


Dataflow Analysis: Dataflow Frameworks

1

- 
- ## Outline of Today's Class
-
- Catch up, four classical dataflow problems
 - Dataflow frameworks
 - Lattices
 - Transfer functions
 - Worklist algorithm (next time)
 - Quiz1 next time (Thursday) on four classical dataflow problems
 - Reading:
 - Dragon Book, Chapter 9.2 and 9.3
- CSCI 4450/6450, A Milanova

2

2

Dataflow Analysis

Entry node: 1

Exit node: 9

- Control-flow graph (CFG):
 - $G = (N, E, 1)$
 - Nodes are basic blocks
- Data
- Dataflow equations

$$\text{out}(j) = (\text{in}(j) - \text{kill}(j)) \cup \text{gen}(j)$$
 (*gen* and *kill* are parameters)
- Merge operator \vee

$$\text{in}(j) = \vee \text{out}(i)$$
 i is predecessor of j

3

Problem 1. Reaching Definitions

(Reach)

- Problem statement: for each CFG node n , compute the set of definitions (x, k) that reach n
- First, define **data** (i.e., the dataflow facts) to propagate
 - Primitive dataflow facts are definitions (x, k)
 - Reach propagates sets of definitions, e.g., $\{(i, 1), (p, 4)\}$

CSCI 4450/6450, A Milanova

4

Reaching Definitions (*Reach*)

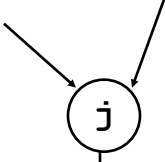
- Next, define the dataflow equations (i.e., effect of code at node j on incoming dataflow facts)

$j: \mathbf{x} = \mathbf{y} + \mathbf{z}$

$\left. \begin{array}{l} \text{kill}(j): \text{all definitions of } \mathbf{x}: (\mathbf{x}, _) \\ \text{gen}(j): \text{this definition of } \mathbf{x}: (\mathbf{x}, j) \end{array} \right\}$

$\text{out}(j) = (\text{in}(j) - \text{kill}(j)) \cup \text{gen}(j)$

E.g., if $\text{in}(4) = \{ (\mathbf{x}, 1), (\mathbf{y}, 2), (\mathbf{x}, 3) \}$
 Node 4 is: $\mathbf{x} = \mathbf{y} + \mathbf{z}$
 Then $\text{out}(4) = \{ (\mathbf{y}, 2), (\mathbf{x}, 4) \}$



CSCI 4450/6450, A Milanova

5

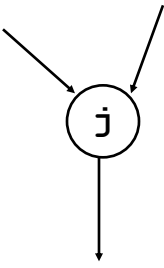
5

Reaching Definitions (*Reach*)

- Next, define the merge operator \vee (i.e., how to combine data from incoming paths)
- For *Reach*, \vee is the set union \cup

$\text{in}(j) = \{ \cup \text{out}(i) \mid i \text{ is predecessor of } j \}$

E.g., if $\text{out}(2) = \{ (\mathbf{x}, 1), (\mathbf{y}, 2) \}$ and
 $\text{out}(3) = \{ (\mathbf{x}, 3) \}$ and
 2 and 3 are predecessors of 4
 $\text{in}(4) = \{ (\mathbf{x}, 1), (\mathbf{x}, 3), (\mathbf{y}, 2) \}$



CSCI 4450/6450, A Milanova

6

6

Reaching Definitions

Forward, *may*
dataflow problem

CSCI 4450/6450, A Milanova 7

7

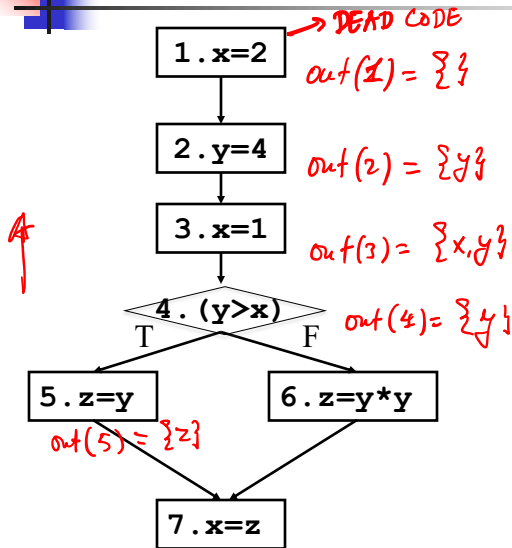
Problem 2. Live Uses of Variables (*Live*)

- We say that a variable x is “live on exit from node j ” if there is a live use of x on exit from j (recall the definition of “live use of x on exit from j ”)
- Problem statement: for each node n , compute the set of variables that are live on exit from n

1. $x=2$; 2. $y=4$; 3. $x=1$; if ($y>x$) then 5. $z=y$; else 6. $z=y*y$; 7. $x=z$;
 What variables are live on exit from statement 3? Statement 1?

8

Live Example



$$in(1) = \{\emptyset\}$$

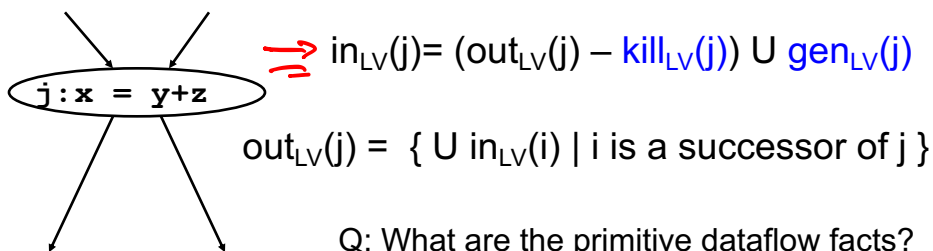
$$\beta_1 \begin{cases} x=2 \\ y=4 \\ x=1 \end{cases}$$

9

9

Live Uses of Variables (*Live*)

- Problem statement: for each node n , compute the set of variables that are live on exit from n



- Q: What are the primitive dataflow facts?
- Q: What is $gen_{LV}(j)$?
- Q: What is $kill_{LV}(j)$?

CSCI 4450/6450, A Milanova

10

10

Live Uses of Variables (*Live*)

- Data
 - Primitive facts: variables \mathbf{x}
 - Propagates sets: $\{\mathbf{x}, \mathbf{y}, \mathbf{z}\}$

- Dataflow equations. At j : $\mathbf{x} = \mathbf{y} + \mathbf{z}$
 - $kill_{LV}(j)$: $\{\mathbf{x}\}$
 - $gen_{LV}(j)$: $\{\mathbf{y}, \mathbf{z}\}$

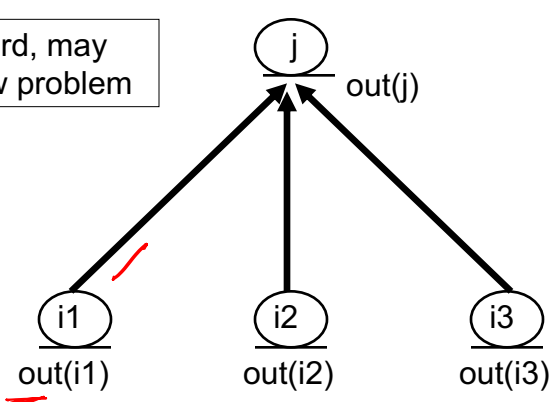
- Merge operator: set union \cup

CSCI 4450/6450, A Milanova 11

11

Live Uses of Variables

Backward, may
dataflow problem



```

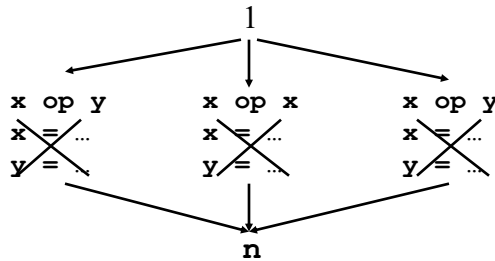
graph BT
  i1((i1)) -- out(i1) --> j((j))
  i2((i2)) -- out(i2) --> j
  i3((i3)) -- out(i3) --> j
  
```

CSCI 4450/6450, A Milanova 12

12

Available Expressions

- An expression $x \text{ op } y$ is **available** at program point n if **every** path from entry to n evaluates $x \text{ op } y$, and there are NO subsequent assignments to x or y after evaluation and prior to reaching n .



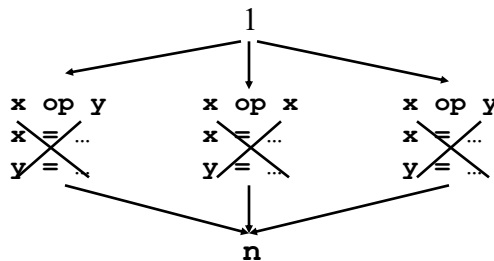
CSCI 4450/6450, A Milanova

13

13

Problem 3. Available Expressions (*Avail*)

- Problem statement: For every node n , compute the set of expressions that are available at n

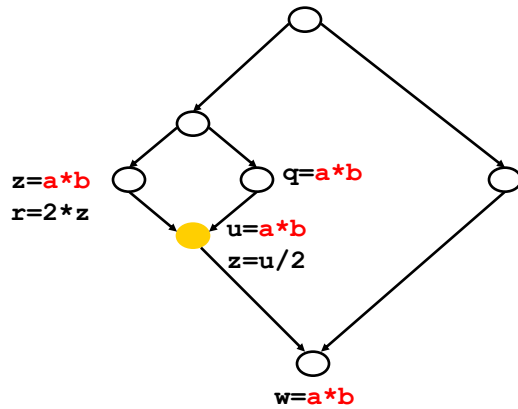


CSCI 4450/6450, A Milanova

14

14

Avail Enables Global Common Subexpression Elimination



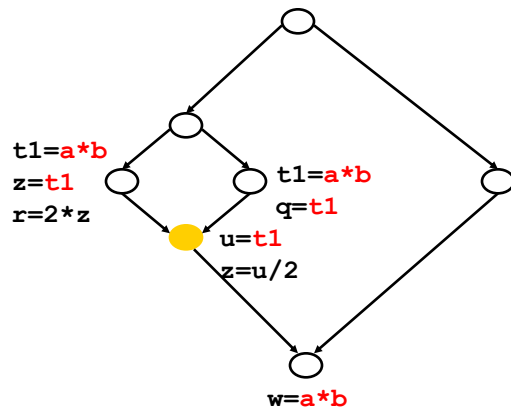
CSCI 4450/6450, A Milanova

15

15

Avail Enables Global Common Subexpression Elimination

Can we eliminate $w=a*b$?



CSCI 4450/6450, A Milanova

16

16

Available Expressions (*Avail*)

- Data?
 - Primitive dataflow facts are expressions, e.g., $x+y, a*b, a+2$
 - Analysis propagates sets of expressions, e.g., $\{x+y, a*b\}$
- Dataflow equations at j : $x = y \text{ op } z$?
 - $out_{AE}(j) = (in_{AE}(j) - kill_{AE}(j)) \cup gen_{AE}(j)$
 - $kill_{AE}(j)$: all expressions with operand x : $(x \text{ op } _), (_ \text{ op } x)$
 - $gen_{AE}(j)$: new expression: $\{(y \text{ op } z)\}$

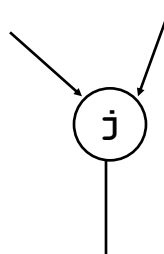
17

17

Available Expressions (*Avail*)

- Merge operator?
 - For *Avail*, it is set intersection \cap

$$in_{AE}(j) = \{\cap out_{AE}(i) \mid i \text{ is predecessor of } j\}$$

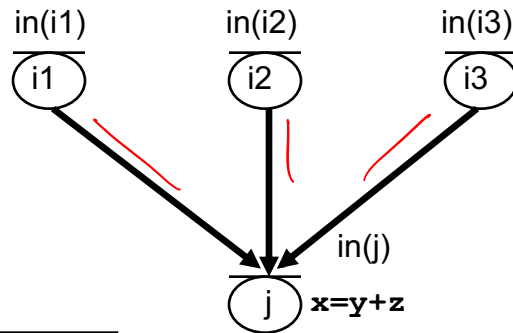


$j: x = x * y \quad \begin{cases} t1 = x * y \\ x = t1 \end{cases}$
 $kill(j) = \{(x \text{ op } _), (_ \text{ op } x)\}$
 $gen(j) = \{j\}$

CSCI 4450/6450, A Milanova 18

18

Available Expressions (Avail)



Forward, must
dataflow problem

CSCI 4450/6450, A Milanova

19

19

Example

1. $y = a + b$

$$out(1) = \{a + b\}$$

2. $x = a * b$

$$out(2) = \{a * b, a + b\}$$

$$in(3) = \{a * b\}$$

3. if $y \leq a * b$

$$in(4) = \{a * b\}$$

4. $a = a + 1$

$$in(5) = \{\}$$

5. $x = a * b$

$$in(6) = \{a * b\}$$

6. goto 3

$$in(7) = \{a * b\}$$

7. ...

20

20

Note on Homework

$$B_1 \begin{cases} 1. x = x + b \\ 2. y = x + 1 \\ 3. x = x + y \end{cases}$$

Reach:
 All definitions of x and all definitions of y .

$$kill(B_1) = \{ (x, -), (y, -) \}$$

$$gen(B_1) = \{ (x, b), (y, 1) \}$$

Avail:

$$kill(B_1) = \{ (x \text{ op } -), (- \text{ op } x), (x \text{ op } -), (- \text{ op } y) \}$$

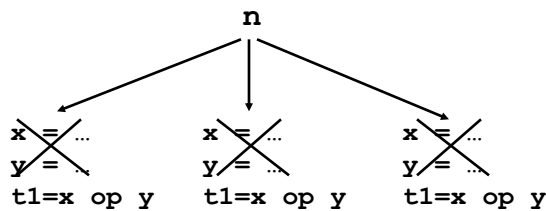
$$gen(B_1) = \{ \}$$

21

21

Very Busy Expressions

- An expression $x \text{ op } y$ is **very busy** at node n , if along EVERY path from n to the end of the program, we come to a computation of $x \text{ op } y$ BEFORE any redefinition of x or y .



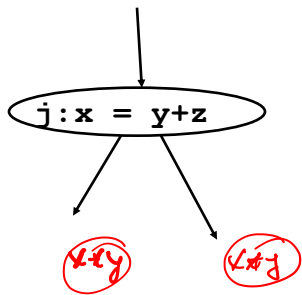
CSCI 4450/6450, A Milanova

22

22

Problem 4. Very Busy Expressions (*VeryB*)

- Problem Statement: For each node n , compute the set of expressions that are very busy on exit from n



Q: What is the data?

Q: What are the equations?

Q: What is $\text{gen}_{\text{VB}}(j)$?

Q: What is $\text{kill}_{\text{VB}}(j)$?

Q: What is the merge operator?

23

23

Very Busy Expressions (*VeryB*)

- Data?
 - Primitive dataflow facts are expressions, e.g., $x+y$, $a*b$
 - Analysis propagates sets of expressions, e.g., $\{x+y, a*b\}$
- Dataflow equations at $j: x = y \text{ op } z$?
 - $\text{in}(j) = \text{gen}(j) \cup (\text{out}(j) - \text{kill}(j))$
 - $\text{kill}(j)$: all expressions with operand x : $(x \text{ op } _)$, $(_ \text{ op } x)$
 - $\text{gen}(j)$: new expression: $\{ (y \text{ op } z) \}$

24

24

Very Busy Expressions (VeryB)

- Merge operator?
 - For *VeryB*, it is set intersection \cap

$$\text{out}_{\text{VB}}(j) = \{ \cap \text{in}_{\text{VB}}(i) \mid i \text{ is successor of } j \}$$

$j: a = a * b$

$\text{kill}(j) = \{ (a \text{ op } -), (- \text{ op } a) \}$

$\text{gen}(j) = \{ a * b \}$

CSCI 4450/6450, A Milanova 25

25

Very Busy Expressions

Backward, must
dataflow problem

CSCI 4450/6450, A Milanova 26

26

Another Example: Taint Analysis

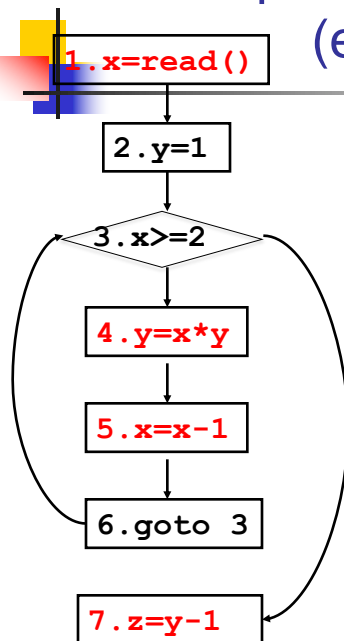
- A definition $i: x = \dots (x, i)$ is **tainted** if
 - $i: x = \text{tainted_source}()$ is designated as a taint **source**
 - e.g., `deviceId=telephony_mgr.getDeviceId();`
 - or $i: x = y \text{ op } z$ and a tainted (y, j) or a tainted (z, k) reaches program point i
- Problem statement: for each node n , compute the set of tainted definitions that reach n

CSCI 4450/6450, A Milanova

27


27

Example: Taint Analysis (explicit flow)



28

28




Outline of Today's Class

- Catch up
- **Dataflow frameworks**
 - Lattices
 - Transfer functions
 - Worklist algorithm
- Reading:
 - Dragon Book, Chapter 9.2 and 9.3

CSCI 4450/6450, A Milanova 29

29




Dataflow Problems

	<i>May Problems</i>	<i>Must Problems</i>
<i>Forward Problems</i>	<div style="border: 2px solid red; border-radius: 50%; padding: 5px; display: inline-block;"> Reaching Definitions </div>	Available Expressions
<i>Backward Problems</i>	Live Uses of Variables	Very Busy Expressions

CSCI 4450/6450, A Milanova 30

30




Similarities

- Analyses operate over similar **property spaces**
- In all cases, analysis operates over a finite set **D** of primitive dataflow facts
 - *Reach*: **D** is the set of all definitions in the program:
e.g., $\{ (x, 1), (y, 2), (x, 4), (y, 5) \}$
 - *Avail* and *VeryB*: **D** is the set of all arithmetic expressions:
e.g., $\{ a+b, a*b, a+1 \}$
 - *Live*: **D** is the set of all variables
e.g., $\{ x, y, z \}$
- Solution at node **n** is a subset of **D** (e.g., a definition either reaches **n** or it does not reach **n**)

CSCI 4450/6450, A Milanova 31

31



Similarities

- Dataflow equations have same form (from now on, we'll focus on forward problems):

$$\mathbf{out(j) = (in(j) - kill(j)) \cup gen(j) = (in(j) \cap pres(j)) \cup gen(j)}$$

$$\mathbf{in(j) = \{ \forall out(i) \mid i \text{ is predecessor of } j \}}$$

pres(j) is the complement of **kill(j)** in **D**

 - A note: what makes the 4 classical problems special is that sets kill(i)/pres(j) and gen(i) do not depend on in(j)
 - Set union and set intersection can be implemented as logical OR and AND respectively

CSCI 4450/6450, A Milanova 32

32



Similarities

- Dataflow equation at node j is a **transfer function**. It takes $\text{in}(j)$ as argument and produces $\text{out}(j)$ as result:

- **$\text{out}(j) = f_j(\text{in}(j))$**



Dataflow Frameworks

- We generalize and study properties of the **property space**
 - Property space is a **lattice**
 - Choice of lattice settles **merge operator**
- We generalize and study properties of the **transfer function space**
 - Functions are **monotone or distributive**
- We generalize and study properties of the **worklist algorithm** that computes a solution

Lattices

- **Partial ordering** (denoted by \leq or \sqsubseteq)
 - Relation between pairs of elements
 - Reflexive $a \leq a$
 - Anti-symmetric $a \leq b$ and $b \leq a \implies a = b$
 - Transitive $a \leq b$ and $b \leq c \implies a \leq c$
- **Partially ordered set** (poset) (set S, \leq)
 - **0** element $0 \leq a$, for every a in S
 - **1** element $a \leq 1$, for every a in S

We don't necessarily need 0 or 1 element

CSCI 4450/6450, A Milanova 35

35

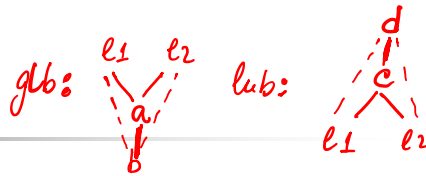
Poset Example

$D = \{a, b, c\}$
 The poset is $\mathcal{P}(D)$, \leq is set inclusion

CSCI 4450/6450, A Milanova 36

36

Lattice Theory



- Greatest lower bound (glb)
 - l_1, l_2 in poset S , a in poset S is the **glb**(l_1, l_2) iff
 - 1) $a \leq l_1$ and $a \leq l_2$
 - 2) for any b in S , $b \leq l_1, b \leq l_2$ implies $b \leq a$

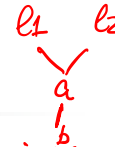
If glb exists, it is unique. Why? Called **meet** (denoted by \wedge or \sqcap) of l_1 and l_2 .

- Least upper bound (lub)
 - l_1, l_2 in poset S , c in poset S is the **lub**(l_1, l_2) iff
 - 1) $c \geq l_1$ and $c \geq l_2$
 - 2) for any d in S , $d \geq l_1, d \geq l_2$ implies $d \geq c$

If lub exists, it is unique. Called **join** (denoted by \vee or \sqcup) of l_1 and l_2 .

37

37



Thus: If glb of l_1 and l_2 exist, then it is unique.

Sketch: let a be a glb of l_1 and l_2 .

(1) Thus: $a \leq l_1, a \leq l_2, \forall b'. b' \leq l_1, b' \leq l_2 \Rightarrow b' \leq a$

let b be a glb of l_1 and l_2 .

(2) Thus: $b \leq l_1, b \leq l_2, \forall a'. a' \leq l_1, a' \leq l_2 \Rightarrow a' \leq b$

Thus $a \leq b$ and $b \leq a$

Thus $a = b$ (by antisymmetry)

CSCI 4450/6450, A Milanova

38

38

Definition of a Lattice (L, \wedge, \vee)

- A lattice L is a poset under \leq , such that every pair of elements has a **glb (meet)** and **lub (join)**
- A lattice need not contain a 0 or 1 element
- A finite lattice must contain 0 and 1 elements
- Not every poset is a lattice
- If there is element a such that $a \leq x$ for every x in L , then a is the 0 element of L
- If there is a such that $x \leq a$ for every x in L , then a is the 1 element of L

CSCI 4450/6450, A Milanova 39

39

A Poset but Not a Lattice

There is no **lub(e3,e4)** in this poset so it is not a lattice.
Suppose we add the **lub(e3,e4)**, is it a lattice?

CSCI 4450/6450, A Milanova 40

40

Is This Poset a Lattice

$D = \{a, b, c\}$
 The poset is 2^D , \leq is set inclusion

$s_1 \leq s_2 \iff s_1 \subseteq s_2$

$a \leq b$
 $glb(a, b) = a$
 $lub(a, b) = b$

CSCI 4450/6450, A Milanova 41

41

Examples of Lattices

- $H = (2^D, \cap, \cup)$ where D is a finite set
 - $glb(s_1, s_2)$ denoted $s_1 \wedge s_2$, is set intersection $s_1 \cap s_2$
 - $lub(s_1, s_2)$ denoted $s_1 \vee s_2$, is set union $s_1 \cup s_2$
- $J = (\mathbb{N}_1, gcd, lcm)$
 - Partial order is integer divide on \mathbb{N}_1
 - $lub(n_1, n_2)$ denoted $n_1 \vee n_2$ is $lcm(n_1, n_2)$
 - $glb(n_1, n_2)$ denoted $n_1 \wedge n_2$ is $gcd(n_1, n_2)$

(\mathbb{N}_1 denotes natural numbers starting at 1)

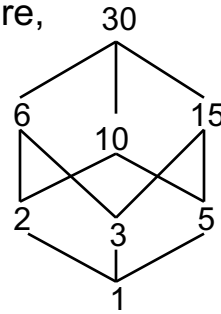
CSCI 4450/6450, A Milanova 42

42



Chain

- A poset C where for every pair of elements c_1, c_2 in C , either $c_1 \leq c_2$ or $c_2 \leq c_1$.
 - E.g., $\{\} \leq \{a\} \leq \{a,b\} \leq \{a,b,c\}$
 - E.g., from the lattice J as shown here,
 $1 \leq 2 \leq 6 \leq 30$
 $1 \leq 3 \leq 15 \leq 30$
- A lattice s.t. every ascending chain is finite, is said to satisfy the *Ascending Chain Condition*



CSCI 4450/6450, A Milanova

43

43



- Does lattice H satisfy the Ascending Chain Condition (ACC)?

- Does lattice J satisfy the ACC?

CSCI 4450/6450, A Milanova

44

44

Lattices in Dataflow Analysis

- Lattices define property space
- Lattice properties lead to certain properties of the **worklist algorithm** (standard way of solving dataflow problems)

CSCI 4450/6450, A Milanova 45

45

Dataflow Lattices: *Reach*

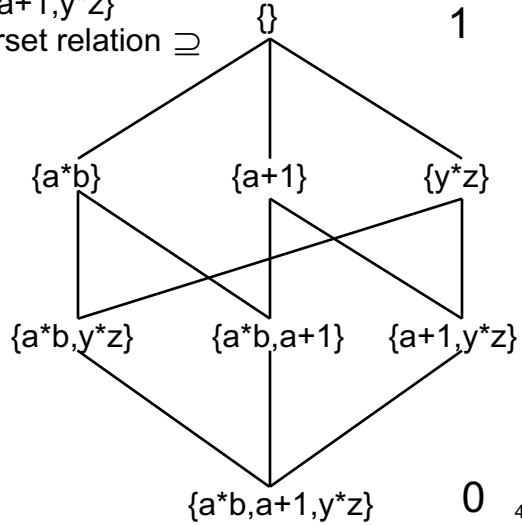
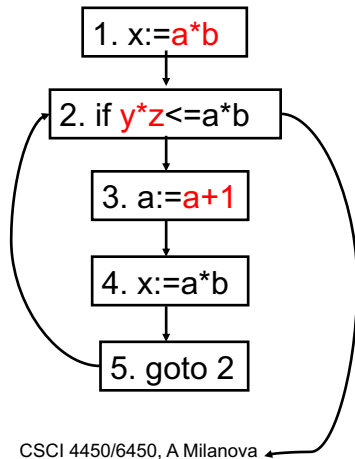
$D = \text{all definitions: } \{(x,1), (x,4), (a,3)\} \quad \{(x,1), (x,4), (a,3)\} \quad 1$
 Poset is 2^D , \leq is the subset relation \subseteq

CSCI 4450/6450, A Milanova 0 46

46

Dataflow Lattices: *Avail*

$D =$ all expressions: $\{a*b, a+1, y*z\}$
 Poset is 2^D , \leq is the superset relation \supseteq



CSCI 4450/6450, A Milanova

47

47

Property Space


■ Property space must be:

1. A lattice L, \leq
2. L satisfies the *Ascending Chain Condition*
 Requires that all ascending chains are finite

CSCI 4450/6450, A Milanova

48

48




Property Space

- **Merge operator V must be the join of L**
- In dataflow, L is often the lattice of the subsets over a finite set of dataflow facts D
 - Choose universal set D (e.g., all definitions)
 - Choose ordering operation \leq . Since the merge operator must be the join of L , a *may* problem sets \leq to **subset** and a *must* problem sets \leq to **superset**

CSCI 4450/6450, A Milanova 49

49




Example: *Reach* Lattice

- Property space is the lattice of the subsets
 - D is the set of all definitions in program
 - \leq is the **subset** operation
 - Thus, **join** is set union, as needed for *Reach*, which is a *may* problem
 - Lattice has a **0** being $\{\}$, and a **1** being D
 - Lattice satisfies the *Ascending Chain Condition*

CSCI 4450/6450, A Milanova 50

50

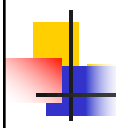


Example: *Avail* Lattice

- Property space is the lattice of the subsets
 - \mathbf{D} is the set of all expressions in the program
 - \leq is **superset**
 - Thus, **join** is set intersection, as needed for *Avail*, which is a *must* problem
 - Lattice has a **0** being \mathbf{D} , and a **1** being $\{\}$
 - Lattice satisfies *Ascending Chain Condition*

CSCI 4450/6450, A Milanova 51

51




(Monotone) Dataflow Framework

- A problem fits into the dataflow framework if
 - its property space is a lattice \mathbf{L}, \leq that satisfies the *Ascending Chain Condition*
 - its merge operator \vee is the join of \mathbf{L}
and
 - its transfer function space $\mathbf{F}: \mathbf{L} \rightarrow \mathbf{L}$ is monotone
- Thus, we can make use of a generic solution procedure, known as the **worklist algorithm** (also the **maximal fixpoint algorithm** or the **fixpoint iteration algorithm**)

52

52




Outline of Today's Class

- Catch up
- Dataflow frameworks
 - Lattices
 - **Transfer functions**
 - Worklist algorithm
- Reading:
 - Dragon Book, Chapter 9.2 and 9.3

CSCI 4450/6450, A Milanova 53

53




Transfer Functions

- **The transfer functions: $f: L \rightarrow L$.** Formally, function space \mathbf{F} is such that
 1. \mathbf{F} contains all f_j
 2. \mathbf{F} contains the identity function $\mathbf{id(x) = x}$
 3. \mathbf{F} is closed under composition
 4. **Each f must be monotone**

CSCI 4450/6450, A Milanova 54

54




Monotonicity Property

- $F: L \rightarrow L$ is **monotone** if and only if:
 - (1) a, b in L , f in F then $a \leq b \implies f(a) \leq f(b)$
 - or (equivalently):
 - (2) x, y in L , f in F then $f(x) \vee f(y) \leq f(x \vee y)$
- Theorem: Definitions (1) and (2) are equivalent.
 - Show that (1) implies (2)
 - Show that (2) implies (1)

55

55



Monotonicity Property

- Show that (1) implies (2)

CSCI 4450/6450, A Milanova

56

56



Distributivity Property

- $F: L \rightarrow L$ is **distributive** if and only if x, y in L , f in F then $f(x) \vee f(y) = f(x \vee y)$
- A distributive function is also monotone but not the other way around
- Distributivity is a very nice property!

CSCI 4450/6450, A Milanova

57

57



Monotonicity and Distributivity

- Is classical *Reach* distributive?
 - Yes

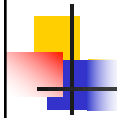
- To show distributivity:

$$\text{For each } j: ((X_1 \cup X_2) \cap \text{pres}(j)) \cup \text{gen}(j) = \\ ((X_1 \cap \text{pres}(j)) \cup \text{gen}(j)) \cup ((X_2 \cap \text{pres}(j)) \cup \text{gen}(j))$$

$$((X_1 \cup X_2) \cap \text{pres}(j)) \cup \text{gen}(j) = \\ ((X_1 \cap \text{pres}(j)) \cup (X_2 \cap \text{pres}(j))) \cup \text{gen}(j) = \\ ((X_1 \cap \text{pres}(j)) \cup \text{gen}(j)) \cup ((X_2 \cap \text{pres}(j)) \cup \text{gen}(j))$$

58

58



Monotone Dataflow Framework

- A problem fits into the dataflow framework if
 - its property space is a lattice \mathbf{L}, \leq that satisfies the *Ascending Chain Condition*
 - its merge operator V is the join of \mathbf{L}and
 - its transfer function space $\mathbf{F}: \mathbf{L} \rightarrow \mathbf{L}$ is monotone
- Thus, we can make use of a generic solution procedure, known as the **worklist algorithm**.

59