

## Lattice-Theoretic Framework for Data-Flow Analysis

### Last time

- Generalizing data-flow analysis

### Today

- Finish generalizing data-flow analysis
- Reaching Constants introduction
- Introduce lattice-theoretic frameworks for data-flow analysis

## Defining Available Expressions Analysis

<b>Must or may Information?</b>	Must
<b>Direction?</b>	Forward
<b>Flow values?</b>	Sets of expressions
<b>Initial guess?</b>	Universal set
<b>Kill?</b>	Set of expressions killed by statements
<b>Gen?</b>	Set of expressions evaluated by s
<b>Merge?</b>	Intersection

## Reaching Constants

### Goal

- Compute value of each variable at each program point (if possible)

### Flow values

- Set of (variable, constant) pairs

### Merge function

- Intersection

### Data-flow equations

- Effect of node  $n$   $x = c$ 
  - $kill[n] = \{(x,d) \mid \forall d\}$
  - $gen[n] = \{(x,c)\}$
- Effect of node  $n$   $x = y + z$ 
  - $kill[n] = \{(x,c) \mid \forall c\}$
  - $gen[n] = \{(x,c) \mid c = \text{val}_y + \text{val}_z, (y, \text{val}_y) \in in[n], (z, \text{val}_z) \in in[n]\}$

## Reality Check!

### Some definitions and uses are ambiguous

- We can't tell whether or what variable is involved  
*e.g.*, `*p = x;` `/* what variable are we assigning?! */`
- Unambiguous assignments are called **strong updates**
- Ambiguous assignments are called **weak updates**

### Solutions

- Be conservative
  - Sometimes we assume that it could be everything  
*e.g.*, Defining `*p` (generating reaching definitions)
  - Sometimes we assume that it is nothing  
*e.g.*, Defining `*p` (killing reaching definitions)
- Try to figure it out: alias/pointer analysis (more later)

## Context

### Goals

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

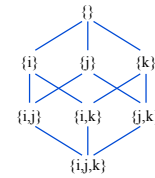
### Approach

- 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

## 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 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

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

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

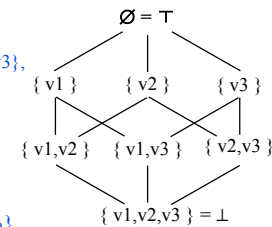
- Meet ( $\sqcap$ ):  $U$

- $\sqsubseteq$ :  $\supseteq$

- 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

## More Examples

### Reaching definitions

- $V: 2^S$  ( $S$  = set of all defs)
- $\sqcup$
- $\sqsubseteq$
- Top( $\top$ ):  $\emptyset$
- Bottom ( $\perp$ ):  $U$
- $F: \dots$

### Reaching Constants

- $V: 2^{v \times c}$ , variables  $v$  and constants  $c$
- $\sqcap$
- $\sqsubseteq$
- Top( $\top$ ):  $U$
- Bottom ( $\perp$ ):  $\emptyset$
- $F: \dots$

## Tuples of Lattices

### Problem

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

### Solution

- Use a tuple of lattices, one per variable

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

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

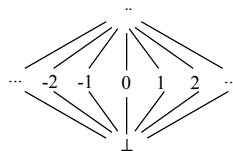
## Tuples of Lattices Example

### 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?

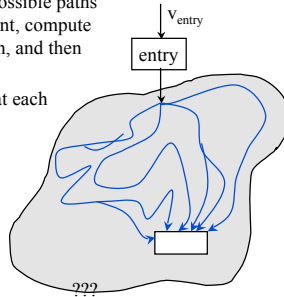
### Powerset lattice ( $2^S$ )

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

## 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
- $\sqcap_{\text{all paths } n_1, n_2, \dots, n_i} (f_{n_1}(\dots f_{n_2}(f_{n_1}(v_{\text{entry}}))))$



CS553 Lecture

Lattice Theoretic Framework for DFA

14

## 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?

CS553 Lecture

Lattice Theoretic Framework for DFA

15

## Correctness

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

### Look at Merges

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

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

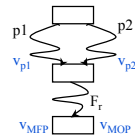
$$v_{\text{MFP}} \sqsubseteq v_{\text{MOP}} = F_r(v_{p1} \sqcap v_{p2}) \sqsubseteq F_r(v_{p1}) \sqcap F_r(v_{p2})$$

### Observation

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

$$f(x \sqcap y) \sqsubseteq f(x) \sqcap f(y) \Leftrightarrow x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$$

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



CS553 Lecture

Lattice Theoretic Framework for DFA

16

## Monotonicity

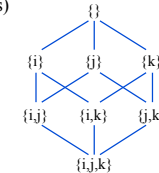
**Monotonicity:**  $(\forall x, y \in V)[x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)]$

- 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



CS553 Lecture

Lattice Theoretic Framework for DFA

17

## 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

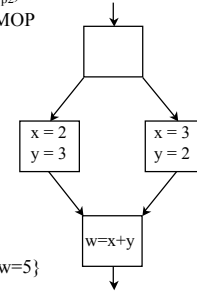
- $f(u \sqcap v) = f(u) \sqcap f(v)$
- $v_{MFP} \sqsubseteq v_{MOP} \equiv F_i(v_{p1} \sqcap v_{p2}) \sqsubseteq F_i(v_{p1}) \sqcap F_i(v_{p2})$
- If the flow functions are distributive, MFP = MOP

### Examples

- Reaching definitions?
- Reaching constants?

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

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



## 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 (monotonic)
- Efficient
- Accurate (distributive)