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 \( \sqcap \)
- \( x, y \in V \Rightarrow x \sqcap y \in V \) (closure)
- \( x \in V \Rightarrow x \sqcap x = x \) (idempotence)
- \( x, y \in V \Rightarrow x \sqcap y = y \sqcap x \) (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)
- \( \forall x \in V - \{ \top \}, x \sqsubseteq \top \)

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

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
- $\bot$ represents “worst-case” information
  - e.g., Universal set
- $\sqcap$ (meet) merges flow values
  - e.g., Set union
- If $x \subseteq y$, 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
\( U = \{ v1, v2, v3 \} \)

- \( V: 2^S = \{ \{ v1,v2,v3 \}, \{ v1,v2 \}, \{ v1,v3 \}, \{ v2,v3 \}, \{ v1 \}, \{ v2 \}, \{ v3 \}, \emptyset \} \)
- Meet \( (\sqcap) \): \( U \)
- \( \sqcup: \emptyset \)
- Top(\( T \)): \( \emptyset \)
- Bottom(\( \bot \)): \( U \)
- \( F: f_n(X) = \text{Gen}_n \cup (X - \text{Kill}_n), \forall n \)
### Lattice Example

What are the data-flow sets 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**

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

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

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

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

**Can define many data-flow analysis in terms of Gen and Kill**
Reaching Constants (aka Constant Propagation)

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

Direction of Flow

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

\[
\begin{align*}
\text{in}[n] &= \text{gen}[n] \bigcup_{s \in \text{succ}[n]} (\text{out}[n] - \text{kill}[n]) \\
\text{out}[n] &= \bigcup_{p \in \text{pred}[n]} \text{out}[p]
\end{align*}
\]

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

\[
\begin{align*}
\text{in}[n] &= \bigcup_{p \in \text{pred}[n]} \text{out}[p] \\
\text{out}[n] &= \text{gen}[n] \bigcup (\text{in}[n] - \text{kill}[n])
\end{align*}
\]

**Some problems need both forward and backward analysis**
- *e.g.*, Partial redundancy elimination (uncommon)
More Examples

Reaching definitions

- \( V: 2^S \) (\( S \) = set of all defs)
- \( \cap: \cap \)
- \( \subseteq: \subseteq \)
- \( \text{Top}(\top): \emptyset \)
- \( \text{Bottom}(\bot): \emptyset \)
- \( F: \ldots \)

Reaching Constants

- \( V: 2^\infty \), variables v and constants c
- \( \cap: \cap \)
- \( \subseteq: \subseteq \)
- \( \text{Top}(\top): \bigcup \)
- \( \text{Bottom}(\bot): \emptyset \)
- \( F: \ldots \)

Merging Flow Values

Live variables and reaching definitions

- Merge flow values via set union

<table>
<thead>
<tr>
<th>Reaching Definitions</th>
<th>Live Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{in}[n] = \bigcup_{p \in \text{pred}[n]} \text{out}[s] )</td>
<td>( \text{out}[n] = \bigcup_{s \in \text{succ}[n]} \text{in}[s] )</td>
</tr>
<tr>
<td>( \text{out}[n] = \text{gen}[n] \bigcup (\text{in}[n] – \text{kill}[n]) )</td>
<td>( \text{in}[n] = \text{gen}[n] \bigcup (\text{out}[n] – \text{kill}[n]) )</td>
</tr>
</tbody>
</table>

Why?

When might this be inappropriate?
**ReachingDefs Example**

What is the lattice?

What is the initial guess?

What is the meet operation?

---

**Available Expressions Iterative Algorithm**

**Data-Flow Equations**

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

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

Plug it in to our general DFA algorithm

```plaintext
for each node n
    in[n] = U; out[n] = U
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
```
Available Expressions Example

What is the initial guess?

What is the meet operation?

What does the lattice look like?

Solving Data-Flow Analyses

Goal

– For a forward problem, consider all 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
  \[ \sqcap_{\text{all paths } n_1, n_2, \ldots, n_i} (f_{n_i}(\ldots f_{n_2}(f_{n_1}(v_{\text{entry}})))) \]
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 \( v_{MFP} \) correct?” = “Is \( v_{MFP} \subseteq v_{MOP} \)?”

Look at Merges

\[
\begin{align*}
    v_{MOP} &= F_r(v_{p1}) \cap F_r(v_{p2}) \\
    v_{MFP} &= F_r(v_{p1} \sqcap v_{p2}) \\
    v_{MFP} \subseteq v_{MOP} &= F_r(v_{p1} \sqcap v_{p2}) \subseteq F_r(v_{p1}) \cap F_r(v_{p2})
\end{align*}
\]

Observation

\[
\forall x, y \in V \\
    f(x \sqcap y) \subseteq f(x) \sqcap f(y) \iff x \subseteq y \Rightarrow f(x) \subseteq f(y)
\]

\[\therefore v_{MFP} \text{ correct when } F_r \text{ (really, the flow functions) are monotonic}\]
**Monotonicity**

**Monotonicity:** \((\forall x, y \in V) [x \leq y \Rightarrow f(x) \leq f(y)]\)
- 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?
Reaching Defs Example

What is the height of the lattice?

How many passes over the nodes are necessary?

What if we visit the nodes in a non-optimal order?

Accuracy

Distributivity

- \( f(u \sqcap v) = f(u) \sqcap f(v) \)
- \( v_{\text{MFP}} \subseteq v_{\text{MOP}} = F_r(v_{p1} \sqcap v_{p2}) \subseteq F_r(v_{p1}) \sqcap F_r(v_{p2}) \)
- If the flow functions are distributive, \( \text{MFP} = \text{MOP} \)

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

\[ L = (V, \sqcap) = (L_T = (V_T, \sqcap_T))^N \]
– \( V = (V_T)^N \)
– Meet (\( \sqcap \)): point-wise application of \( \sqcap_T \)
– \((\ldots, v_i, \ldots) \sqsubseteq (\ldots, u_i, \ldots) \iff v_i \sqsubseteq u_i, \forall i \)
– Top (\( \top \)): tuple of tops (\( \top_T \))
– Bottom (\( \bot \)): tuple of bottoms (\( \bot_T \))
– Height (\( L \)) = \( N \times \text{height}(L_T) \)

Tuples of Lattices Example

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

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

The whole problem is a tuple of lattices, one for each variable
**Tuple of Lattices example**

For reaching constants, how big is the tuple for this example?

```
- s1: a = 3
- s2: b = a + 2
- s3: c = freed()
- s4: c = c + 1
- s5: if (c > a)
- s6: c = c + 1
- s7: r = a * b
```

**Examples of Lattice Domains**

**Two-point lattice** (⊤ and ⊥)
- Examples?
- Implementation?

**Set of incomparable values** (and ⊤ and ⊥)
- Examples?

**Powerset lattice** ($2^S$)
- ⊤ = ∅ and ⊥ = 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