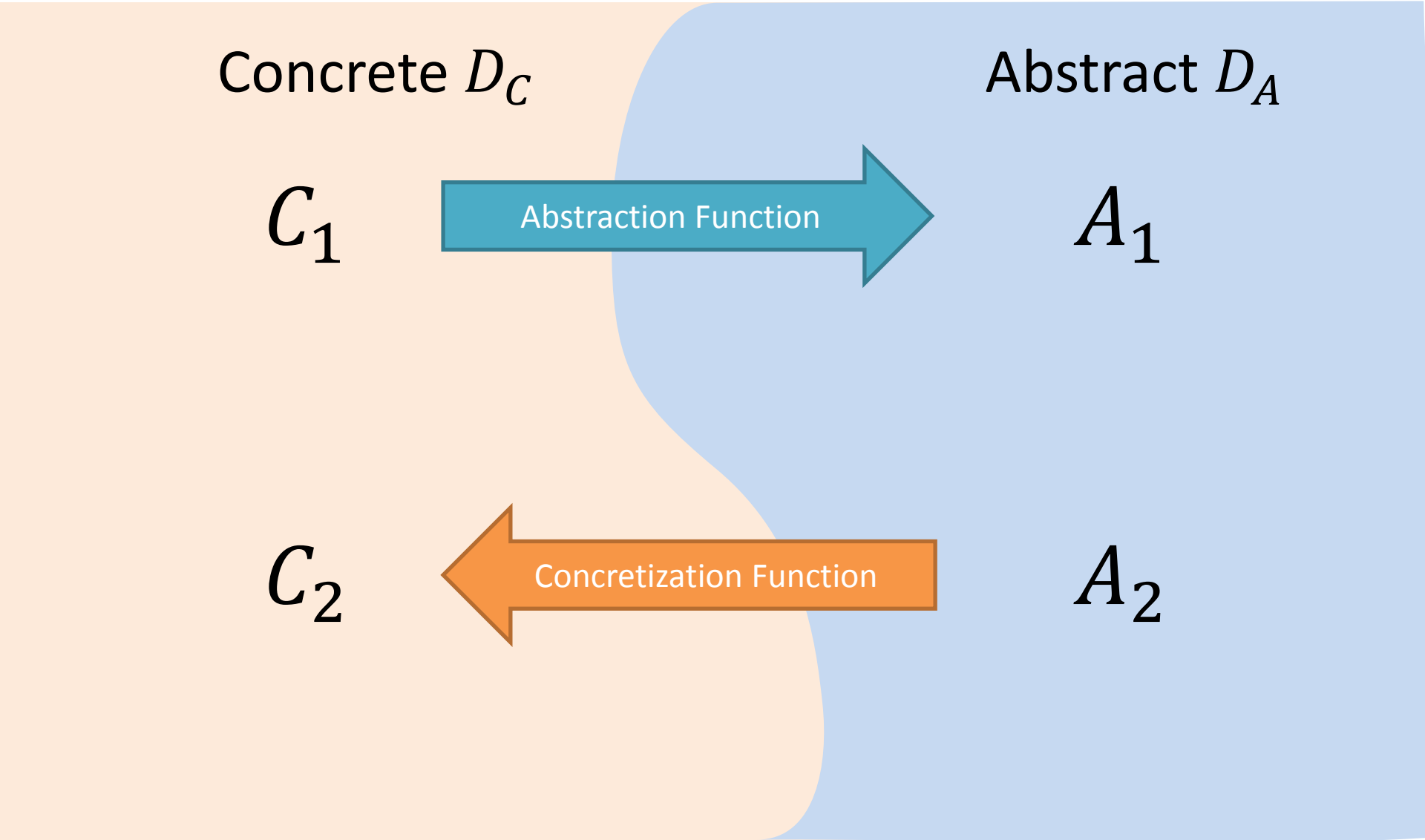
Given a statement S and an abstract pre-state, compute the abstract post-state.

Needs to be monotonous:

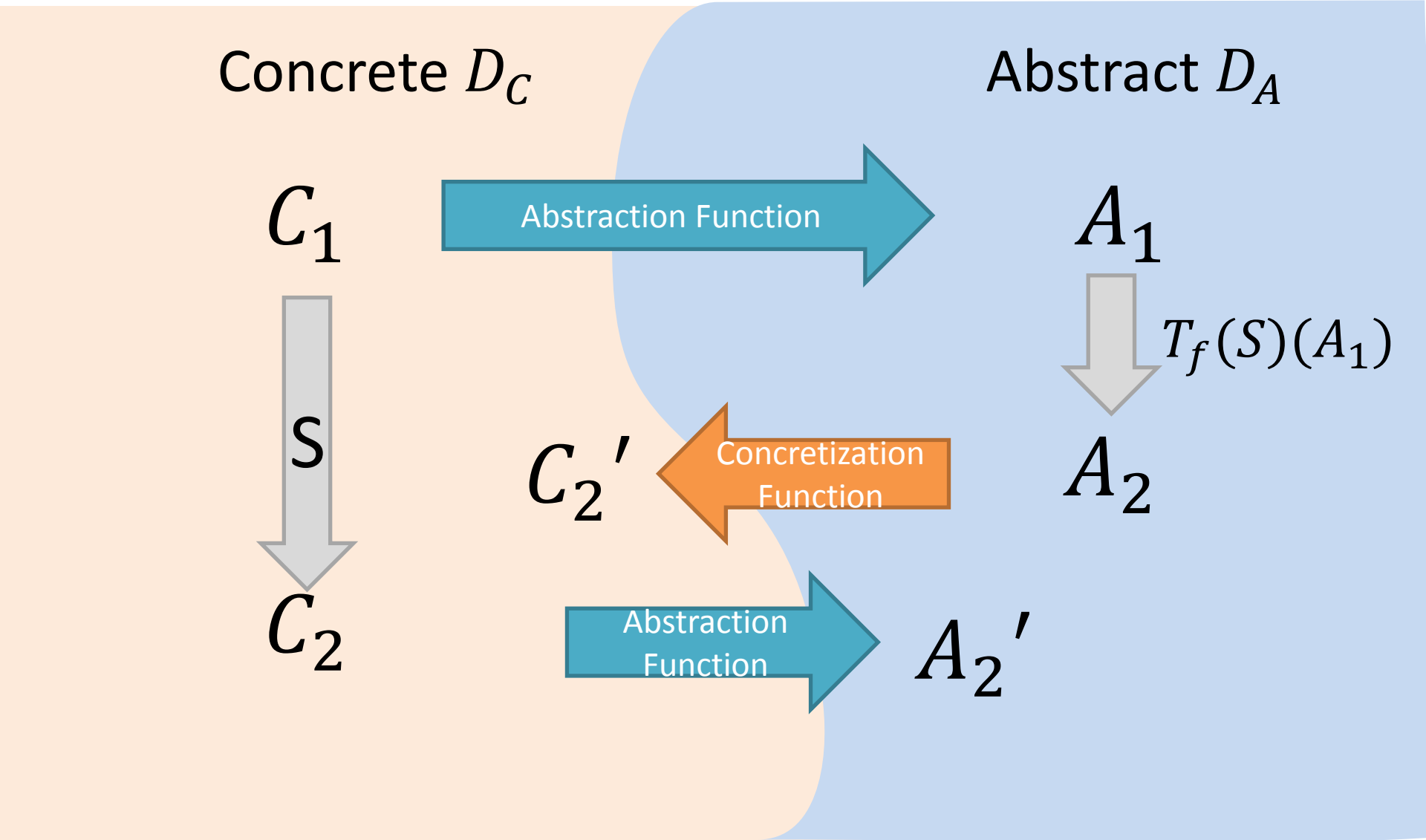
$$A_1 \sqsubseteq A_2 \Rightarrow T_f(S)(A_1) \sqsubseteq T_f(S)(A_2)$$



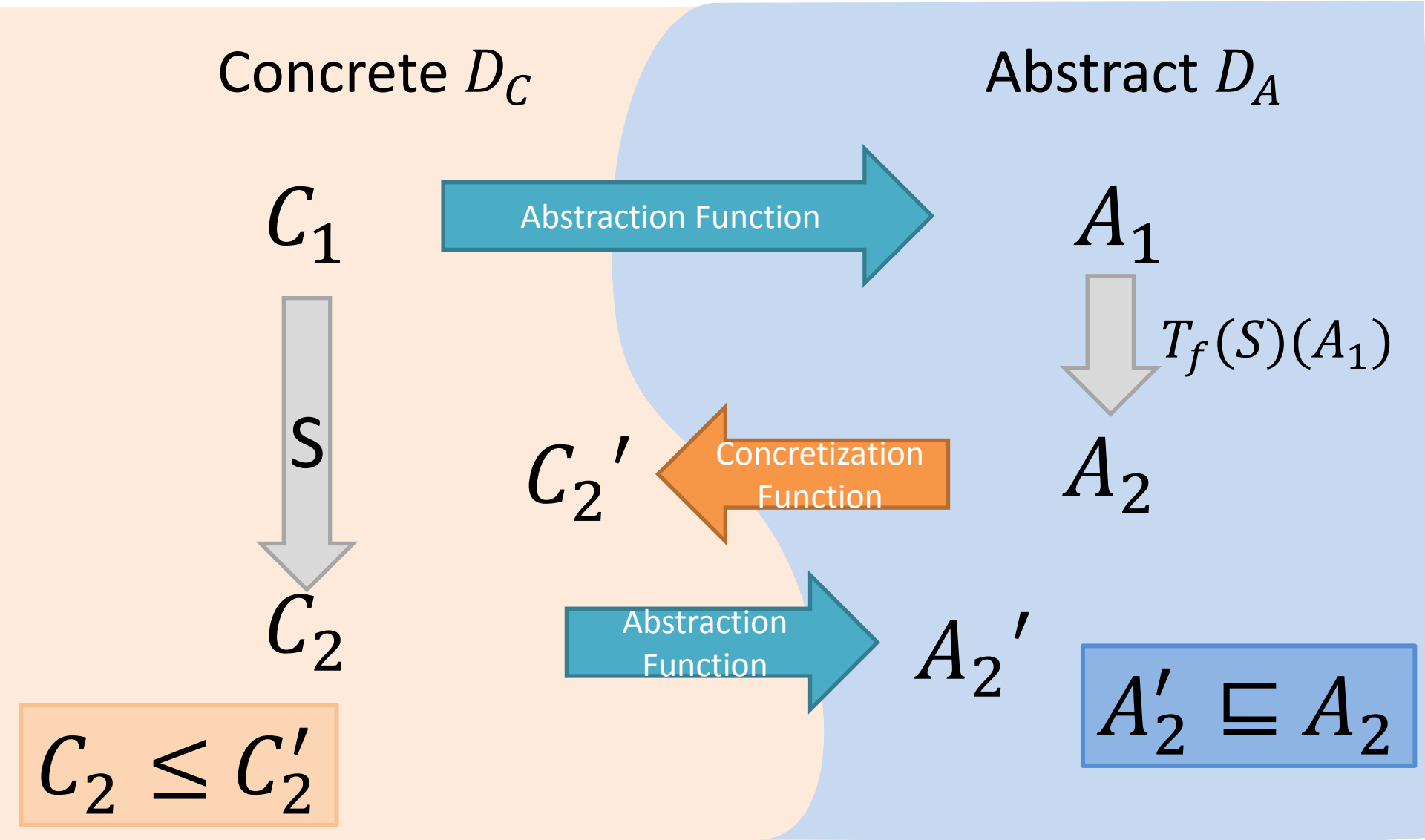
Abstraction/Concretization



Galois Connection



Galois Connection



Fix-point Algorithm

```
val Q = Queue(CFG.entry)
val Facts = CFG.nodes.map{n => n -> Bottom}.toMap


while(!Q.isEmpty) {
   val node = Q.dequeue()


    val res = Lattice.join(node.inEdges.map {
        case Edge(from, stmt, _) => Tf(stmt)(Facts(from))
    })


    if (res != Facts(node) || node == entry) {
        node.outEdges.foreach {
            case Edge(_, _, to) => Q.enqueue(to)
        }
    }
}
```

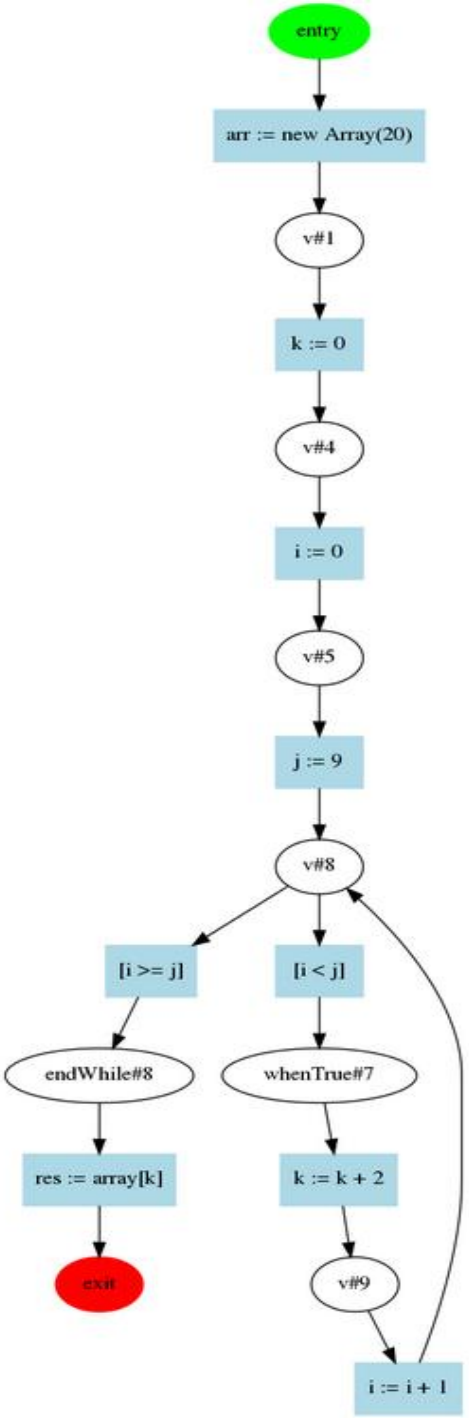
Example

```
var arr = new Array[Int](20)
var k   = 0
var i   = 0
var j   = 9

while(i < j) {
    k += 2
    i += 1

}

val res = arr(k)
```



Example: Ingredients

- Values Abstract Domain: Integer Intervals
- Partial order: interval inclusion
- Join:
  $\sqcup\,([l_1, u_1], .., [l_n, u_n]) = [\min(l_1, .., l_n), max(u_1, .., u_n)]$
- Abstraction function: $\alpha(v) = [v, v]$
- Concretization function: $\gamma([l, u]) = \{\, v \mid v \geq l \wedge v \leq u \,\}$

This is all per-value! For the actual abstract domain, we use a point-wise lattice and the corresponding point-wise operations.

Transfer Function

- r = a+b
  env(r) = [env(a).l + env(b).l, env(a).u + env(b).u]

- r = new Array(s)
  env(r_size) = [env(s).l, env(s).u] = env(s)
  env(r) = [0,0]

- r = arr(i)
  env(r) = env(arr)

- [a < b]
  env(a) = [env(a).l, min(env(a).u, env(b).u)]
  env(b) = [max(env(b).l, env(a).l), env(b).u]

Validity Conditions

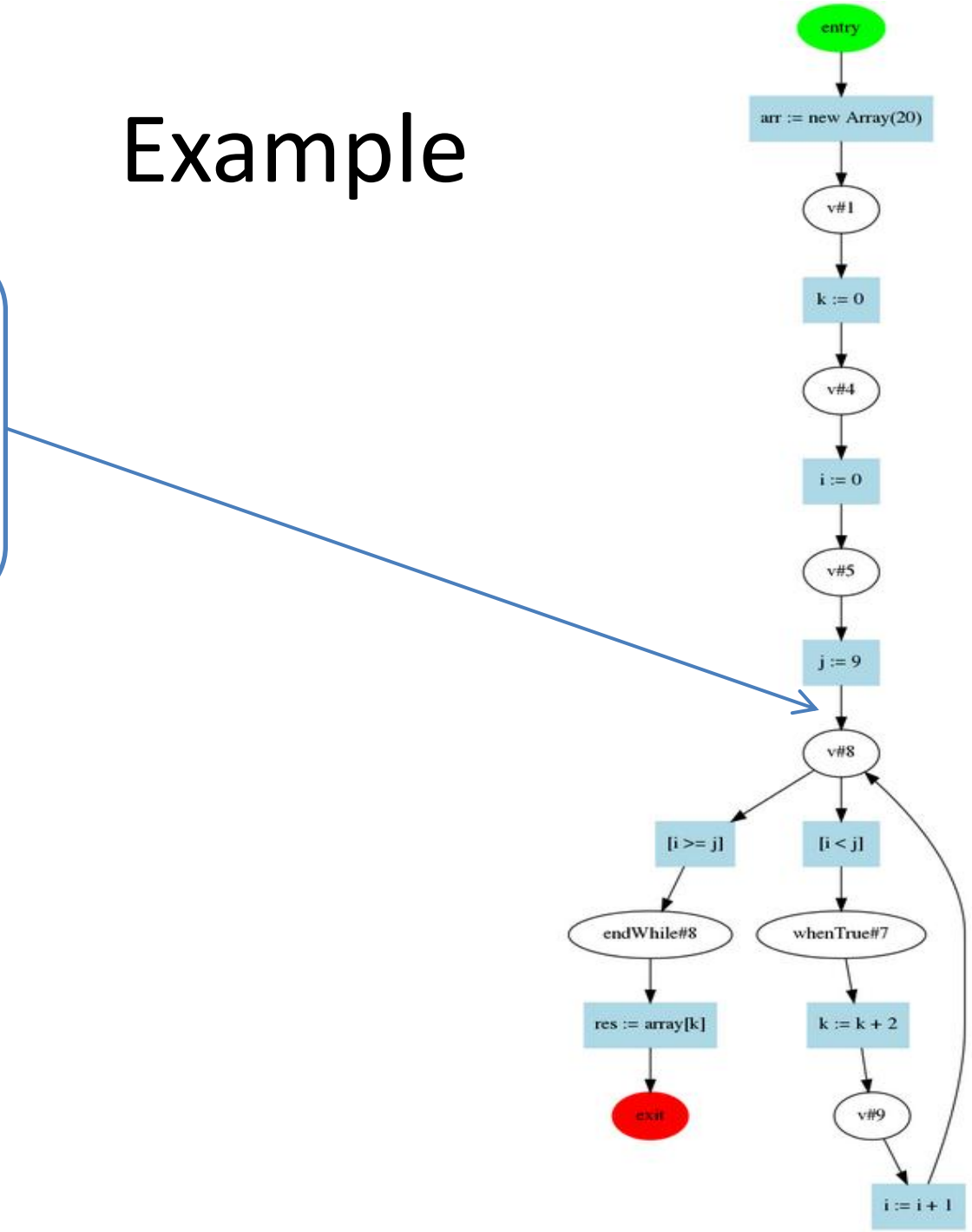
- new Array(s)
  $env(s) \sqsubseteq [0, +\infty]$

- r = a[i]
  $env(i) \sqsubseteq [0, env(a_{size}).l - 1]$

- r[i] = b
  $env(i) \sqsubseteq [0, env(r_{size}).l - 1]$

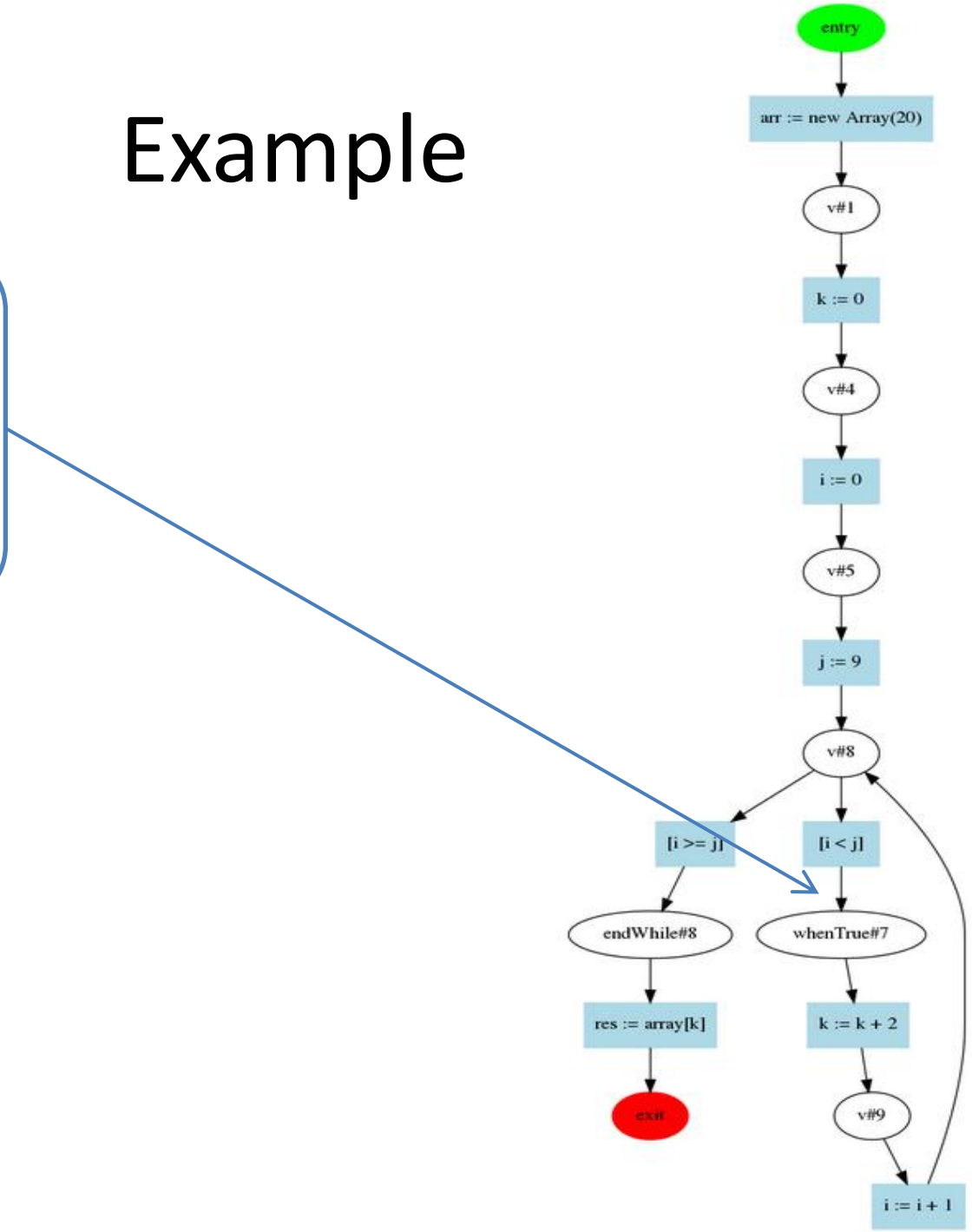
Example

$$
\begin{array}{ccc}
arr_{size} & \longmapsto & [20,20] \\
k & \longmapsto & [0,0] \\
i & \longmapsto & [0,0] \\
j & \longmapsto & [9,9]
\end{array}
$$



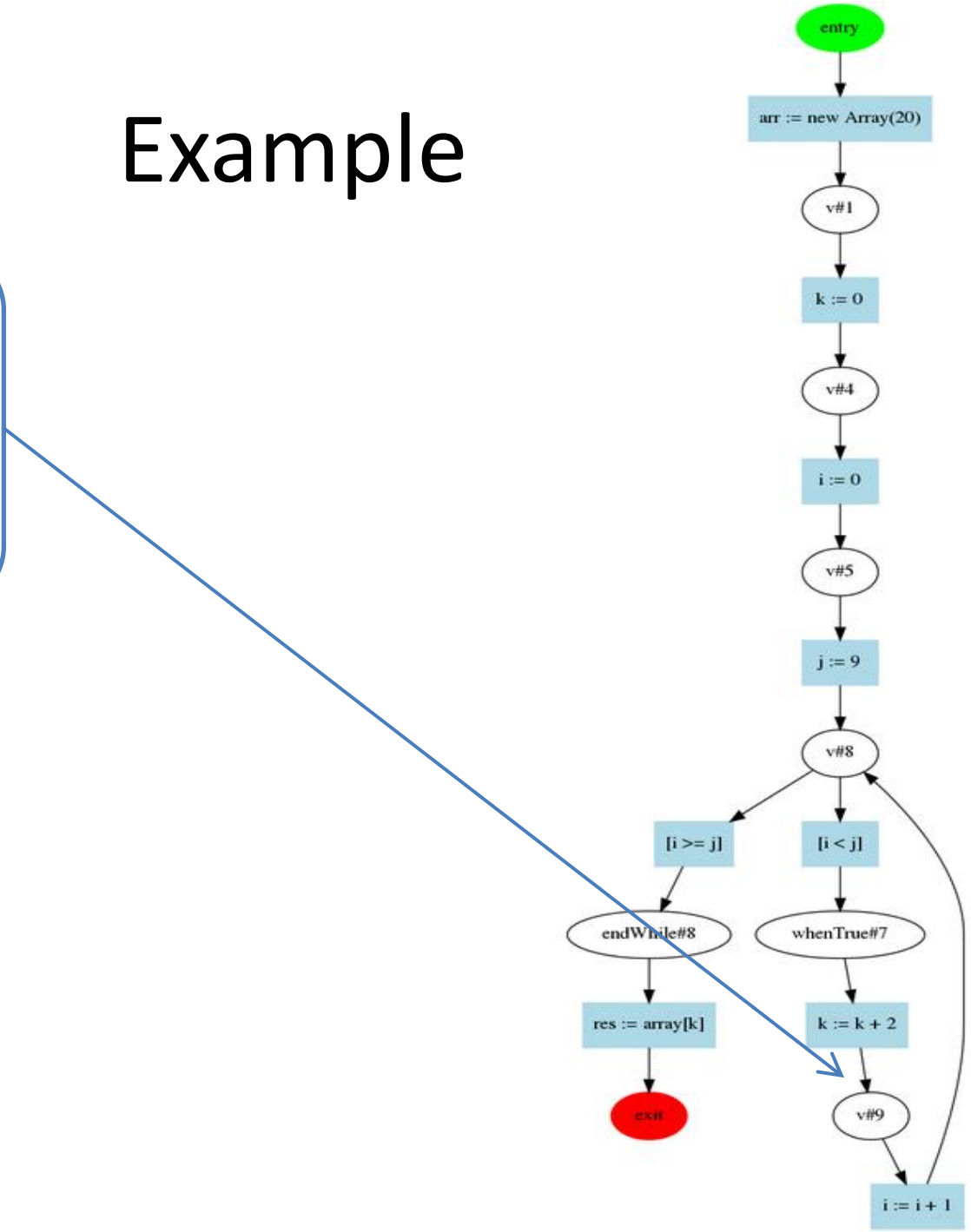
Example

$$
\begin{array}{ccc}
arr_{size} & \longmapsto & [20,20] \\
k & \longmapsto & [0,0] \\
i & \longmapsto & [0,0] \\
j & \longmapsto & [9,9]
\end{array}
$$



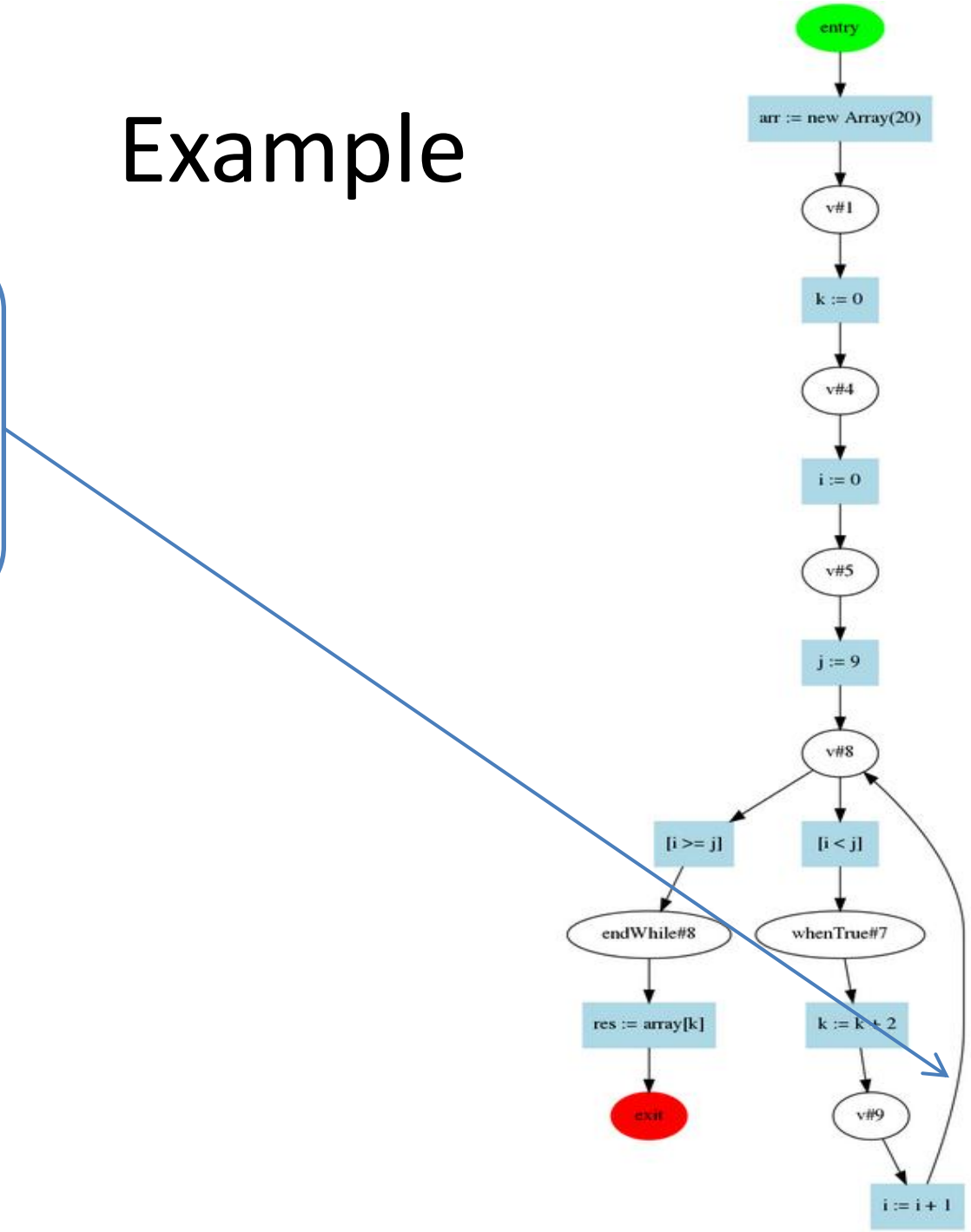
Example

$$
\begin{array}{ccc}
arr_{size} & \longmapsto & [20,20] \\
k & \longmapsto & [\mathbf{2},\mathbf{2}] \\
i & \longmapsto & [0,0] \\
j & \longmapsto & [9,9]
\end{array}
$$



Example

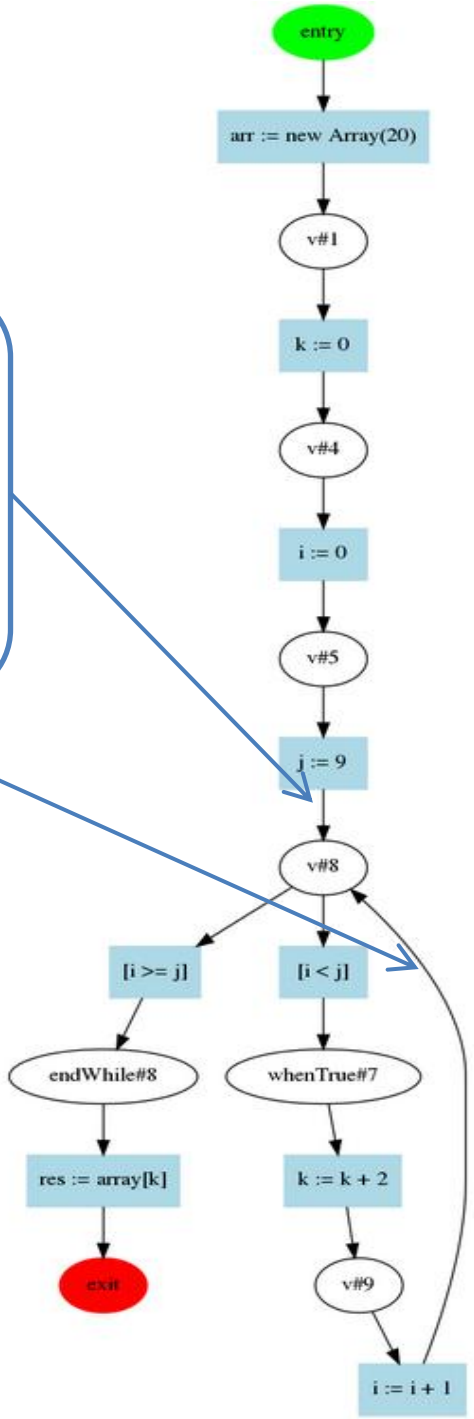
$$
\begin{array}{ccc}
arr_{size} & \longmapsto & [20,20] \\
k & \longmapsto & [2,2] \\
i & \longmapsto & [\mathbf{1},\mathbf{1}] \\
j & \longmapsto & [9,9]
\end{array}
$$



Example

$$\begin{array}{ccc} arr_{size} & \longmapsto & [20,20] \\ k & \longmapsto & [2,2] \\ i & \longmapsto & [1,1] \\ j & \longmapsto & [9,9] \end{array}$$

$$\begin{array}{ccc} arr_{size} & \longmapsto & [20,20] \\ k & \longmapsto & [0,0] \\ i & \longmapsto & [0,0] \\ j & \longmapsto & [9,9] \end{array}$$



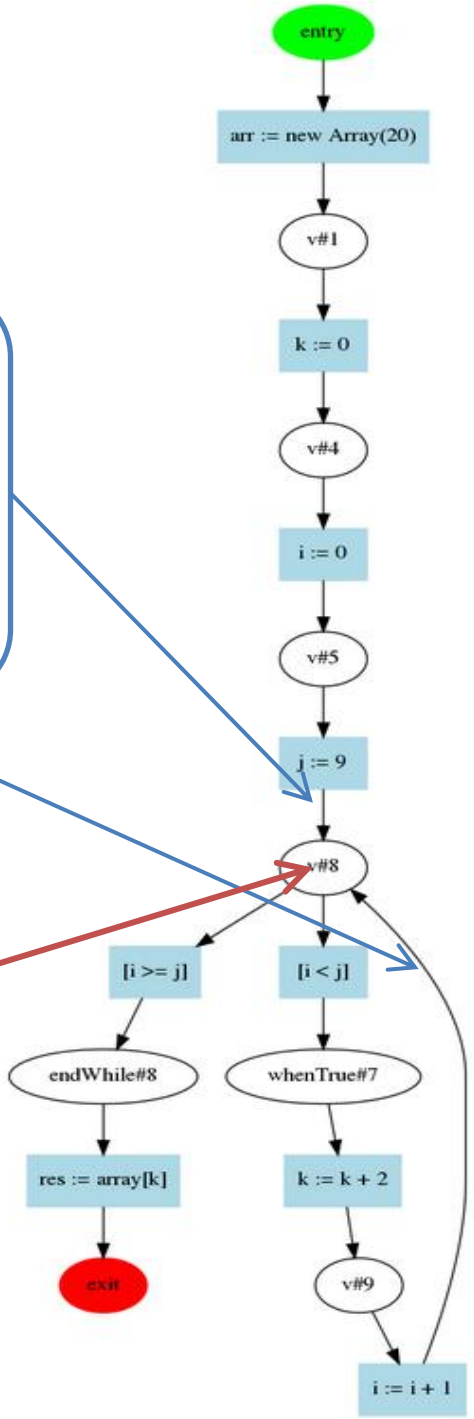
Example

$$\begin{array}{ccc} arr_{size} & \mapsto & [20,20] \\ k & \mapsto & [2,2] \\ i & \mapsto & [1,1] \\ j & \mapsto & [9,9] \end{array}$$

$$\begin{array}{ccc} arr_{size} & \mapsto & [20,20] \\ k & \mapsto & [0,0] \\ i & \mapsto & [0,0] \\ j & \mapsto & [9,9] \end{array}$$

$\sqcup$

$$\begin{array}{ccc} arr_{size} & \mapsto & [20,20] \\ k & \mapsto & [\mathbf{0},\mathbf{2}] \\ i & \mapsto & [\mathbf{0},\mathbf{1}] \\ j & \mapsto & [9,9] \end{array}$$

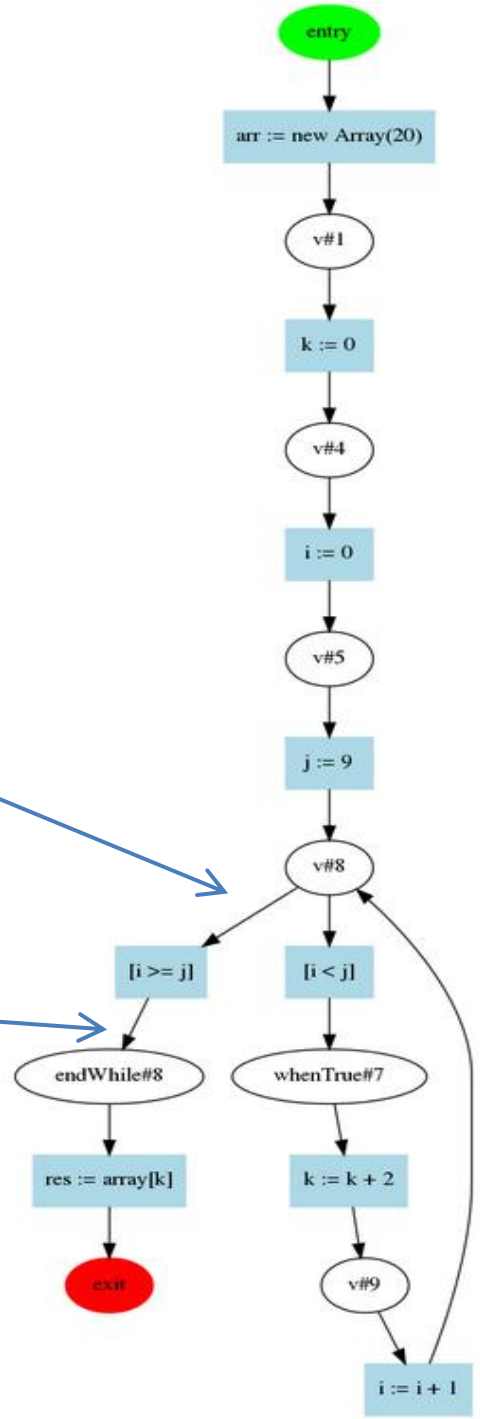


Example

After 1 iteration:

$$
\begin{aligned}
arr_{size} &\longmapsto [20,20] \\
k &\longmapsto [0,2] \\
i &\longmapsto [0,1] \\
j &\longmapsto [9,9]
\end{aligned}
$$

$$
\begin{aligned}
arr_{size} &\longmapsto [20,20] \\
k &\longmapsto [0,2] \\
i &\longmapsto \emptyset \\
j &\longmapsto \emptyset
\end{aligned}
$$

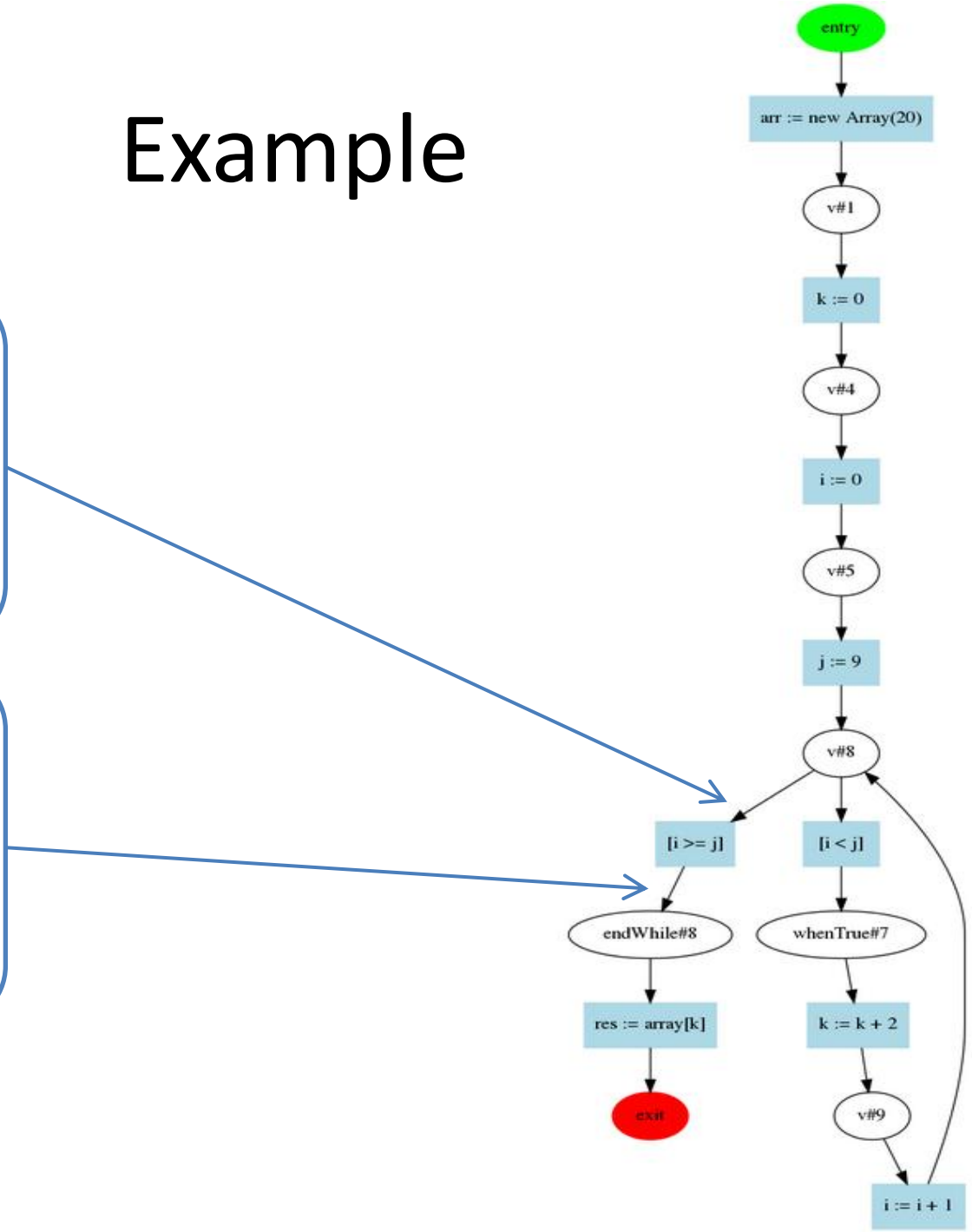


Example

After N iterations:

$$
\begin{aligned}
arr_{size} &\longmapsto [20,20] \\
k &\longmapsto [0,18] \\
i &\longmapsto [0,9] \\
j &\longmapsto [9,9]
\end{aligned}
$$

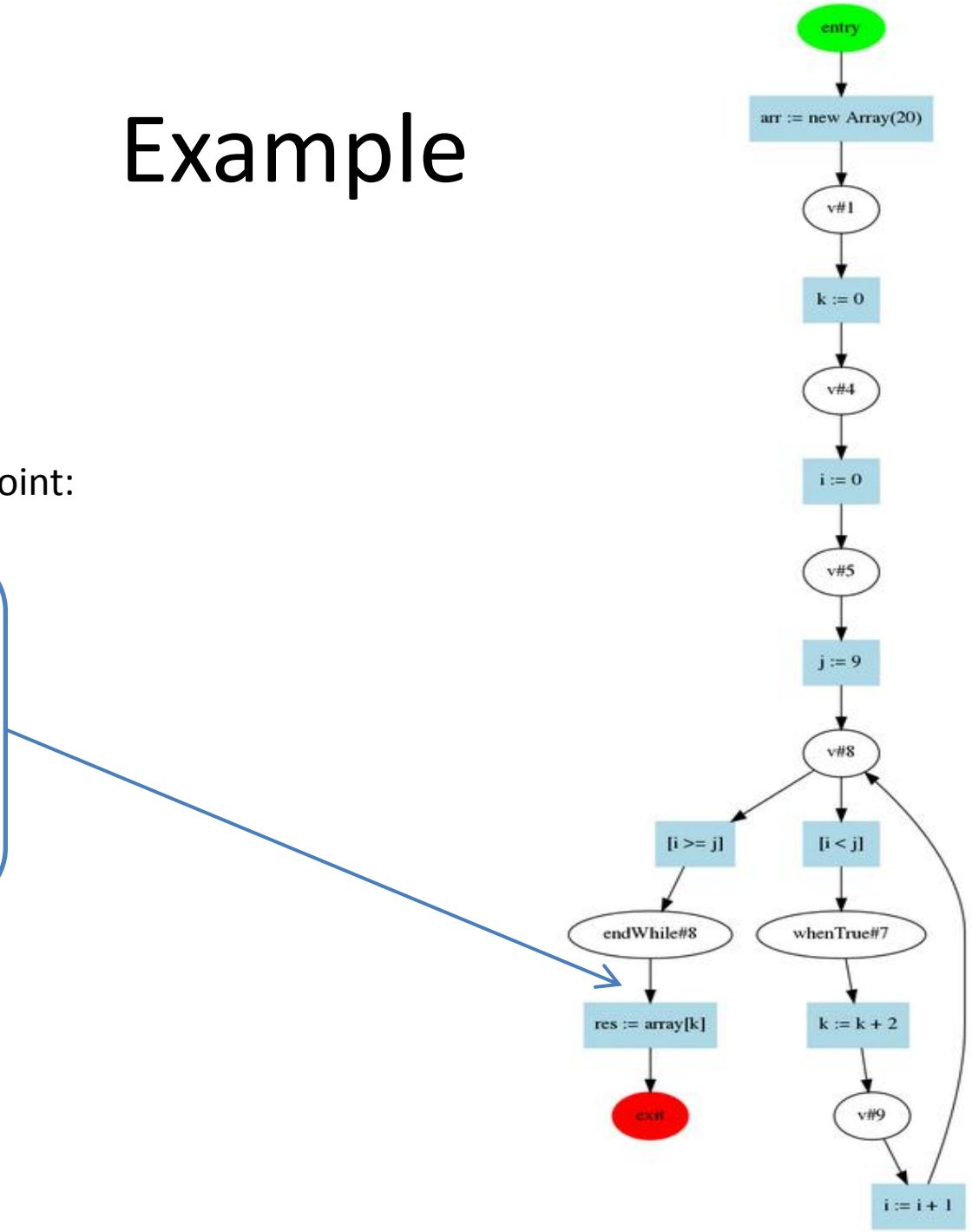
$$
\begin{aligned}
arr_{size} &\longmapsto [20,20] \\
k &\longmapsto [0,18] \\
i &\longmapsto [9,9] \\
j &\longmapsto [9,9]
\end{aligned}
$$



Example

Once we reached the fix-point:

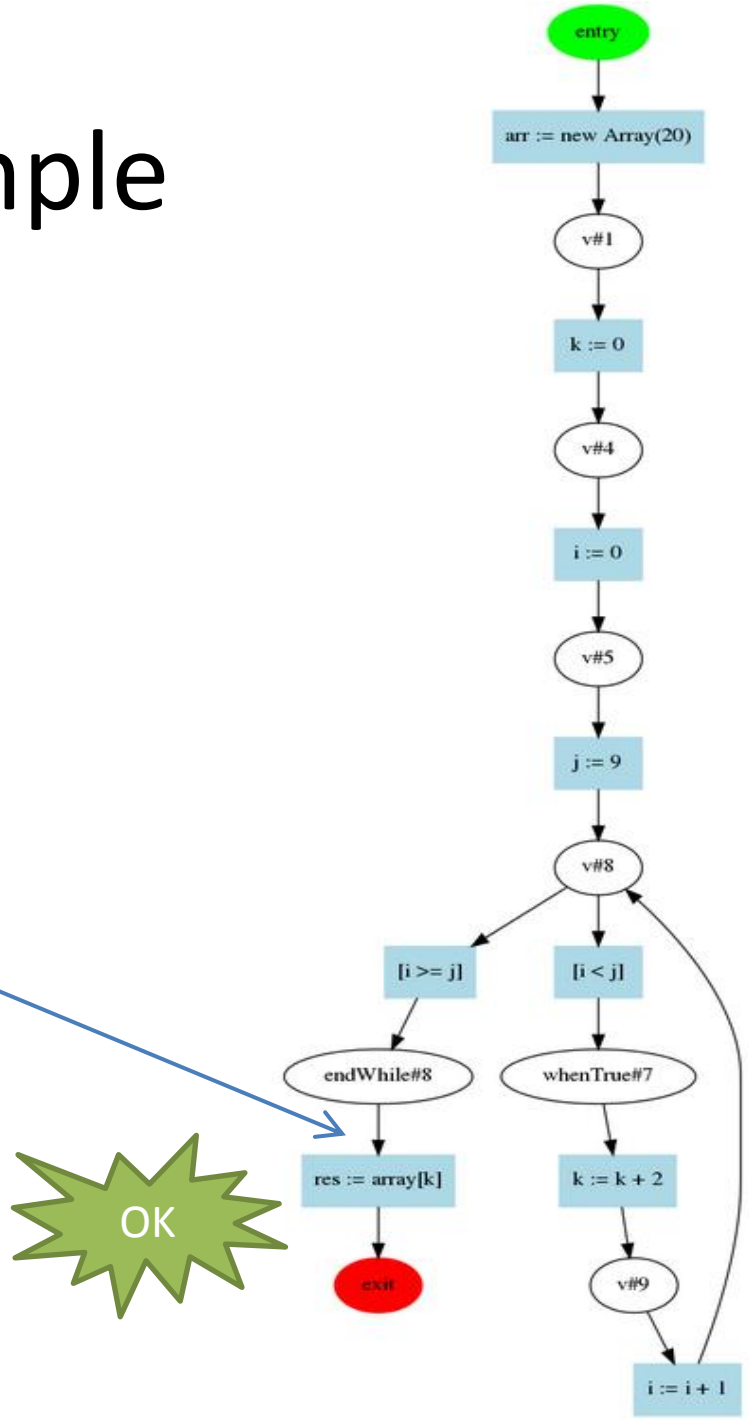
$$
\begin{array}{ccc}
arr_{size} & \longmapsto & [20,20] \\
k & \longmapsto & [0,18] \\
i & \longmapsto & [9,9] \\
j & \longmapsto & [9,9]
\end{array}
$$



Example

Once we reached the fix-point:

$$
\begin{array}{ccc}
arr_{size} & \longmapsto & [20,20] \\
k & \longmapsto & [0,18] \\
i & \longmapsto & [9,9] \\
j & \longmapsto & [9,9]
\end{array}
$$



Second Example: First Annoyance

```
var arr = new Array[Int](200000)
var k   = 0
var i   = 0
var j   = 100000

while(i < j) {
    k += 2
    i += 1

}

val res = arr(k)
```

Third Example: Big Problem

```
var arr = new Array[Int](20)
var k   = 0
var i   = 0
var j   = -1

while(i < j) {
    k += 2
    i += 1

}

val res = arr(k)
```

Solution: Widening Operator ∇

- $\nabla_* : \wp(D) \mapsto D$

  - ✓ $\forall v \in V . \nabla_*(V) \sqsupseteq v$
  - ✓ For every ascending chains $\{v_i\}_{i \geq 0}$ the chain defined as:
    $w_0 = v_0,\ w_i = \nabla_*(\{ \quad_i | 0 \leq j \leq i\})$
    is ascending and eventually stabilizes

Solution: Widening Operator ∇

At P:

1: env(i) = [0,1]

2: env(i) = [0,2]

…

5: env(i) = [0,10]

6: env(i) = [0,20]

…

10: env(i) = [0,+∞]

11: env(i) = [0,+∞]

Is it precise enough?

$$x \quad \longmapsto \quad [0,1]$$

…

y = x

…

z = y-x

arr(z) = 1

Is it precise enough?

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

…

$$x \longmapsto [0,1]$$
$$y \longmapsto [0,1]$$

y = x

…

z = y-x

arr(z) = 1

Is it precise enough?

…

y = x

…

z = y-x

arr(z) = 1

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

$$x \longmapsto [0,1]$$
$$y \longmapsto [0,1]$$

$$x \longmapsto [0,1]$$
$$y \longmapsto [0,1]$$
$$z \longmapsto [-1,1]$$

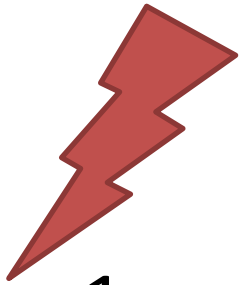
Is it precise enough?

…

y = x

…

z = y-x



arr(z) = 1

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

$$x \longmapsto [0,1]$$
$$y \longmapsto [0,1]$$

$$x \longmapsto [0,1]$$
$$y \longmapsto [0,1]$$
$$z \longmapsto [-1,1]$$

Another Abstract Domain: Octagons

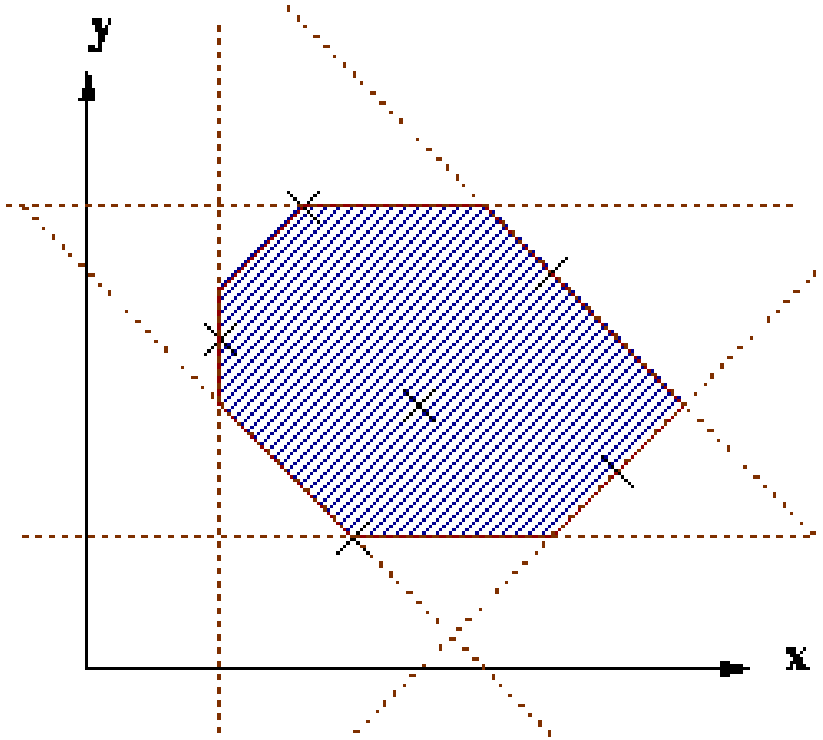
- Tracks relations between pairs of variables:

$$(x + y) \in [c1, c2]$$
$$(x - y) \in [c3, c4]$$
$$x \in [c5, c6]$$
$$y \in [c7\ c8]$$

Another Abstract Domain: Octagons

- Tracks relations between pairs of variables:

$(x + y) \in [c1, c2]$
$(x - y) \in [c3, c4]$
$x \in [c5, c6]$
$y \in [c7\ c8]$



Now, is it precise enough?

$$x \quad \longmapsto \quad [0,1]$$

…

y = x

…

z = y-x

arr(z) = 1

Now, is it precise enough?

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

$$\begin{aligned} x &\longmapsto [0,1] \\ y &\longmapsto [0,1] \\ y+x &\longmapsto [0,2] \\ y-x &\longmapsto [0,0] \end{aligned}$$

…

y = x

…

z = y-x

arr(z) = 1

Now, is it precise enough?

…

y = x

…

z = y-x

arr(z) = 1

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

$$\begin{aligned} x &\mapsto [0,1] \\ y &\mapsto [0,1] \\ y + x &\mapsto [0,2] \\ y - x &\mapsto [0,0] \end{aligned}$$

$$z \mapsto [0,0]$$

Now, is it precise enough?

…

y = x

…

z = y-x

arr(z) = 1

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

$$\begin{aligned} x &\longmapsto [0,1] \\ y &\longmapsto [0,1] \\ y + x &\longmapsto [0,2] \\ y - x &\longmapsto [0,0] \end{aligned}$$

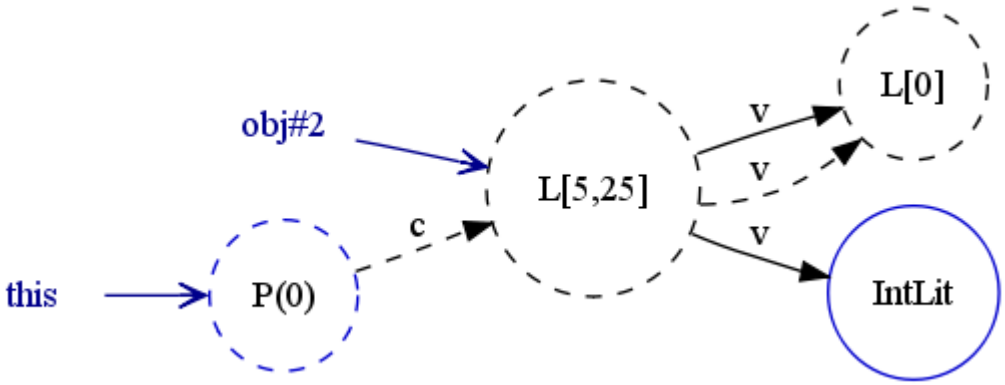
$$z \longmapsto [0,0]$$

Another Abstract Domain: Octagons

- Efficient encoding into matrices
- Can be seen as the shortest path between two edges
- Ensures a unique lub

What can we do with AI?

- Almost any kind of analysis!
- Used personally for:
  - Type reconstruction/checking for PHP
    - Abstract domain: Complex types
  - Interprocedural and compositional effect analysis for Scala
    - Abstract domain: Heap-modeling effect graphs



What we have seen

- Abstract Interpretation
  - Analysis Framework based on approximation that ensures:
    - Soundness
    - Termination
  - Parametric in almost every aspects
- Two abstract domains for tracking numerical values:
  - Intervals: very simple, fast, but imprecise
  - Octagons: refined, more precise, fast implementation, used in Astree

Thank you!