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 $\sqcap$
- $x, y \in V \Rightarrow x \sqcap y \in V$
- $x, y \in V \Rightarrow x \sqcap y = y \sqcap x$ (commutativity)
- $x, y, z \in V \Rightarrow (x \sqcap y) \sqcap z = x \sqcap (y \sqcap z)$ (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)
- $\forall x \in V - \{\top\}, x \subseteq \top$

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

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
- ⊤ represents “best-case” information (initial flow value)
  - e.g., Empty set
- ⊥ represents “worst-case” information
  - e.g., Universal set
- ⋈ (meet) merges flow values
  - e.g., Set union
- If x ⊑ 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**

\[ x = y \quad \text{and} \quad \{ x, y \} \]

**Finally**

\[ x = y \quad \text{and} \quad \{ y \} \]

---

Data-Flow Analysis Frameworks

**Data-flow analysis framework**
- A set of flow values (V)
- A binary meet operator (⊓)
- 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
\( U = \{ v1, v2, v3 \} \)

- \( V: 2^U = \{ \{ v1, v2, v3 \}, \{ v1, v2 \}, \{ v1, v3 \}, \{ v2, v3 \}, \{ v1 \}, \{ v2 \}, \{ v3 \}, \emptyset \} \)
- \( \emptyset = \top \)
- ⊤; \quad \emptyset
- Top(\( T \)): \quad \emptyset
- Bottom (\( \bot \)): \quad U
- \( F: \{ f(X) = \text{Gen}_U (X - \text{Kill}_U) \cup \forall \} \)

Inferior solutions are lower on the lattice
More conservative solutions are lower on the lattice
Recall Liveness Analysis

Data-flow equations for liveness

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

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

Liveness equations in terms of Gen and Kill

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

A use of a variable generates liveness

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

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

- \( V: \ 2^S \) (set of all defs)
- \( \top: \ \cup \)
- \( \bot: \ \emptyset \)
- \( \top(\top): \ \emptyset \)
- \( \bot(\bot): \ \cup \)
- \( F: \ \ldots \)

Reaching Constants

- \( V: \ 2^w, \text{variables v and constants c} \)
- \( \top: \ \cap \)
- \( \bot: \ \emptyset \)
- \( \top(\top): \ \emptyset \)
- \( \bot(\bot): \ \cup \)
- \( F: \ \ldots \)

Direction of Flow

Backward data-flow analysis

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

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

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

Forward data-flow analysis

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

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

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

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

<table>
<thead>
<tr>
<th>Live Variables</th>
</tr>
</thead>
</table>

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

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

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

\[\text{in}[n] = \bigcup \text{out}[s]\]

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

Why?

When might this be inappropriate?

Available Expressions (cont)

Data-Flow Equations

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

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

Plug it in to our general DFA algorithm

for each node n

\[\text{in}[n] = \nu; \text{out}[n] = \nu\]

repeat

for each n

\[\text{in}'[n] = \text{in}[n]\]

\[\text{out}'[n] = \text{out}[n]\]

\[\text{in}[n] = \bigcap \text{out}[p]\]

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

untill \[\text{in}'[n] = \text{in}[n]\] and \[\text{out}'[n] = \text{out}[n]\] for all n

Available Expressions Example

What is the initial guess?

\[s1: \ a = 3\]

\[s2: b = a + 2\]

\[s3: \ c = 3\]

What is the meet operation?

\[s4: c = c + 1\]

\[s5: \text{if}(p > q)\]

\[s6: c = c + 1\]

\[s7: r = a * b\]

Reaching Defs Example

What is the initial guess?

\[s1: \ a = 3\]

\[s2: b = a + 2\]

\[s3: \ c = 3\]

What is the meet operation?

\[s4: c = c + 1\]

\[s5: \text{if}(p > q)\]

\[s6: c = c + 1\]

\[s7: r = a * b\]

What does the lattice look like?
**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
- $\bigwedge_{n_1, n_2, \ldots, n_i} (f_{n_1}(f_{n_2}(...f_{n_i}(v_{\text{entry}}))))$

**Correctness**

"Is $v_{\text{MFP}}$ correct?" $\equiv$ "Is $v_{\text{MFP}} \subseteq v_{\text{MOP}}$?"

**Look at Merges**
- $v_{\text{MOP}} = F(v_{p_1}) \sqcap F(v_{p_2})$
- $v_{\text{MFP}} = F(v_{p_1} \sqcap v_{p_2})$
- $v_{\text{MFP}} \sqsubseteq v_{\text{MOP}} = F(v_{p_1} \sqcap v_{p_2}) \sqsubseteq F(v_{p_1}) \sqcap F(v_{p_2})$

**Observation**

$\forall x, y \in V$

$f(x \sqcap y) \subseteq f(x) \sqcap f(y) \iff x \subseteq y \Rightarrow f(x) \subseteq f(y)$

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

**Monotonicity**

**Monotonicity:** $(\forall x, y \in V)(x \subseteq y \Rightarrow f(x) \subseteq f(y))$
- 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**
- \(f(u \sqcap v) = f(u) \sqcap f(v)\)
- \(v_{\text{MFP}} \sqsubseteq v_{\text{MOP}} \equiv F_r(v_{p1} \sqcap v_{p2}) \sqsubseteq F_r(v_{p1}) \sqcap F_r(v_{p2})\)
- If the flow functions are distributive, \(\text{MFP} = \text{MOP}\)

**Examples**
- Reaching definitions?
- Reaching constants?
  
  \[
  \begin{align*}
  f(u \sqcap v) &= f(\{x=2,y=3\} \sqcap \{x=3,y=2\}) \\
  &= f(\emptyset) = \emptyset \\
  f(u) \sqcap f(v) &= f(\{x=2,y=3\}) \sqcap f(\{x=3,y=2\}) \\
  &= \{x=2,y=3,w=5\} \sqcap \{x=2,y=3,w=5\} = \{w=5\} \\
  \Rightarrow \text{MFP} &\neq \text{MOP}
  \end{align*}
  \]

---

**Tuples of Lattices**

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

**Solution**
- Use a tuple of lattices, one per variable

\[
L = (V, \sqcap) \equiv (L_1, \sqcap_1) \times (L_2, \sqcap_2) \times \cdots \times (L_n, \sqcap_n)
\]

- \(V = (V_1) \times (V_2) \times \cdots \times (V_n)\)
- Meet (\(\sqcap\)): point-wise application of \(\sqcap_i\)
- \((\ldots, v_i, \ldots) \sqsubseteq (\ldots, u_i, \ldots) \equiv v_i \sqsubseteq u_i, \forall i\)
- Top (\(\top\)): tuple of tops (\(\top_1, \top_2, \ldots, \top_n\))
- Bottom (\(\bot\)): tuple of bottoms (\(\bot_1, \bot_2, \ldots, \bot_n\))
- Height (\(L\)) = \(N \times \text{height}(L_i)\)

---

**Tuples of Lattices Example**

**Reaching constants (previously)**
- \(P = v \times c\), for variables \(v\) & constants \(c\)
- \(V: 2^P\)

**Alternatively**
- \(V = c \cup \{\top, \bot\}\)

The whole problem is a tuple of lattices, one for each variable.
Examples of Lattice Domains

Two-point lattice (\( \top \) and \( \bot \))
- Examples?
- Implementation?

Set of incomparable values (and \( \top \) and \( \bot \))
- Examples?

Powerset lattice (\( 2^S \))
- \( \top = \emptyset \) and \( \bot = S \), 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