

# **CSC2/455 Software Analysis and Improvement**

## **Foundations of Data Flow Analysis - II**

---

Sreepathi Pai

February 17, 2020

URCS

# Outline

Review

Proofs

Constant Propagation

Postscript

# Outline

Review

Proofs

Constant Propagation

Postscript

# Part I of Foundations

- Methods to solve dataflow analysis equations
  - IDEAL
  - Meet over paths (MOP)
  - Maximum Fixed Point (MFP)
  - IDEAL  $\subseteq$  MOP  $\subseteq$  MFP
- (Semi)lattice-based framework
  - $(D, V, \wedge, F)$ , dataflow analysis
  - $(V, \wedge)$ , meet semilattice
  - $(V, \leq)$ , partial order, where  $x \leq y$  iff  $x \wedge y = x$
  - Monotone framework
- Greatest Lower Bound
  - If  $z = x \wedge y$ , then  $z \leq x$  and  $z \leq y$

# Monotone Framework

- A given  $(D, V, \wedge, F)$  is monotone if for all  $x, y \in V$ , and  $f \in F$ :
  - $x \leq y \rightarrow f(x) \leq f(y)$
  - equivalently,  $x \leq y \rightarrow f(x \wedge y) \leq f(x) \wedge f(y)$
  - The proof of equivalence is in the textbook.
- In addition, the framework is *distributive* if:
  - $f(x \wedge y) = f(x) \wedge f(y)$
- Note that these properties do not necessarily arise automatically,  $F$  must be designed to have these properties
  - And proofs must be written to show that  $F$  does.
  - We'll see this for a complicated example today.

# General Iterative Algorithm

```
forwards(IN, OUT, meet, top, v_entry, f_transfer)
  OUT[entry] = v_entry

  for each basic block B except ENTRY:
    OUT[B] = top

  do {
    for each basic block B except ENTRY:
      # this calculates the meet over predecessors, /p OUT[p]
      IN[B] = reduce(meet, [OUT[p] for p in B.predecessors])
      OUT[B] = f_transfer(IN[B])

  } while(some OUT changes value)
```

- Does this calculate the solution to the dataflow problem?
- Does this algorithm terminate?
- Does this algorithm calculate the *maximum* fixed point – i.e. the most precise solution admissible?

# This class

- Proofs that answer these three questions
- Relationships between IDEAL, MOP and MFP in terms of the framework
- Examples of:
  - a non-distributive framework (from Dragon 9.4, Constant Propagation)
  - lattices containing infinite values
  - possibly some proof writing exercises (from Dragon 9.3)

# Outline

Review

Proofs

Constant Propagation

Postscript

# Proof #1

The iterative algorithm computes the solution to the dataflow problem.

- The iterative algorithm performs an unbounded number of iterations as long as IN and OUT change
- *When it terminates*, IN and OUT have not changed for an iteration
- The values of IN and OUT therefore satisfy the equations
  - Hence they are solutions of the dataflow problem

## Proof #2

The iterative algorithm terminates (i.e. converges to a fix point).

- When we apply the  $\wedge$  operator, we obtain the glb
  - i.e.  $z = x \wedge y$  and  $z \leq x$  and  $z \leq y$
- Since the framework is monotone:
  - $f(x) \leq f(y)$  if  $x \leq y$
  - i.e. OUT values are no greater than the IN values
- At each step, these values decrease or remain the same
  - When they all remain the same, we terminate
- If values decrease, recall the lattice has finite height
  - Implies a finite number of steps before we reach  $\perp$
  - $x \wedge \perp = \perp$  and  $f(\perp) = \perp$  (i.e once a value becomes  $\perp$ , it no longer changes)
  - We terminate in this case as well

## Proof #3

The fixed point solution computed by the iterative algorithm is the *maximum* fixed point.

**Proof** By induction, for forward analyses

(*BASIS*) After the first iteration, values of  $IN[B]$  and  $OUT[B]$  are  $\leq$  their initial values.

- At initialization,  $OUT[B]$  is  $\top$  for all blocks  $B$  except ENTRY
- After the first iteration, in a monotone framework, all values will be  $\leq$  those at initialization by definitions of the  $\wedge$  and transfer functions

## Proof #3: Inductive step

Assume that:

- $\text{IN}[B]^k \leq \text{IN}[B]^{k-1}$
- $\text{OUT}[B]^k \leq \text{OUT}[B]^{k-1}$

Show that:

- $\text{IN}[B]^{k+1} \leq \text{IN}[B]^k$
- $\text{OUT}[B]^{k+1} \leq \text{OUT}[B]^k$

## Proof #3: Continued

- To obtain  $\text{IN}[B]$  we must apply  $\wedge$  to all  $\text{OUT}[P]$ 
  - $P$  is a predecessor of  $B$
  - This implies  $\text{IN}[B] \leq \text{OUT}[P]$  ( $\wedge$  yields glb)
  - From our inductive hypothesis,  $\text{OUT}[P]^k \leq \text{OUT}[P]^{k-1}$
  - applying  $\wedge$  on both sides over all  $P$ ,  $\text{IN}[B]^{k+1} \leq \text{IN}[B]^k$
- Now,  $\text{OUT}[B] = f(\text{IN}[B])$ 
  - In the monotone framework,  $f(x) \leq f(y)$  when  $x \leq y$
  - We have shown  $\text{IN}[B]^{k+1} \leq \text{IN}[B]^k$
  - Therefore, after applying  $f$  to both sides, by monotonicity, we have  $\text{OUT}[B]^{k+1} \leq \text{OUT}[B]^k$

## Properties of the IDEAL solution

- Any solution greater than IDEAL is incorrect (or unsafe)
- Any solution less than or equal to IDEAL is conservative<sup>1</sup>, or safe.

To see why, consider IDEAL solution  $x = p_1 \wedge p_2 \wedge \dots \wedge p_n$ :

- How can we obtain a value  $z = p_1 \wedge \dots$  *greater than*  $x$ ?
- How can we obtain a value  $y = p_1 \wedge \dots$  *less than*  $x$ ?

(recall the relationship between the results of the meet operator and its operands)

---

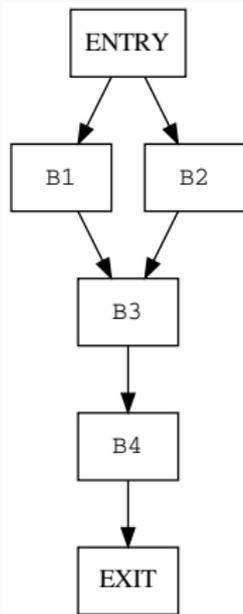
<sup>1</sup>In the English sense

## Relationship between IDEAL and MOP

- MOP considers a superset of all executable paths
  - MOP solution  $y = p_1 \wedge p_2 \wedge \dots \wedge p_n \wedge p_{n+1} \dots$
- What is the relationship between MOP ( $y$ ) and IDEAL ( $z$ )?

# Relationship between MOP and MFP

- $MOP[B_4] = ((f_{B_3} \circ f_{B_1}) \wedge (f_{B_3} \circ f_{B_2}))(v_{entry})$
- $IN[B_4] = f_{B_3}(f_{B_1}(v_{entry}) \wedge f_{B_2}(v_{entry}))$



## In a distributive framework, MOP = MFP

- $\text{MOP}[B_4] = ((f_{B_3} \circ f_{B_1}) \wedge (f_{B_3} \circ f_{B_2}))(v_{\text{entry}})$
- $\text{IN}[B_4] = f_{B_3}(f_{B_1}(v_{\text{entry}}) \wedge f_{B_2}(v_{\text{entry}}))$

If  $f(x \wedge y) = f(x) \wedge f(y)$  (i.e. distributive):

- $\text{IN}[B_4] = f_{B_3}(f_{B_1}(v_{\text{entry}})) \wedge f_{B_3}(f_{B_2}(v_{\text{entry}}))$
- If the framework is distributive, then MOP solution = MFP solution
  - Otherwise by monotonicity  $\text{MFP} \leq \text{MOP}$
- In either case,
  - $\text{MFP} \leq \text{MOP} \leq \text{IDEAL}$
  - So all methods produce “safe” solutions

# Outline

Review

Proofs

Constant Propagation

Postscript

## Analyses so far

- Live variable analysis
- Available Expressions
- Reaching Definitions
- These are all distributive (implies monotonicity)
- Their lattices contain a finite number of values
- Their lattices have finite height

# Constant Propagation

- Does this variable have a constant value at this point in the program?
  - Used to perform constant folding (i.e. evaluate constant expressions at compile time)
- Data flow analysis framework
  - Direction?
  - Values?
  - Meet operator?
  - Transfer function?

# Constant Propagation

- Direction: Forward
- Values:
  - UNDEF: variable is undefined so far
  - $c$ : variable is constant value  $c$
  - NAC: variable is not a constant
- Meet operators and transfer functions are slightly more complicated.

# Meet for Constant Propagation

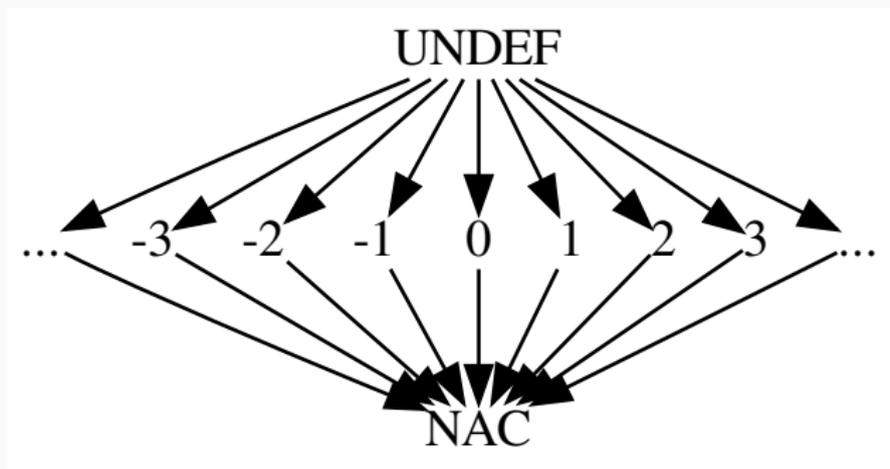
- $\text{UNDEF} \wedge v = ?$
- $\text{NAC} \wedge v = ?$
- $c \wedge c = ?$
- $c_1 \wedge c_2 = ? (c_1 \neq c_2)$

# Meet for Constant Propagation

- $\text{UNDEF} \wedge v = v$ 
  - $\text{UNDEF}$  is  $\top$
- $\text{NAC} \wedge v = \text{NAC}$ 
  - $\text{NAC}$  is  $\perp$
- $c \wedge c = c$
- $c_1 \wedge c_2 = \text{NAC}$

What does the lattice for constant propagation look like?

## The lattice for constant propagation



# The Transfer Function

- $\text{OUT}[s] = f(\text{IN}[s])$  for a statement  $s$ 
  - Slightly easier to understand if we use statements instead of basic blocks
- Observe that non-assignment statements do not change values
  - $f$  is simply the identity function  $f(x) = x$  for such statements
- What about assignment statements?
  - $x = c$ , where  $x$  is a variable, and  $c$  is a constant
  - $x = y + z$ , where  $+$  is any binary operator
  - $x = *y$  or  $x = f(\dots)$ , where  $f$  is a function call

## The Transfer Function - II

- Note that IN (and OUT) are maps (i.e. dictionaries)
  - From variables to their current dataflow values (UNDEF,  $c$ , or NAC)
  - Let's call this map  $m$ , so that  $m(x)$  returns the dataflow value for variable  $x$
- $x = c$ , changes  $m(x) \leftarrow c$
- $x = y + z$ , where  $+$  is any binary operator (not just addition)
  - $m(x) \leftarrow m(y) + m(z)$  if  $m(y)$  and  $m(z)$  are constants
  - $m(x) \leftarrow \text{NAC}$  if either  $m(y)$  or  $m(z)$  is NAC
  - $m(x) \leftarrow \text{UNDEF}$  otherwise
- $x = *y$  or  $x = f(\dots)$ ,  $m(x) \leftarrow \text{NAC}$  (conservatively)
- Note that  $m(v) \leftarrow m(v)$  for all  $v \neq x$ 
  - I.e. the other values of the map remain unchanged

## Is this monotonic?

Is  $\text{OUT}[s] \leq \text{IN}[s]$  for every  $s$ ?

- For the two cases below, it is “surely ... monotone”:
  - $m(x) \leftarrow c$
  - $m(x) \leftarrow \text{NAC}$
- What about  $x = y + z$ ?
  - Need to show that  $m(x)$  does not get greater as  $m(y)$  (and/or)  $m(z)$  get smaller
  - Show by case analysis and symmetry

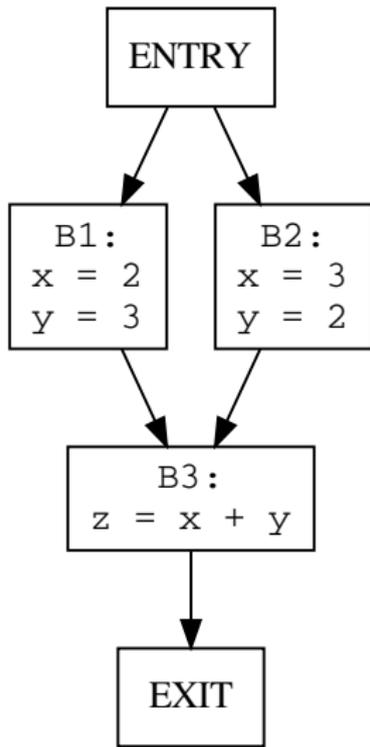
$x = y + z$  as  $m(z)$  gets smaller

$m(y)$	$m(z)$	output $m(x)$
UNDEF	UNDEF $c_2$ NAC	UNDEF
$c_1$	UNDEF $c_2$ NAC	
NAC	UNDEF $c_2$ NAC	NAC

$x = y + z$  as  $m(z)$  gets smaller (answers)

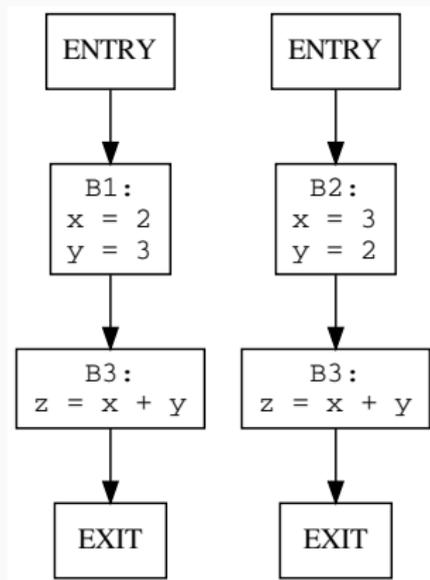
$m(y)$	$m(z)$	output $m(x)$
UNDEF	UNDEF $c_2$ NAC	UNDEF UNDEF NAC
$c_1$	UNDEF $c_2$ NAC	UNDEF $c_1 + c_2$ NAC
NAC	UNDEF $c_2$ NAC	NAC NAC NAC

## Is it distributive?



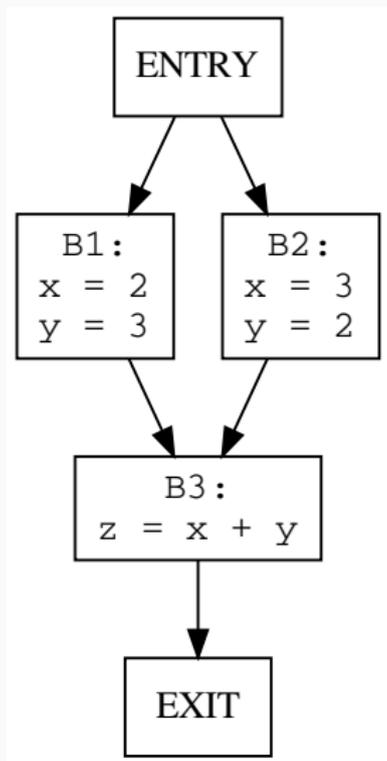
# MOP solution

- Path 1 ( $x = 2; y = 3; z = x + y$ )
  - $m(z) = 5$ , so  $z$  is a constant
- Path 2 ( $x = 3; y = 2; z = x + y$ )
  - $m(z) = 5$ , so  $z$  is a constant
- Meet over Path 1 and Path 2
  - $m(z) = 5 \wedge 5$ , so  $z$  is a constant



# MFP solution

- At end of block  $B_1$ 
  - $m(x) = 2$  and  $m(y) = 3$
- At end of block  $B_2$ 
  - $m(x) = 3$  and  $m(y) = 2$
- Meet before block  $B_3$ 
  - $m(x) = 2 \wedge 3$  (i.e. case  $c_1 \wedge c_2$ )
  - $m(y) = 3 \wedge 2$
- Conclusion?



## Constant Propagation is not distributive

- For constant propagation, in most non-trivial programs
  - $MFP < MOP$

# Outline

Review

Proofs

Constant Propagation

Postscript

# References

- Chapter 9 of the Dragon book
  - Section 9.3, 9.4