

## Lattice-Theoretic Framework for Data-Flow Analysis

### Last time

- Generalizing data-flow analysis

### Today

- Introduce lattice-theoretic frameworks for data-flow analysis

## Context for Lattice-Theoretic Framework

### Goals

- Provide a single formal model that describes all data-flow analyses
- Formalize the notions of “correct,” “conservative,” and “optimistic”
- Correctness proof for IDFA (iterative data-flow analysis)
- Place bounds on time complexity of data-flow analysis

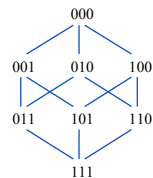
### Approach

- Define **domain** of program properties (flow values) computed by data-flow analysis, and organize the domain of elements as a **lattice**
- Define flow functions and a merge function over this domain using lattice operations
- Exploit lattice theory in achieving goals

## Lattices

### Define lattice $L = (V, \sqcap)$

- $V$  is a set of elements of the lattice
- $\sqcap$  is a binary relation over the elements of  $V$  (**meet** or **greatest lower bound**)



### Properties of $\sqcap$

- $x, y \in V \Rightarrow x \sqcap y \in V$
- $x, y \in V \Rightarrow x \sqcap y = y \sqcap x$
- $x, y, z \in V \Rightarrow (x \sqcap y) \sqcap z = x \sqcap (y \sqcap z)$

(closure)

(commutativity)

(associativity)

## Lattices (cont)

### Under ( $\sqsubseteq$ )

- Imposes a partial order on  $V$
- $x \sqsubseteq y \Leftrightarrow x \sqcap y = x$

### Top ( $\top$ )

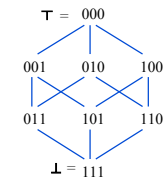
- A unique “greatest” element of  $V$  (if it exists)
- $\forall x \in V - \{\top\}, x \sqsubseteq \top$

### Bottom ( $\perp$ )

- A unique “least” element of  $V$  (if it exists)
- $\forall x \in V - \{\perp\}, \perp \sqsubseteq x$

### Height of lattice $L$

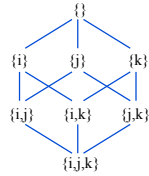
- The longest path through the partial order from greatest to least element (top to bottom)



## Data-Flow Analysis via Lattices

### Relationship

- Elements of the lattice ( $V$ ) represent flow values (in[] and out[] sets)
  - e.g., Sets of live variables for liveness
- $\top$  represents “best-case” information (initial flow value)
  - e.g., Empty set
- $\perp$  represents “worst-case” information
  - e.g., Universal set
- $\sqcap$  (meet) merges flow values
  - e.g., Set union
- If  $x \sqsubseteq y$ , then  $x$  is a conservative approximation of  $y$ 
  - e.g., Superset

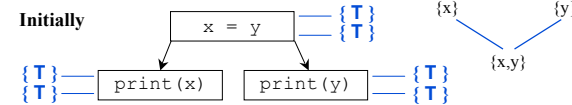


## Data-Flow Analysis via Lattices (cont)

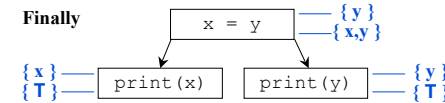
### Remember what these flow values represent

- At each program point a lattice element represents an in[] set or an out[] set

Initially



Finally



## Data-Flow Analysis Frameworks

### Data-flow analysis framework

- A set of **flow values** ( $V$ )
- A binary **meet operator** ( $\sqcap$ )
- A set of **flow functions** ( $F$ ) (also known as **transfer functions**)

### Flow Functions

- $F = \{f: V \rightarrow V\}$
- $f$  describes how each node in CFG affects the flow values
- Flow functions map program behavior onto lattices

## Visualizing DFA Frameworks as Lattices

### Example: Liveness analysis with 3 variables

$$U = \{v1, v2, v3\}$$

- $V: 2^U = \{\{v1, v2, v3\}, \{v1, v2\}, \{v1, v3\}, \{v2, v3\}, \{v1\}, \{v2\}, \{v3\}, \emptyset\}$

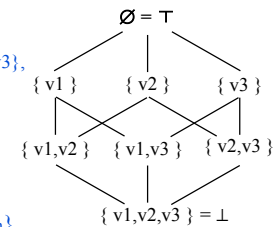
- Meet ( $\sqcap$ ):  $U$

- $\sqsubseteq$ :  $\supseteq$

- Top ( $\top$ ):  $\emptyset$

- Bottom ( $\perp$ ):  $U$

- $F: \{f_n(X) = \text{Gen}_n \cup (X - \text{Kill}_n), \forall n\}$



Inferior solutions are lower on the lattice

More conservative solutions are lower on the lattice

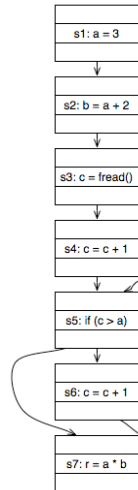
## Lattice Example

What is the data-flow set for liveness?

What is the meet operation for liveness?

What partial order does the meet operation induce?

What is the liveness lattice for this example?



## Recall Liveness Analysis

Data-flow equations for liveness

$$\text{in}[n] = \text{use}[n] \cup (\text{out}[n] - \text{def}[n])$$

$$\text{out}[n] = \bigcup_{s \in \text{succ}[n]} \text{in}[s]$$

Liveness equations in terms of Gen and Kill

$$\text{in}[n] = \text{gen}[n] \cup (\text{out}[n] - \text{kill}[n])$$

} A use of a variable **generates** liveness  
 } A def of a variable **kills** liveness

**Gen:** New information that's added at a node

**Kill:** Old information that's removed at a node

Can define (almost) any data-flow analysis in terms of Gen and Kill

## More Examples

Reaching definitions

– V:  $2^S$  (S = set of all defs)

–  $\cap$ :  $\cup$

–  $\sqsubseteq$ :  $\supseteq$

– Top( $\top$ ):  $\emptyset$

– Bottom ( $\perp$ ):  $\mathcal{U}$

– F: ...

Reaching Constants

– V:  $2^{V \times c}$ , variables v and constants c

–  $\cap$ :  $\cap$

–  $\sqsubseteq$ :  $\supseteq$

– Top( $\top$ ):  $\mathcal{U}$

– Bottom ( $\perp$ ):  $\emptyset$

– F: ...

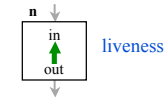
## Direction of Flow

Backward data-flow analysis

– Information at a node is based on what happens **later** in the flow graph  
i.e.,  $\text{in}[]$  is defined in terms of  $\text{out}[]$

$$\text{in}[n] = \text{gen}[n] \cup (\text{out}[n] - \text{kill}[n])$$

$$\text{out}[n] = \bigcup_{s \in \text{succ}[n]} \text{in}[s]$$

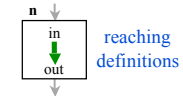


Forward data-flow analysis

– Information at a node is based on what happens **earlier** in the flow graph  
i.e.,  $\text{out}[]$  is defined in terms of  $\text{in}[]$

$$\text{in}[n] = \bigcup_{p \in \text{pred}[n]} \text{out}[p]$$

$$\text{out}[n] = \text{gen}[n] \cup (\text{in}[n] - \text{kill}[n])$$



Some problems need both forward and backward analysis

– e.g., Partial redundancy elimination (uncommon)

## Merging Flow Values

### Live variables and reaching definitions

- Merge **flow values** via set union

#### Reaching Definitions

$$\begin{aligned} \text{in}[n] &= \bigcup_{p \in \text{pred}[n]} \text{out}[p] \\ \text{out}[n] &= \text{gen}[n] \cup (\text{in}[n] - \text{kill}[n]) \end{aligned}$$

#### Live Variables

$$\begin{aligned} \text{out}[n] &= \bigcup_{s \in \text{succ}[n]} \text{in}[s] \\ \text{in}[n] &= \text{gen}[n] \cup (\text{out}[n] - \text{kill}[n]) \end{aligned}$$

Why?

When might this be inappropriate?

CS553 Lecture

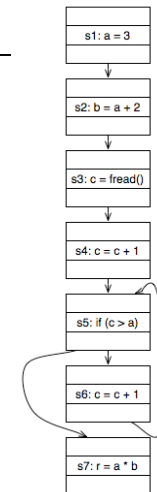
Lattice Theoretic Framework for DFA

13

## Reaching Defs Example

What is the initial guess?

What is the meet operation?



CS553 Lecture

Lattice Theoretic Framework for DFA

## Available Expressions (cont)

### Data-Flow Equations

$$\begin{aligned} \text{in}[n] &= \bigcap_{p \in \text{pred}[n]} \text{out}[p] \\ \text{out}[n] &= \text{gen}[n] \cup (\text{in}[n] - \text{kill}[n]) \end{aligned}$$

Plug it in to our general DFA algorithm

for each node n

in[n] = v; out[n] = v

repeat

for each n

in'[n] = in[n]

out'[n] = out[n]

in[n] =  $\bigcap_{p \in \text{pred}[n]} \text{out}[p]$

out[n] =  $\text{gen}[n] \cup (\text{in}[n] - \text{kill}[n])$

until in'[n]=in[n] and out'[n]=out[n] for all n

CS553 Lecture

Lattice Theoretic Framework for DFA

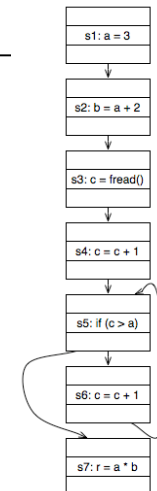
15

## Available Expressions Example

What is the initial guess?

What is the meet operation?

What does the lattice look like?



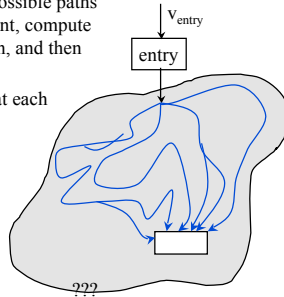
CS553 Lecture

Lattice Theoretic Framework for DFA

## Solving Data-Flow Analyses

### Goal

- For a forward problem, consider all possible paths from the entry to a given program point, compute the flow values at the end of each path, and then meet these values together
- Meet-over-all-paths (MOP) solution at each program point
- $\sqcap_{\text{all paths } n_1, n_2, \dots, n_i} (f_{n_1}(\dots f_{n_2}(f_{n_1}(v_{\text{entry}}))))$



## Solving Data-Flow Analyses (cont)

### Problems

- Loops result in an infinite number of paths
- Statements following merge must be analyzed for all preceding paths
  - Exponential blow-up

### Solution

- Compute meets early (at merge points) rather than at the end
- Maximum fixed-point (MFP)

### Questions

- Is this correct?
- Is this efficient?
- Is this accurate?

## Correctness

“Is  $v_{\text{MFP}}$  correct?” ■ “Is  $v_{\text{MFP}} \sqsubseteq v_{\text{MOP}}$ ?”

### Look at Merges

$$v_{\text{MOP}} = F_r(v_{p1}) \sqcap F_r(v_{p2})$$

$$v_{\text{MFP}} = F_r(v_{p1} \sqcap v_{p2})$$

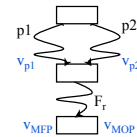
$$v_{\text{MFP}} \sqsubseteq v_{\text{MOP}} = F_r(v_{p1} \sqcap v_{p2}) \sqsubseteq F_r(v_{p1}) \sqcap F_r(v_{p2})$$

### Observation

$$\forall x, y \in V$$

$$f(x \sqcap y) \sqsubseteq f(x) \sqcap f(y) \iff x \sqsubseteq y \implies f(x) \sqsubseteq f(y)$$

$\therefore v_{\text{MFP}}$  correct when  $F_r$  (really, the flow functions) are monotonic



## Monotonicity

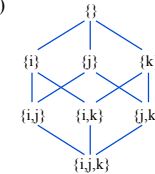
**Monotonicity:**  $(\forall x, y \in V)[x \sqsubseteq y \implies f(x) \sqsubseteq f(y)]$

- If the flow function  $f$  is applied to two members of  $V$ , the result of applying  $f$  to the “lesser” of the two members will be under the result of applying  $f$  to the “greater” of the two
- Giving a flow function more conservative inputs leads to more conservative outputs (never more optimistic outputs)

### Why else is monotonicity important?

#### For monotonic $F$ over domain $V$

- The maximum number of times  $F$  can be applied to self w/o reaching a fixed point is  $\text{height}(V) - 1$
- IDFA is guaranteed to terminate if the flow functions are monotonic and the lattice has finite height



## Efficiency

### Parameters

- n: Number of nodes in the CFG
- k: Height of lattice
- t: Time to execute one flow function

### Complexity

- $O(nkt)$

### Example

- Reaching definitions?

## Accuracy

### Distributivity

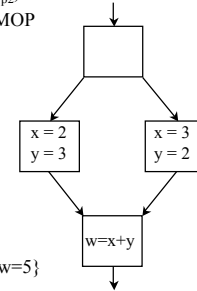
- $f(u \sqcap v) = f(u) \sqcap f(v)$
- $V_{MFP} \sqsubseteq V_{MOP} \equiv F_i(v_{p1} \sqcap v_{p2}) \sqsubseteq F_i(v_{p1}) \sqcap F_i(v_{p2})$
- If the flow functions are distributive, MFP = MOP

### Examples

- Reaching definitions?
- Reaching constants?

$$f(u \sqcap v) = f(\{x=2, y=3\} \sqcap \{x=3, y=2\}) \\ = f(\emptyset) = \emptyset$$

$$f(u) \sqcap f(v) = f(\{x=2, y=3\}) \sqcap f(\{x=3, y=2\}) \\ = [\{x=2, y=3, w=5\} \sqcap \{x=2, y=2, w=5\}] = \{w=5\} \\ \Rightarrow \text{MFP} \neq \text{MOP}$$



## Tuples of Lattices

### Problem

- Simple analyses may require very complex lattices (e.g., Reaching constants)

### Solution

- Use a tuple of lattices, one per variable

$$L = (V, \sqcap) \quad \blacksquare \quad (L_T = (V_T, \sqcap_T))^N$$

- $V = (V_T)^N$
- Meet ( $\sqcap$ ): point-wise application of  $\sqcap_T$
- $(\dots, v_i, \dots) \sqcap (\dots, u_i, \dots) \equiv v_i \sqcap u_i, \forall i$
- Top ( $\top$ ): tuple of tops ( $\top_T$ )
- Bottom ( $\perp$ ): tuple of bottoms ( $\perp_T$ )
- Height ( $L$ ) =  $N * \text{height}(L_T)$

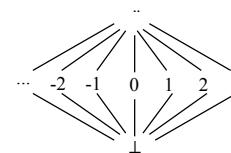
## Tuples of Lattices Example

### Reaching constants (previously)

- $P = v \times c$ , for variables  $v$  & constants  $c$
- $V: 2^P$

### Alternatively

- $V = c \cup \{\top, \perp\}$



The whole problem is a tuple of lattices, one for each variable

## Examples of Lattice Domains

---

### Two-point lattice ( $\top$ and $\perp$ )

- Examples?
- Implementation?

### Set of incomparable values (and $\top$ and $\perp$ )

- Examples?

### Powerset lattice ( $2^S$ )

- $\top = \emptyset$  and  $\perp = S$ , or vice versa
- Isomorphic to tuple of two-point lattices

## Concepts

---

### Lattices

- Conservative approximation
- Optimistic (initial guess)
- Data-flow analysis frameworks
- Tuples of lattices

### Data-flow analysis

- Fixed point
- Meet-over-all-paths (MOP)
- Maximum fixed point (MFP)
- Legal/safe/correct (monotonic)
- Efficient
- Accurate (distributive)

## Next Time

---

### Lecture

- Some transformations that you can implement for Project 4
  - Copy propagation
  - Constant propagation
  - Common sub-expression elimination