Data-flow Analysis Theory and Loop Transformations (take 2)

Announcements
- PA2 due Monday, have enough information to do it

Today
- Tuples of lattices and CSE
- Why iterative solutions to data-flow analysis converge
- Intro to loop transformations
**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**)

**Note:**
- Really a semi-lattice, full lattice would have a join operator to go up

**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 $V$
- $x \sqsubseteq y \iff x \sqcap y = x$

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 \( x \sqsubseteq y \), then \( x \) is a conservative approximation of \( y \)
  - e.g., Superset

Note: PL semantics and abstract interpretation use the reverse convention.
Typical Lattices in Dataflow Analysis

**Powerset lattice:** set of all subsets of a set U
- meet operator ($\cap$) is union ($\cup$) or intersection ($\cap$)
- Partial ordering ($\subseteq$) is $\supseteq$ or
- Bottom ($\bot$) and Top ($\top$) are U and $\emptyset$, or vice versa
- Height = $|U|$ (infinite if U is infinite)

Set of unordered values plus top and bottom
- Example: Reaching constants domain for a particular variable
- Height = 2 (width may be infinite)

Two-point lattice: top and bottom
- Represents a boolean property
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 $n \ x = y + z$
  – $\text{kill}[n] = \{(x,c) | \forall c\}$
  – $\text{gen}[n] = \{(x,c) | c=\text{val}(y)+\text{val}z, (y, \text{val}y) \in \text{in}[n], (z, \text{val}z) \in \text{in}[n]\}$
Tuples of Lattices

Problem
  – Simple analyses may require complex lattices (e.g., Reaching constants)

Possible Solutions for reaching constants
  – Tuple of lattices, (variable, constant) tuples
  – Tuple of lattices, one entry in tuple per variable

\[ L = (V, \sqcap) \equiv (L_i = (V_i, \sqcap_i))^N \]
  – \( V = V_1 \times V_2 \times \ldots \times V_N \)
  – Meet (\( \sqcap \)): point-wise application of \( \sqcap_T \)
  – \((..., v_i, ...) \sqsubseteq (..., u_i, ...) \equiv v_i \sqsubseteq u_i, \forall i \)
  – Top (\( \top \)): tuple of tops \( (\top_i)^N \)
  – Bottom (\( \bot \)): tuple of bottoms \( (\bot_i)^N \)
  – Height \( (L) = \text{height}(L_1) \times \text{height}(L_2) \times \ldots \times \text{height}(L_N) \)

Equivalence of Power Set Lattices and Tuple of two-point lattices (bitvectors)
Tuples of Lattices for Reaching Constants

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

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

The whole problem is a tuple of lattices, one for each variable
Reaching Constants, Various Ways to do Tuple of Lattices

Reaching Constants

- $V$: $C \times C \times \ldots \times C$
- $\cap$: $\top \cap c = c$
- $\sqsubseteq$: if $c_i = k_i$ then $c_i$ else $\perp$
- $\bot$ under any constant
- $\top$ (Top): $\top$
- $\bot$ (Bottom): $\bot$
- $F$: $\ldots$

Reaching Constants

- $V$: $2^{v \times C}$, variables $v$ and constants $C$
- $\cap$
- $\sqsubseteq$
- $\top$ (Top): $\top$
- $\bot$ (Bottom): $\bot$
- $F$: $\ldots$
CSE in SSA and LLVM (PA2) Another Suggestion

Available Expressions: Possible approach

– Going to miss out on some possibilities (see example to the right)
– Could use lattice like better formulation of reaching definitions from lecture06
  – Lattice values are sets of (expression, instruction) pairs
  – Intersection for meet
– Consequence: Same expression available from two different instructions will not end up available

Lattice Theoretic Framework Approach

– Effect of instruction n: \( x = e \)
  – \( \text{kill}[n] = \{(e, d) | \forall d\} \)
  – \( \text{gen}[n] = \{(e, n)\} \)

\[
\begin{align*}
i.1 & := j.1 + x \\
a.1 & := 4 \times i.1
\end{align*}
\]

\[
\begin{align*}
i.2 & := i.1 + 1 \\
b & := 4 \times i.2
\end{align*}
\]

\[
\begin{align*}
i.3 & := \text{phi}(i.1, i.2) \\
c & := 4 \times i.3 \\
d & := j.1 + x
\end{align*}
\]
## PA2: Common Subexpression Elimination

<table>
<thead>
<tr>
<th>Available Expressions (non-SSA)</th>
<th>Reaching Expressions (SSA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>– V: (2^S) (S = set of all expressions)</td>
<td>– V: (2^{</td>
</tr>
<tr>
<td>– (\cap): (\cap)</td>
<td>– (\cap): (\cap)</td>
</tr>
<tr>
<td>– (\subseteq): (\subseteq)</td>
<td>– (\subseteq): (\subseteq)</td>
</tr>
<tr>
<td>– Top((\top)): (U)</td>
<td>– Top((\top)): (U)</td>
</tr>
<tr>
<td>– Bottom ((\bot)): (\emptyset)</td>
<td>– Bottom ((\bot)): (\emptyset)</td>
</tr>
<tr>
<td>– F: (\ldots)</td>
<td>– F: (\ldots)</td>
</tr>
<tr>
<td>– Start value: (\emptyset)</td>
<td>– Start value: (\emptyset)</td>
</tr>
</tbody>
</table>
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

– $\square_{\text{all paths } n_1, n_2, \ldots, n_i} (f_{n_i}(\ldots f_{n_2}(f_{n_1}(v_{\text{entry}}))))$
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?” \iff “Is \( v_{MFP} \subseteq v_{MOP} \)?”

Look at Merges

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

Observation

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

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

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

- 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

\[
\begin{align*}
\{&\} \\
\{&i\} & \{&j\} & \{&k\} \\
\{&i,j\} & \{&i,k\} & \{&j,k\} \\
\{&i,j,k\}
\end{align*}
\]
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?

What if we use a tuple of boolean lattices (bit-vector)?
### Accuracy

#### Distributivity

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

#### Examples

- Reaching definitions?
- Reaching constants?

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

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

Lattice Theoretic framework for common subexpression elimination

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

Stencil Computations

- Used to solve partial differential equations, in graphics, and cellular automata
- Computations operate over some mesh or grid
- Computation is modifying the value of something over time or as part of a relaxation to find steady state
- Each computation has some nearest neighbor(s) data dependence pattern
- The coefficients multiplied by neighbor can be constant or variable

1D data, 1D time Stencil Computation version 1 <demo in class>

// assume A[0,i] initialized to some values
for (t=1; t<(T+1); t++) {
    for (i=1; i<(N-1); i++) {
    }
}

CS 553 Stencils and Loop Transformations 20
1D Stencil Computation (take 2)

1D data, 2D time Stencil Computation, version 2 <demo in class>

// assume A[i] initialized to some values
for (t=0; t<T; t++) {
    for (i=1; i<(N-1); i++) {
    }
}

Analysis
– Are version 1 and version 2 computing the same thing?
– What is the operational intensity of version 1 versus version 2?
– What parallelism is there in version 1 versus version 2?
– Where is the data reuse in version 1 versus version 2?
Jacobi in SWM code (Stencil Computation with Explicit Weights)

Source: David Randall’s research group

DO  j = 2,jm-1
    DO  i = 2,im-1
        x2(i,j) = area_inv(i,j)* &
            ((x1(i+1,j)-x1(i,j))*laplacian_wghts(1,i,j)+ &
             (x1(i+1,j+1)-x1(i,j))*laplacian_wghts(2,i,j)+ &
             (x1(i  ,j+1)-x1(i,j))*laplacian_wghts(3,i,j)+ &
             (x1(i-1,j  )-x1(i,j))*laplacian_wghts(4,i,j)+ &
             (x1(i-1,j-1)-x1(i,j))*laplacian_wghts(5,i,j)+ &
             (x1(i  ,j-1)-x1(i,j))*laplacian_wghts(6,i,j))
    ENDDO
ENDDO