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

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

Defining Available Expressions Analysis

Must or may Information?
- Must

Direction?
- Forward

Flow values?
- Sets of expressions

Initial guess?
- Universal set

Kill?
- Set of expressions killed by statement $s$

Gen?
- Set of expressions evaluated by $s$

Merge?
- Intersection

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
- \( \bot \) represents “worst-case” information
  - e.g., Universal set
- \( \cap \) (meet) merges flow values
  - e.g., Set union
- If \( x \subseteq y \), then \( x \) is a conservative approximation of \( y \)
  - e.g., Superset

\[ S = \{ \{v1,v2,v3\}, \{v1,v2\}, \{v1,v3\}, \{v2,v3\}, \{v1\}, \{v2\}, \{v3\}, \emptyset \} \]

\[ \Omega = \{ v1, v2, v3 \} \]

**Data-Flow Analysis Frameworks**

**Data-flow analysis framework**
- A set of *flow values* (V)
- A binary *meet operator* (\( \cap \))
- 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

\[ \Omega = \{ v1, v2, v3 \} \]

\[ V = 2^S = \{ \{v1,v2,v3\}, \{v1,v2\}, \{v1,v3\}, \{v2,v3\}, \{v1\}, \{v2\}, \{v3\}, \emptyset \} \]

- \( \cap \): \( \Omega \)
- \( \cup \)
- Top(\( \top \)): \( \emptyset \)
- Bottom (\( \bot \)): \( \Omega \)
- \( f \): \( \{ f_n(X) = Gen_n(X - Kill_n) \} \)

Inferior solutions are lower on the lattice
More conservative solutions are lower on the lattice
**More Examples**

<table>
<thead>
<tr>
<th>Reaching definitions</th>
<th>Reaching Constants</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V: \mathcal{2}^S ) (( S = \text{set of all defs} ))</td>
<td>( V: \mathcal{2}^{\infty}, \text{variables } v \text{ and } \text{constants } c )</td>
</tr>
<tr>
<td>( \cap: \mathcal{U} )</td>
<td>( \cap: \mathcal{U} )</td>
</tr>
<tr>
<td>( \cup: \emptyset )</td>
<td>( \cup: \emptyset )</td>
</tr>
<tr>
<td>Top (( \top )): ( \emptyset )</td>
<td>Top (( \top )): ( \mathcal{U} )</td>
</tr>
<tr>
<td>Bottom (( \bot )): ( \mathcal{U} )</td>
<td>Bottom (( \bot )): ( \emptyset )</td>
</tr>
<tr>
<td>F: \ldots</td>
<td>F: \ldots</td>
</tr>
</tbody>
</table>

**Tuples of Lattices**

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

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

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

**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

**Tuples of Lattices Example**

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

**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
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
- \( \bigwedge \) all paths \( n_1, n_2, \ldots, n_i \)

\[ f_n \circ \cdots \circ f_1 (v_{\text{entry}}) \]

---

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 \( v_{\text{MFP}} \subseteq v_{\text{MOP}} \)?

**Look at Merges**
- \( v_{\text{MOP}} = F(v_{p_1}) \cap F(v_{p_2}) \)
- \( v_{\text{MFP}} = F(v_{p_1} \cap v_{p_2}) \)
- \( v_{\text{MFP}} \subseteq v_{\text{MOP}} = F(v_{p_1} \cap v_{p_2}) \subseteq F(v_{p_1}) \cap F(v_{p_2}) \)

**Observation**
- \( \forall x, y \in V \)
  - \( x \sqsubseteq y \implies f(x) \sqsubseteq f(y) \)
- \( v_{\text{MFP}} \) legal when \( F \) (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**
- \( f(u ▩ v) = f(u) ▩ f(v) \)
- \( v_{MFP} ▩ v_{MOP} = F_u(v_{MOP} ▩ v_{MFP}) \subseteq F_u(v_{MFP}) ▩ F_u(v_{MOP}) \)
- If the flow functions are distributive, \( MFP = MOP \)

**Examples**
- Reaching definitions?
- Reaching constants?

\[
\begin{align*}
 f(u ▩ v) &= f(\{x=2,y=3\} ▩ \{x=3,y=2\}) \\
 &= f(\emptyset) = \emptyset \\
 f(u) ▩ f(v) &= f(\{x=2,y=3\}) ▩ f(\{x=3,y=2\}) \\
 &= \{x=2,y=3,w=5\} ▩ \{x=2,y=2,w=5\} = \{w=5\} \\
 \Rightarrow & \ MFP \neq MOP
\end{align*}
\]

---

### 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)