

# CS738: Advanced Compiler Optimizations

## Foundations of Data Flow Analysis

Amey Karkare

karkare@cse.iitk.ac.in

<http://www.cse.iitk.ac.in/~karkare/cs738>

Department of CSE, IIT Kanpur



# Agenda

- ▶ Poset, Lattice, and Data Flow Frameworks: Review
- ▶ Connecting Tarski Lemma with Data Flow Analysis
- ▶ Solutions of Data Flow Analysis constraints

# Knaster-Tarski Fixed Point Theorem

- ▶ Let  $f$  be a monotonic function on a complete lattice  $(S, \wedge, \vee)$ . Define
  - ▶  $\text{red}(f) = \{v \mid v \in S, f(v) \leq v\}$ , pre fix-points
  - ▶  $\text{ext}(f) = \{v \mid v \in S, f(v) \geq v\}$ , post fix-points
  - ▶  $\text{fix}(f) = \{v \mid v \in S, f(v) = v\}$ , fix-points

Then,

- ▶  $\bigwedge \text{red}(f) \in \text{fix}(f)$ . Further,  $\bigwedge \text{red}(f) = \bigwedge \text{fix}(f)$
- ▶  $\bigvee \text{ext}(f) \in \text{fix}(f)$ . Further,  $\bigvee \text{ext}(f) = \bigwedge \text{fix}(f)$
- ▶  $\text{fix}(f)$  is a complete lattice

# Application of Fixed Point Theorem

- ▶  $f : S \rightarrow S$  is a **monotonic** function
- ▶  $(S, \wedge)$  is a **finite height** semilattice
- ▶  $\top$  is the top element of  $(S, \wedge)$
- ▶ Notation:  $f^0(x) = x, f^{i+1}(x) = f(f^i(x)), \forall i \geq 0$
- ▶ The greatest fixed point of  $f$  is

$$f^k(\top), \text{ where } f^{k+1}(\top) = f^k(\top)$$

# Fixed Point Algorithm

```
// monotonic function f on a meet semilattice  
x :=  $\top$ ;  
while (x  $\neq$  f(x)) x := f(x);  
return x;
```

## Resemblance to Iterative Algorithm (Forward)

$$\text{OUT}[Entry] = \text{Info} \circ_{Entry} i$$

## Resemblance to Iterative Algorithm (Forward)

```
OUT[Entry] = InfoEntry;  
for (other blocks  $B$ ) OUT[ $B$ ] = T;
```

## Resemblance to Iterative Algorithm (Forward)

```
OUT[Entry] = InfoEntry;  
for (other blocks  $B$ ) OUT[ $B$ ] =  $\top$ ;  
while (changes to any OUT) {
```



## Resemblance to Iterative Algorithm (Forward)

```
OUT[Entry] = InfoEntry;  
for (other blocks  $B$ ) OUT[ $B$ ] =  $\top$ ;  
while (changes to any OUT) {  
    for (each block  $B$ ) {
```

## Resemblance to Iterative Algorithm (Forward)

```
OUT[Entry] = InfoEntry;  
for (other blocks  $B$ ) OUT[ $B$ ] =  $\top$ ;  
while (changes to any OUT) {  
    for (each block  $B$ ) {  
         $IN(B) = \bigwedge_{P \in \text{PRED}(B)} OUT(P)$ ;
```

## Resemblance to Iterative Algorithm (Forward)

```
OUT[Entry] = InfoEntry;  
for (other blocks  $B$ ) OUT[ $B$ ] =  $\top$ ;  
while (changes to any OUT) {  
    for (each block  $B$ ) {  
        IN( $B$ ) =  $\bigwedge_{P \in \text{PRED}(B)} \text{OUT}(P)$ ;  
        OUT( $B$ ) =  $f_B(\text{IN}(B))$ ;  
    }  
}
```

# Iterative Algorithm

►  $f_B(X) = X - \text{KILL}(B) \cup \text{GEN}(B)$

# Iterative Algorithm

- ▶  $f_B(X) = X - \text{KILL}(B) \cup \text{GEN}(B)$
- ▶ Backward:

# Iterative Algorithm

- ▶  $f_B(X) = X - \text{KILL}(B) \cup \text{GEN}(B)$
- ▶ Backward:
  - ▶ Swap IN and OUT everywhere

# Iterative Algorithm

- ▶  $f_B(X) = X - \text{KILL}(B) \cup \text{GEN}(B)$
- ▶ Backward:
  - ▶ Swap IN and OUT everywhere
  - ▶ Replace *Entry* by *Exit*

# Iterative Algorithm

- ▶  $f_B(X) = X - \text{KILL}(B) \cup \text{GEN}(B)$
- ▶ Backward:
  - ▶ Swap IN and OUT everywhere
  - ▶ Replace *Entry* by *Exit*
  - ▶ Replace predecessors by successors



# Iterative Algorithm

- ▶  $f_B(X) = X - \text{KILL}(B) \cup \text{GEN}(B)$
- ▶ Backward:
  - ▶ Swap IN and OUT everywhere
  - ▶ Replace *Entry* by *Exit*
  - ▶ Replace predecessors by successors
- ▶ In other words: just “invert” the flow graph!!

# Acknowledgement

Rest of the slides based on the material at

<http://infolab.stanford.edu/~ullman/dragon/w06/w06.html>

# Solutions

- ▶ **IDEAL solution** = meet over all executable paths from entry to a point (ignore unrealizable paths)

# Solutions

- ▶ **IDEAL solution** = meet over all executable paths from entry to a point (ignore unrealizable paths)
- ▶ **MOP** = meet over all paths from entry to a given point, of the transfer function along that path applied to  $\text{Info}_{\text{Entry}}$ .

# Solutions

- ▶ **IDEAL solution** = meet over all executable paths from entry to a point (ignore unrealizable paths)
- ▶ **MOP** = meet over all paths from entry to a given point, of the transfer function along that path applied to  $\text{Info}_{\text{Entry}}$ .
- ▶ **MFP** (*maximal fixedpoint*) = result of iterative algorithm.

# Maximum Fixedpoint

- **Fixedpoint** = solution to the equations used in the iteration:

$$\text{IN}(B) = \bigwedge_{P \in \text{PRED}(B)} \text{OUT}(P)$$

$$\text{OUT}(B) = f_B(\text{IN}(B))$$

# Maximum Fixedpoint

- ▶ **Fixedpoint** = solution to the equations used in the iteration:

$$\text{IN}(B) = \bigwedge_{P \in \text{PRED}(B)} \text{OUT}(P)$$

$$\text{OUT}(B) = f_B(\text{IN}(B))$$

- ▶ **Maximum Fixedpoint** = any other solution is  $\leq$  the result if the iterative algorithm (MFP)

# Maximum Fixedpoint

- ▶ **Fixedpoint** = solution to the equations used in the iteration:

$$\text{IN}(B) = \bigwedge_{P \in \text{PRED}(B)} \text{OUT}(P)$$

$$\text{OUT}(B) = f_B(\text{IN}(B))$$

- ▶ **Maximum Fixedpoint** = any other solution is  $\leq$  the result if the iterative algorithm (MFP)
- ▶  $\leq$ : carries less information.



# MOP and IDEAL

- ▶ All solutions are really meets of the result of starting with  $\text{Info}_{\text{Entry}}$  and following some set of paths to the point in question.

# MOP and IDEAL

- ▶ All solutions are really meets of the result of starting with  $\text{Info}_{\text{Entry}}$  and following some set of paths to the point in question.
- ▶ If we don't include **at least the IDEAL paths**, we have an error.

# MOP and IDEAL

- ▶ All solutions are really meets of the result of starting with  $\text{Info}_{\text{Entry}}$  and following some set of paths to the point in question.
- ▶ If we don't include **at least the IDEAL paths**, we have an error.
- ▶ But try not to include too many more.

# MOP and IDEAL

- ▶ All solutions are really meets of the result of starting with  $\text{Info}_{\text{Entry}}$  and following some set of paths to the point in question.
- ▶ If we don't include **at least the IDEAL paths**, we have an error.
- ▶ But try not to include too many more.
- ▶ Less “ignorance,” but we “know too much.”

# MOP Versus IDEAL

- ▶ Any solution that is  $\leq$  IDEAL accounts for all executable paths (and maybe more paths)

# MOP Versus IDEAL

- ▶ Any solution that is  $\leq$  IDEAL accounts for all executable paths (and maybe more paths)
  - ▶ and is therefore conservative (safe)

# MOP Versus IDEAL

- ▶ Any solution that is  $\leq$  IDEAL accounts for all executable paths (and maybe more paths)
  - ▶ and is therefore conservative (safe)
  - ▶ even if not accurate.

# MFP vs MOP

- ▶ If  $MFP \leq MOP$ ?



# MFP vs MOP

- ▶ If  $\text{MFP} \leq \text{MOP}$ ?
  - ▶ If so, then  $\text{MFP} \leq \text{MOP} \leq \text{IDEAL}$ , therefore MFP is safe.

# MFP vs MOP

- ▶ If  $\text{MFP} \leq \text{MOP}$ ?
  - ▶ If so, then  $\text{MFP} \leq \text{MOP} \leq \text{IDEAL}$ , therefore MFP is safe.
- ▶ Yes, but ...

# MFP vs MOP

- ▶ If  $MFP \leq MOP$ ?
  - ▶ If so, then  $MFP \leq MOP \leq IDEAL$ , therefore MFP is safe.
- ▶ Yes, but ...
- ▶ Requires two assumptions about the framework:

# MFP vs MOP

- ▶ If  $MFP \leq MOP$ ?
  - ▶ If so, then  $MFP \leq MOP \leq IDEAL$ , therefore MFP is safe.
- ▶ Yes, but . . .
- ▶ Requires two assumptions about the framework:
  - ▶ “Monotonicity.”

# MFP vs MOP

- ▶ If  $\text{MFP} \leq \text{MOP}$ ?
  - ▶ If so, then  $\text{MFP} \leq \text{MOP} \leq \text{IDEAL}$ , therefore MFP is safe.
- ▶ Yes, but . . .
- ▶ Requires two assumptions about the framework:
  - ▶ “Monotonicity.”
  - ▶ Finite height

# MFP vs MOP

- ▶ If  $\text{MFP} \leq \text{MOP}$ ?
  - ▶ If so, then  $\text{MFP} \leq \text{MOP} \leq \text{IDEAL}$ , therefore MFP is safe.
- ▶ Yes, but ...
- ▶ Requires two assumptions about the framework:
  - ▶ “Monotonicity.”
  - ▶ Finite height  
no infinite chains  $\dots < x_2 < x_1 < x < \dots$

# MFP vs MOP

- ▶ **Intuition:** If we computed the MOP directly, we would compose functions along all paths, then take a big meet.

# MFP vs MOP

- ▶ **Intuition:** If we computed the MOP directly, we would compose functions along all paths, then take a big meet.
- ▶ But the MFP (iterative algorithm) alternates compositions and meets arbitrarily.



# Good News

- ▶ The frameworks we've studied so far are all monotone.

# Good News

- ▶ The frameworks we've studied so far are all monotone.
  - ▶ Easy proof for functions in Gen-Kill form.

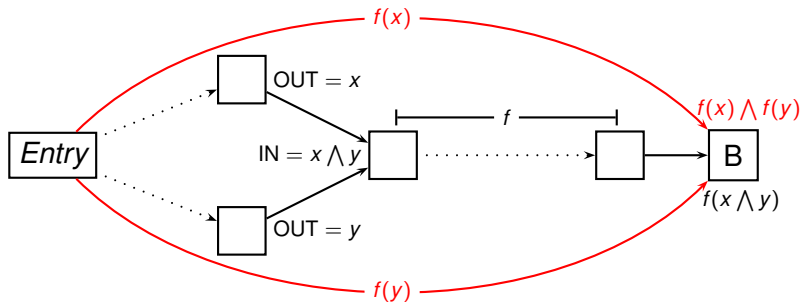
# Good News

- ▶ The frameworks we've studied so far are all monotone.
  - ▶ Easy proof for functions in Gen-Kill form.
- ▶ And they have finite height.

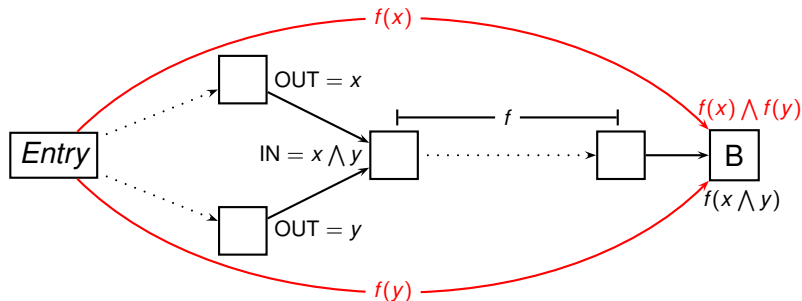
# Good News

- ▶ The frameworks we've studied so far are all monotone.
  - ▶ Easy proof for functions in Gen-Kill form.
- ▶ And they have finite height.
  - ▶ Only a finite number of defs, variables, etc. in any program.

## Two Paths to $B$ that Meet Early

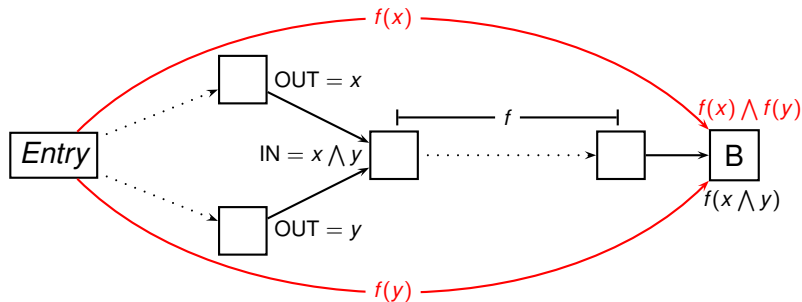


## Two Paths to $B$ that Meet Early



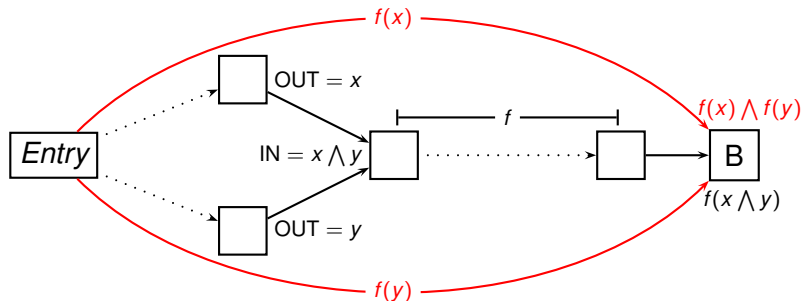
- MOP considers paths independently and combines at the last possible moment.

## Two Paths to $B$ that Meet Early



- ▶ MOP considers paths independently and combines at the last possible moment.
- ▶ In MFP, Values  $x$  and  $y$  get combined too soon.

## Two Paths to $B$ that Meet Early



- ▶ **MOP considers paths independently and combines at the last possible moment.**
- ▶ In MFP, Values  $x$  and  $y$  get combined too soon.
- ▶ Since  $f(x \wedge y) \leq f(x) \wedge f(y)$ , it is as we added non-existent paths.



# Distributive Frameworks

- ▶ Distributivity:

$$f(x \bigwedge y) = f(x) \bigwedge f(y)$$

# Distributive Frameworks

- ▶ Distributivity:

$$f(x \bigwedge y) = f(x) \bigwedge f(y)$$

- ▶ Stronger than Monotonicity

# Distributive Frameworks

- ▶ Distributivity:

$$f(x \bigwedge y) = f(x) \bigwedge f(y)$$

- ▶ Stronger than Monotonicity
  - ▶ Distributivity  $\Rightarrow$  Monotonicity

# Distributive Frameworks

- ▶ Distributivity:

$$f(x \bigwedge y) = f(x) \bigwedge f(y)$$

- ▶ Stronger than Monotonicity
  - ▶ Distributivity  $\Rightarrow$  Monotonicity
  - ▶ But the reverse is not true

# Even More Good News!

- ▶ The 4 example frameworks are distributive.

# Even More Good News!

- ▶ The 4 example frameworks are distributive.
- ▶ If a framework is distributive, then combining paths early doesn't hurt.

# Even More Good News!

- ▶ The 4 example frameworks are distributive.
- ▶ If a framework is distributive, then combining paths early doesn't hurt.
  - ▶  $MOP = MFP$ .

# Even More Good News!

- ▶ The 4 example frameworks are distributive.
- ▶ If a framework is distributive, then combining paths early doesn't hurt.
  - ▶  $MOP = MFP$ .
  - ▶ That is, the iterative algorithm computes a solution that takes into account all and only the physical paths.