Announcements

- Homework is due Thursday (may extend)
  - Problem 6(b) is Extra credit
  - Submit in Submitty
  - Lock your team. Max size is 3. Ideal size is 2

- I’m away on Thursday and Friday
  - No class on Thursday
  - Quiz 1 today at the end of class

Outline of Today’s Class

- Dataflow frameworks, conclusion
- MOP solution vs. MFP solution
- Non-distributive analyses
  - Constant propagation

Catch up: Definition of a Lattice 
$(L, \leq)$

- A lattice $L$ is a poset under $\leq$, such that every pair of elements has a glb (meet) and lub (join)

- A lattice need not contain a 0 or a 1 element
- A finite height lattice must contain 0 and 1
- Not every poset is a lattice
- If there is element $a$ such that $a \leq x$ for every $x$ in $L$, then $a$ is the 0 element of $L$
- If there is $a$ such that $x \leq a$ for every $x$ in $L$, then $a$ is the 1 element of $L$

Catch up: Is This Lattice?

$D = \{a, b, c\}$

Poset is $2^3$, $\leq$ is set inclusion

D = \{a, b, c\}

\{(a, b, c), (a, b), (a, c), (b, c), (a), (b), (c), \emptyset\}

Spring 20 CSCI 4450/6450, A Milanova


Catch up: Is This Poset a Lattice

$D = \{a, b, c\}$

\{(a, b, c), (a, b), (a, c), (b, c), (a), (b), (c), \emptyset\}

Spring 20 CSCI 4450/6450, A Milanova


Catch up: Is This a Lattice?

$D = \{a, b, c\}$

\{(a), (b), (c), (a, b), (a, c), (b, c), \emptyset\}

Spring 20 CSCI 4450/6450, A Milanova


Catch up: Is This Lattice?

$D = \{a, b, c\}$

\{(a, b, c), (a, b), (a, c), (b, c), (a), (b), (c), \emptyset\}

Spring 20 CSCI 4450/6450, A Milanova


Catch up: Is This Poset a Lattice

$D = \{a, b, c\}$

\{(a, b, c), (a, b), (a, c), (b, c), (a), (b), (c), \emptyset\}

Spring 20 CSCI 4450/6450, A Milanova


Catch up: Is This Lattice?

$D = \{a, b, c\}$

\{(a), (b), (c), (a, b), (a, c), (b, c), \emptyset\}

Spring 20 CSCI 4450/6450, A Milanova


Catch up: Is This Poset a Lattice

$D = \{a, b, c\}$

\{(a, b, c), (a, b), (a, c), (b, c), (a), (b), (c), \emptyset\}

Spring 20 CSCI 4450/6450, A Milanova


Catch up: Is This Lattice?

$D = \{a, b, c\}$

\{(a), (b), (c), (a, b), (a, c), (b, c), \emptyset\}

Spring 20 CSCI 4450/6450, A Milanova


Catch up: Is This Poset a Lattice

$D = \{a, b, c\}$

\{(a, b, c), (a, b), (a, c), (b, c), (a), (b), (c), \emptyset\}

Spring 20 CSCI 4450/6450, A Milanova


Catch up: Is This Lattice?

$D = \{a, b, c\}$

\{(a), (b), (c), (a, b), (a, c), (b, c), \emptyset\}

Spring 20 CSCI 4450/6450, A Milanova


Catch up: Is This Poset a Lattice

$D = \{a, b, c\}$

\{(a, b, c), (a, b), (a, c), (b, c), (a), (b), (c), \emptyset\}

Spring 20 CSCI 4450/6450, A Milanova


Catch up: Is This Lattice?

$D = \{a, b, c\}$

\{(a), (b), (c), (a, b), (a, c), (b, c), \emptyset\}

Spring 20 CSCI 4450/6450, A Milanova


Catch up: Is This Poset a Lattice

$D = \{a, b, c\}$

\{(a, b, c), (a, b), (a, c), (b, c), (a), (b), (c), \emptyset\}

Spring 20 CSCI 4450/6450, A Milanova


Catch up: Is This Lattice?

$D = \{a, b, c\}$

\{(a), (b), (c), (a, b), (a, c), (b, c), \emptyset\}

Spring 20 CSCI 4450/6450, A Milanova


Catch up: Is This Poset a Lattice

$D = \{a, b, c\}$

\{(a, b, c), (a, b), (a, c), (b, c), (a), (b), (c), \emptyset\}

Spring 20 CSCI 4450/6450, A Milanova


Catch up: Is This Lattice?

$D = \{a, b, c\}$

\{(a), (b), (c), (a, b), (a, c), (b, c), \emptyset\}

Spring 20 CSCI 4450/6450, A Milanova


Catch up: Is This Poset a Lattice

$D = \{a, b, c\}$

\{(a, b, c), (a, b), (a, c), (b, c), (a), (b), (c), \emptyset\}

Spring 20 CSCI 4450/6450, A Milanova


Catch up: Is This Lattice?

$D = \{a, b, c\}$

\{(a), (b), (c), (a, b), (a, c), (b, c), \emptyset\}

Spring 20 CSCI 4450/6450, A Milanova


Catch up: Is This Poset a Lattice

$D = \{a, b, c\}$

\{(a, b, c), (a, b), (a, c), (b, c), (a), (b), (c), \emptyset\}

Spring 20 CSCI 4450/6450, A Milanova


Catch up: Is This Lattice?

$D = \{a, b, c\}$

\{(a), (b), (c), (a, b), (a, c), (b, c), \emptyset\}

Spring 20 CSCI 4450/6450, A Milanova
A poset $C$ where for every pair of elements $c_1, c_2$ in $C$, either $c_1 \leq c_2$ or $c_2 \leq c_1$.
- E.g., $\emptyset \leq \{a\} \leq \{a,b\} \leq \{a,b,c\}$
- E.g., from the lattice $J$ as shown here,
  - $1 \leq 2 \leq 6 \leq 30$
  - $1 \leq 3 \leq 15 \leq 30$
- A lattice s.t. every ascending chain is finite, is said to satisfy the Ascending Chain Condition.

Lattices define property space
- Lattice properties lead to properties of the standard dataflow analysis solution procedure (the worklist algorithm, which we will study shortly)

A problem fits into the dataflow framework if
- its property space is a lattice $L$, $\leq$ that satisfies the Ascending Chain Condition
- its merge operator $V$ is the join of $L$
- its transfer function space $F$: $L \rightarrow L$ is monotone
- Thus, we can make use of a generic solution procedure, known as the worklist algorithm or the maximal fixpoint algorithm or the fixpoint iteration algorithm

Property space is the lattice of the subsets where
- $D$ is the set of all definitions in the program
- $\leq$ is the subset operation
- Join is set union $\cup$. (Merge in Reach is set union, merge operator is join of lattice, thus $\leq$ is subset.)
- Lattice has $\emptyset$ being $\emptyset$, and $1$ being $D$
- Lattice satisfies the Ascending Chain Condition
Catch up: *Reach* Lattice

\[ \mathcal{D} = \text{all definitions: } \{(x,1),(x,4),(a,3)\} \]

Poset is 2\(^n\), \(\leq\) is the subset relation

1. \(x = a \cdot b\)
2. \(y = a \cdot b\)
3. \(a = a + 1\)
4. \(x = a \cdot b\)
5. goto 3

---

Catch up: *Avail* Lattice

Property space is the lattice of the subsets where

- \(\mathcal{D}\) is the set of all expressions in the program
- \(\leq\) is *superset*
  - join operation of the lattice is set intersection, since the merge operator in *Avail* is set intersection

- Lattice has 0 being \(\mathcal{D}\), and 1 being \(\emptyset\)
- Lattice satisfies *Ascending Chain Condition*

---

Catch up: *Avail* lattice

\[ \mathcal{D} = \text{all expressions: } \{a \cdot b, a + 1, y \cdot z\} \]

Poset is 2\(^n\), \(\supseteq\) is the superset relation

1. \(x \leftarrow a \cdot b\)
2. if \(y \leq a \cdot b\)
3. \(a \leftarrow a + 1\)
4. \(x \leftarrow a \cdot b\)
5. goto 2

---

Catch up: Transfer Functions

- The transfer functions
  \[ \text{out}(j) = (\text{in}(j)) \cap \text{pres}(j) \cup \text{gen}(j) \]
  or \(\text{out}(j) = f_j(\text{in}(j))\)

- **The transfer functions:** \(f: L \rightarrow L\). Formally, function space \(F\) is such that
  1. \(F\) contains all \(f_j\)
  2. \(F\) contains the identity function \(\text{id}(x) = x\)
  3. \(F\) is closed under composition
  4. Each \(f\) is monotone

---

Catch up: Monotonicity

- \(F: L \rightarrow L\) is *monotone* if and only if:
  1. \(a, b \in L, f \in F\) then \(a \leq b \implies f(a) \leq f(b)\)
     or (equivalently):
  2. \(x, y \in L, f \in F\) then \(f(x) \vee f(y) \leq f(x \vee y)\)

- Theorem: Definitions (1) and (2) are equivalent.
  - Show that (1) implies (2)
  - Show that (2) implies (1)

---

Catch up: Distributivity

- \(F: L \rightarrow L\) is *distributive* if and only if
  \(x, y \in L, f \in F\) then \(f(x \vee y) = f(x) \vee f(y)\)

- Every distributive function is also monotone but not the other way around

- Distributivity is a very nice property!
Catch up: Monotonicity and Distributivity

- Is classical Reach distributive?
  - Yes
    - Key property: \( \text{pres}(j) \) and \( \text{gen}(j) \) are constants
  - To show distributivity we have to show:
    - For each \( j \) \( \left( (X \cup Y) \cap \text{pres}(j) \right) \cup \text{gen}(j) = (X \cap \text{pres}(j)) \cup (Y \cap \text{pres}(j)) \cup \text{gen}(j) \)

Monotone Dataflow Frameworks

- A problem fits into the dataflow framework if
  - its property space is a lattice \( L, \leq \) that satisfies the Ascending Chain Condition
  - its merge operator \( V \) is the join of \( L \) and
  - its transfer function space \( F: L \to L \) is monotone
- Thus, we can make use of a generic solution procedure, known as the worklist algorithm or the maximal fixpoint algorithm or the fixpoint iteration algorithm

Worklist Algorithm for Forward Dataflow Problems

/* Initialize to initial values; 1 is entry node of CFG */
in(1) = InitialValue;
in_{\text{Reach}}(1) = \text{UNDEF}
for m = 2 to n do
  in(m) = 0
  in_{\text{Reach}}(m) = \emptyset
W = \{1,2,\ldots,n\} /* put every node on the worklist */
while W \( \neq \emptyset \) do {
  remove j from W
  in(j) = V \left\{ \text{out}(i) \mid i \text{ is predecessor of } j \right\}
  if out(j) \leq in(j) then {
    out(j) = f_j(\text{in}(j))
    if out(j) changed then
      W = W \cup \{ k \mid k \text{ is successor of } j \}
  }
  W = W \cup \{ i \}
}

Worklist Algorithm for Forward Dataflow Problems (slightly different)

/* Initialize to initial values; 1 is entry node of CFG */
in(1) = InitialValue;
in_{\text{Reach}}(1) = \text{UNDEF} out(1) = f_1(\text{in}(1))
for m = 2 to n do
  in(m) = 0
  out(m) = f_m(0)
W = \{2,\ldots,n\} /* put every node but 1 on the worklist */
while W \( \neq \emptyset \) do {
  remove j from W
  in(j) = V \left\{ \text{out}(i) \mid i \text{ is predecessor of } j \right\}
  if out(j) changed then
    W = W \cup \{ k \mid k \text{ is successor of } j \}
}

Termination Argument

- Why does the algorithm terminate?
- Sketch of argument:
  - At each “phase”, at least one \( \text{out}(j) \) changes.
  - Monotonicity of \( f_j \) entails that change is up the chain: \( \text{out}(j) \geq \text{out}(j) \).
  - Since \( \text{out}(j) \) in \( L \), and \( L \) satisfies the Ascending Chain Condition, \( \text{out}(j) \) changes at most \( O(h) \) times where \( h \) is the height of the lattice \( L \).
**Precision Argument**

- Theorem: The algorithm computes the least solution of the dataflow equations.
  - Historically though, this solution is called the maximal fixpoint solution (MFP)
  - I.e., For every node \( j \), the worklist algorithm computes a solution \( MFP(j) = \{ \text{in}(j), \text{out}(j) \} \), such that every other solution \( \{ \text{in}'(j), \text{out}'(j) \} \) of the dataflow equations is \( \text{in}(j) \leq \text{in}'(j), \text{out}(j) \leq \text{out}'(j) \)

**Example**

\[
\begin{align*}
1. & \quad z := x + y \\
2. & \quad \text{if } (z > 500) \\
3. & \quad \text{skip}
\end{align*}
\]

\[
\begin{align*}
\text{in}_{\text{avail}}(2) &= \text{out}_{\text{avail}}(1) \\
\text{in}_{\text{avail}}(3) &= \text{out}_{\text{avail}}(2) \\
\text{in}_{\text{avail}}(1) &= \emptyset \\
\text{out}_{\text{avail}}(2) &= \text{in}_{\text{avail}}(2) \\
\text{out}_{\text{avail}}(3) &= \text{in}_{\text{avail}}(3) \\
\text{out}_{\text{avail}}(1) &= (\text{in}_{\text{avail}}(1) - E_z) \cup \{x + y\}
\end{align*}
\]

Equivalent to: \( \text{in}_{\text{avail}}(2) = \{x + y\} \) and recall that \( \cap \) (i.e., set intersection).

**Question**

Willy Wazoo changed the worklist algorithm to initialize to 1.

\[
\begin{align*}
\text{in}(1) &= \text{InitialValue}; \\
\text{in}_{\text{avail}}(1) &= \emptyset \\
\text{for } m = 2 \text{ to } n & \text{ do } \text{in}(m) = 1 \\
 W &= \{1, 2, ..., n\}" \\
\text{while } W \neq \emptyset \text{ do } \{ \\
 & \quad \text{remove } j \text{ from } W \\
 & \quad \text{out}(j) = f_j(\text{in}(j)) \\
 & \quad \text{for } i \text{ in } \text{successors}(j) \quad \text{if } \text{out}(j) \subseteq \text{in}(i) \text{ then } \{ \\
 & \quad \quad \text{in}(i) = \text{out}(j) \cup \text{in}(i) \\
 & \quad \quad \text{W} = \text{W} \cup \{i\} \\
\} \\
\}\end{align*}
\]

1. Does Willy’s algorithm compute an over-approximation or an under-approximation of the MFP?

**Meet Over All Paths (MOP)**

- Desired dataflow information at \( n \) is obtained by traversing ALL PATHS from 1 (entry node) to \( n \).
  - For every path \( p = (1, n_2, n_3, ..., n_k) \) we compute \( f_{n_k}(...f_{n_2}(f_1(\text{init}(1)))) \)
  - The MOP at entry of \( n \) is \( V f_{n_k}(...f_{n_2}(f_1(\text{init}(1)))) \) over all paths \( p \) from 1 to \( n \)

**MOP vs. MFP**

- The MOP is an abstract model for the best solution computable with this kind of static analysis
  - It is a common assumption in this kind of static analysis that all program paths are executable
  - Abstract interpretation, axiomatic semantics, and symbolic execution abstract state more precisely, and can rule out some paths
  - The MFP is the solution computed by the worklist algorithm

**MOP vs. MFP**

- For distributive problems \( \text{MFP} = \text{MOP}! \)
  - Unfortunately, for monotone problems this is not true. But we still have a safe solution: it is a theorem that for monotone problems, \( \text{MFP} \geq \text{MOP} \)
Safety of a Dataflow Solution

- A safe (also, correct or sound) solution $X$ over approximates the “best” possible dataflow solution, i.e., $X \geq \text{MOP}$.
- Historically, an acceptable solution $X$ is one that is better than what we can do with the MFP, i.e., $X \leq \text{MFP}$.

Safe Solutions

- In many problems 1 is the universal set of facts, the merge operator is set union. It is safe to err by saying that a fact reaches a node when in fact it doesn’t.
- E.g., intuitively, it is safe to err by saying that a definition $(x,k)$ reaches a node, when in fact it MAY NOT REACH that node.
- Safe entails “larger” than the MOP under our partial order. Our definition of $\leq$ is subset inclusion (which is natural).

Safe Solutions: Reach

$D = \{ (x,1), (x,4), (a,3) \}$

Poset is $2^D$, $\subseteq$ is the subset relation.

1. $x = a \cdot b$
2. if $y = a \cdot b$
3. $a = a + 1$
4. $x = a \cdot b$
5. goto 3

Safe Solutions: Avail

$D = \{ a \cdot b, a + 1, y \cdot z \}$

Poset is $2^D$, $\supseteq$ is the superset relation.

1. $x = a \cdot b$
2. if $y \cdot z \leq a \cdot b$
3. $a = a + 1$
4. $x = a \cdot b$
5. goto 2

Precision of a Dataflow Solution

- Precise solution is one that is “close” to MOP.
  - A precise solution contains few spurious dataflow facts (spurious facts are what we call noise).
  - Unfortunately, for most problems even the MOP (an approximation itself) is undecidable.

  - $\text{MOP} \leq X \leq Y$: $X$ is more precise than $Y$.
  - We can compare two solutions $X$ and $Y$.
  - But, for most problems, we have no way of knowing the “ground truth.”
Outline of Today’s Class

- Dataflow frameworks
- MOP vs. MPF
- Non-distributive analyses
  - Constant propagation

Constant Propagation (Simple)

- Problem statement: What variables always hold constant values at a given program point

Example:

1. \( \text{in}(1): x \) is not const
   \( \text{out}(1): x = 1 \)

2. \( \text{in}(2): x = 1 \)
   \( \text{out}(2): x = 2 \)

3. \( \text{in}(3): x = 0 \)
   \( \text{out}(3): x = 1 \)

4. \( \text{in}(4): x \) is NOT a const!

Aside: Defining an Analysis

- Define program syntax
- In practice, we deal with a lot more than the simple abstraction
- Define property space
- The abstract program state that approximates the concrete program state
- Define transfer function space over syntax
- Abstract program execution over abstract state

Aside: Defining an Analysis

- If property space has desired properties
  - is a lattice \( L, \leq \) that satisfies the *Ascending Chain Condition*
  - merge operator \( V \) is the join of \( L \) and
- Function space \( F: L \rightarrow L \) is monotone then analysis fits the monotone dataflow framework and can be solved using the worklist algorithm

Constant Propagation: Syntax

\[ S ::= S; S \mid \text{while} (b) \{ S \} \mid \text{if} (b) \{ S \} \text{else} \{ S \} \mid x = V \mid x = V \text{Op} V \]
\[ \text{Op ::= + | - | * | /} \]
\[ V ::= x \mid y \mid z \mid C \]

- \( x, y, z \) are program variables
- \( C \) is constant
- We have to define transfer functions for \( x = V \) and \( x = V \text{Op} V \)

Constant Propagation: Property Space

- Associate one of the following values with variable \( x \) at each program point

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (or T)</td>
<td>x is NOT a constant</td>
</tr>
<tr>
<td>c</td>
<td>x has constant value c</td>
</tr>
<tr>
<td>0 (or ⊥)</td>
<td>x is unknown</td>
</tr>
</tbody>
</table>
Constant Propagation: Lattice

- Lattice $L_x \leq$
  
  \[
  \begin{align*}
  &2 &1 &0 &1 &2 \\
  \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
  1 & & & & & \\
  \end{align*}
  \]

- Dataflow lattice $L$ is the product of $L_x$
  - $1, 2 \in L$, $1 \leq 2$ if $1_x \leq 2_x$ for every variable $x$
  - $1 \lor 2$ amounts to $1_x \lor 2_x$ for every variable $x$
  - Merge operator is join of $L$

- Does the product lattice satisfy the ACC?

Constant Propagation: Transfer Functions

- $j: x = c$
  - $f_j$: kill $x \to \text{val}$, generate $x \to c$

- $j: x = y$
  - $f_j$: kill $x \to \text{val}$, add $x \to \text{val}'$, s.t. $y \to \text{val}'$ in $\text{in}(j)$.

- $\text{val}$ and $\text{val}'$ are one of
  - $\bot$: bottom (unknown)
  - $c$: constant
  - $T$: top (not a constant)

Example

1. if ($b > 0$)

2. $x = 1$
   $y = 2$

3. $x = 2$
   $y = 1$

4. $z = x + y$

5. $w = 10 \times z$

Not Distributive! A Counter Example

- $f_4(f_2(f_1(T)))$ implies $z \to 3$
- $f_4(f_3(f_1(T)))$ implies $z \to 3$

- Thus, $\text{MOP} \leq 5$
- $f_4(f_2(f_1(T))) \lor f_4(f_3(f_1(T)))$

- $\text{MFP} \leq 5$ gave us $z \to T$
  (i.e., $z$ is NOT a const)

Spring 20 CSCI 4450/6450, A Milanova