Dataflow Frameworks, cont.
Announcements

- Quiz 1 today
  - On four classical dataflow problems

- Homework due Thursday
  - Questions?
Outline of Today’s Class

- Dataflow framework
  - Lattices
  - Transfer functions
  - Worklist algorithm

- MOP solution vs. MFP solution

- Reading:
  - Dragon Book, Chapter 9.2 and 9.3
Lattice Theory

- **Partial ordering** (denoted by ≤ or ⊆)
  - Relation between pairs of elements
  - Reflexive $a ≤ a$
  - Anti-symmetric $a ≤ b$ and $b ≤ a$ $⇒$ $a = b$
  - Transitive $a ≤ b$ and $b ≤ c$ $⇒$ $a ≤ c$

- **Partially ordered set** (poset) (set $S$, $≤$)
  - 0 element $0 ≤ a$, for every $a$ in $S$
  - 1 element $a ≤ 1$, for every $a$ in $S$

We don’t necessarily need 0 or 1 element
Lattice Theory

- Greatest lower bound (glb)
  \( l_1, l_2 \) in poset \( S \), \( a \) in poset \( S \) is the \( \text{glb}(l_1, l_2) \) iff
  1) \( a \leq l_1 \) and \( a \leq l_2 \)
  2) for any \( b \) in \( S \), \( b \leq l_1, b \leq l_2 \) implies \( b \leq a \)

If glb exists, it is unique. Why? Called **meet** (denoted by \( \wedge \) or \( \sqcap \)) of \( l_1 \) and \( l_2 \).

- Least upper bound (lub)
  \( l_1, l_2 \) in poset \( S \), \( c \) in poset \( S \) is the \( \text{lub}(l_1, l_2) \) iff
  1) \( c \geq l_1 \) and \( c \geq l_2 \)
  2) for any \( d \) in \( S \), \( d \geq l_1, d \geq l_2 \) implies \( d \geq c \)

If lub exists, it is unique. Called **join** (denoted by \( \vee \) or \( \sqcup \)) of \( l_1 \) and \( l_2 \).
Definition of a Lattice \((L, \Lambda, V)\)

- 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 1 element
- A finite lattice must contain 0 and 1 elements
- 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 \(L\)
Is This Poset a Lattice

D = \{a, b, c\}
The poset is $2^D$, $\leq$ is set inclusion

$l_1 \leq l_2 \iff l_1 \subseteq l_2$

$\text{glb}(l_1, l_2) = l_1 \cap l_2$

$\text{lub}(l_1, l_2) = l_1 \cup l_2$
Thus: If $\text{glb}(l_1, l_2)$ exists then it is unique.

Suppose $a = \text{glb}(l_1, l_2)$ and $b = \text{glb}(l_1, l_2)$ and $a \neq b$.

By def of $\text{glb}$: 

$$a \leq l_1 \quad b \leq l_1$$

$$a \leq l_2 \quad b \leq l_2$$

Since $[a]$ is a glb of $l_1$ and $l_2$, any other lower bound $[b]$ is $b \leq a$.

Analogously, $a \leq b$.

By anti-symmetry $[a = b]$.
Examples of Lattices

- $H = (2^D, \cap, U)$ where $D$ is a finite set
  - Partial order is subset
  - $\text{glb}(s_1, s_2)$ denoted $s_1 \Lambda s_2$, is set intersection $s_1 \cap s_2$
  - $\text{lub}(s_1, s_2)$ denoted $s_1 V s_2$, is set union $s_1 U s_2$

- $J = (N_1, \gcd, \text{lcm})$
  - Partial order is integer division
  - $\text{lub}(n_1, n_2)$ denoted $n_1 V n_2$ is $\text{lcm}(n_1, n_2)$
  - $\text{glb}(n_1, n_2)$ denoted $n_1 \Lambda n_2$ is $\gcd(n_1, n_2)$
  ($N_1$ denotes natural numbers starting at 1)
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., $\{\} \leq \{a\} \leq \{a,b\} \leq \{a,b,c\}$
- 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 in Dataflow Analysis

- Lattices define property space

- Lattice properties give rise to certain properties of the worklist algorithm (standard way of solving dataflow problems)
Dataflow Lattices: \textit{Reach}

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

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

1. \(x = a \ast b\)
2. \(\text{if } y \leq a \ast b\)
3. \(a = a + 1\)
4. \(x = a \ast b\)
5. \(\text{goto 3}\)
Dataflow Lattices: \textit{Avail}

D = all expressions: \{a*b,a+1,y*z\}

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

1. $x := a*b$

2. if $y*z \leq a*b$

3. $a := a+1$

4. $x := a*b$

5. goto 2
Property Space

- Property space must be:
  1. A lattice $L, \leq$
  2. $L$ satisfies the *Ascending Chain Condition*
     
     Requires that all ascending chains are finite
Property Space

- **Merge operator** $V$ must be the join of $L$.
- In dataflow, $L$ is often the lattice of the subsets over a finite set of dataflow facts $D$.
  - Choose universal set $D$ (e.g., all definitions).
  - Choose ordering operation $\leq$. Since merge operator must be the join of $L$, a *may* problem sets $\leq$ to *subset* and a *must* problem sets $\leq$ to *superset*.
Example: *Reach* Lattice

- Property space is the lattice of the subsets
  - \( D \) is the set of all definitions in program
  - \( \leq \) is the *subset* operation
    - Thus, *join* is set union, as needed for *Reach*, which is a *may* problem

- Lattice has a 0 being \( \{\} \), and a 1 being \( D \)
- Lattice satisfies the *Ascending Chain Condition*
Example: *Avail* Lattice

- Property space is the lattice of the subsets

  - \( \mathcal{D} \) is the set of all expressions in the program
  - \( \leq \) is superset
    - Thus, join is set intersection, as needed for *Avail*, which is a *must* problem

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

- A problem fits into the dataflow framework if:
  - problem’s property space is a lattice \( L, \leq \) that satisfies the \textit{Ascending Chain Condition}
  - problem’s merge operator 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 \textit{worklist algorithm} (also the \textit{maximal fixpoint algorithm} or the \textit{fixpoint iteration algorithm})
Outline of Today’s Class

- Dataflow frameworks
  - Lattices
  - Transfer functions
  - Worklist algorithm

- MOP solution vs. MFP solution

- Reading:
  - Dragon Book, Chapter 9.2 and 9.3
Transfer Functions

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 \( f \) must be monotone

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

\[
\text{out}(j) = f_j(\text{in}(j))
\]

\( \gamma = f_j(x) \)
Monotonicity Property

- 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) \lor f(y) \leq f(x \lor y)

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

- Show that (1) implies (2)

\[ a \leq b \implies f(a) \leq f(b) \]

We need to show

\[ x, y \quad f(x) \lor f(y) \leq f(x \lor y) \]

\[ x \leq x \lor y \implies \begin{cases} f(x) \leq f(x \lor y) & \text{(by (1))} \\ f(y) \leq f(x \lor y) & \text{(by (1))} \end{cases} \]

\[ f(x \lor y) \text{ is an upper bound of } f(x) \text{ and } f(y) \]

Therefore

\[ f(x \lor y) \geq f(x) \lor f(y) \]

the least upper bound
Distributivity Property

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

- A distributive function is also monotone but not the other way around
  - Distributivity is a very nice property!
Monotonicity and Distributivity

Is classicalReach distributive?

- Yes

To show distributivity:

\[
\begin{align*}
\text{out}(j) &= (\text{in}(j) \cap \text{pres}(j)) \cup \text{gen}(j) \\
\text{out}(j) &= (\text{in}(j) \cap \text{pres}(j)) \cup \text{gen}(j)
\end{align*}
\]

For each \( j \):
\[
\begin{align*}
\mathcal{f}_j(X_1 \cup X_2) &= ( (X_1 \cup X_2) \cap \text{pres}(j) ) \cup \text{gen}(j) \\
\mathcal{f}_j(X_1) &= ( (X_1 \cap \text{pres}(j)) \cup \text{gen}(j) ) \cup ( (X_2 \cap \text{pres}(j)) \cup \text{gen}(j) ) \\
\mathcal{f}_j(X_2) &= ( (X_1 \cap \text{pres}(j)) \cup \text{gen}(j) ) \cup ( (X_2 \cap \text{pres}(j)) \cup \text{gen}(j) )
\end{align*}
\]
Monotone Dataflow Framework

- A problem fits into the dataflow framework if
  - problem’s **property space** is a lattice $\mathbb{L}$, $\leq$ that satisfies the *Ascending Chain Condition*
  - problem’s merge operator is the join of $\mathbb{L}$ and
  - its **transfer function** space $F: \mathbb{L} \to \mathbb{L}$ is monotone

- Thus, we can make use of a generic solution procedure, known as the **worklist algorithm**.
Worklist Algorithm for Forward Dataflow Problems

/* Initialize to initial values; 1 is entry node of CFG */
in(1) = InitialValue; out(1) = f_1(in(1))
for m = 2 to n do in(m) = 0; out(m) = f_m(0)
W = {2,…,n} /* put every node but 1 on the worklist */

while W ≠ Ø do {
    remove j from W
    in(j) = V \{ out(i) | i is predecessor of j \}
    out(j) = f_j(in(j))
    if out(j) changed then
        W = W U \{ k | k is successor of j \}
}
Worklist Algorithm on \textit{Reach}

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

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

1. \(x = a \times b\)

2. if \(y \leq a \times b\)

3. \(a = a + 1\)

4. \(x = a \times b\)

5. goto 3
Theorem: the algorithm terminates. Why?

Sketch of argument:

A node $k$ is placed on worklist only if the $\text{out}(j)$ of a predecessor $j$ changes. Monotonicity of $f$ guarantees $\text{in}^n(j) \leq \text{in}^{n+1}(j)$ and $\text{out}^n(j) \leq \text{out}^{n+1}(j)$. (Here $\text{in}^n(j)$, $\text{out}^n(j)$ are the sets at iteration $n$.)

$\text{in}$ and $\text{out}$ sets are elements of $L$ and $L$ satisfies the Ascending Chain Condition; thus, there is only a finite number of times each $\text{out}(j)$ changes.
Correctness Argument

Theorem: Worklist algorithm computes a solution that satisfies the dataflow equations. Why?

Sketch of argument:
Suppose either (1) $V_{out}(i) \neq in(j)$ or (2) $out(j) \neq f_j(in(j))$
For (1) to hold we must have “grown” $out(i)$ in some iteration and not added successor $j$ to worklist; this is impossible.
Theorem: Worklist algorithm computes the least solution of the dataflow equations.

Historically, solution computed by worklist algorithm is called the maximal fixpoint solution (MFP solution).

For every node $j$, worklist algorithm computes a solution $\text{MFP}(j) = (\text{in}(j), \text{out}(j))$, such that for every solution $(\text{in}'(j), \text{out}'(j))$ of the dataflow equations we have $\text{in}(j) \leq \text{in}'(j)$ and $\text{out}(j) \leq \text{out}'(j)$.
Example

1. \( z = x + y \)

2. if \( z > 500 \)

3. skip

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

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

Solution 1
- \( \emptyset \)
- \( \{x+y\} \)

Solution 2
- \( \emptyset \)
- \( \{x+y\} \)
Outline of Today’s Class

- Dataflow frameworks
  - Lattices
  - Transfer functions
  - Worklist algorithm

- MOP solution vs. MFP solution

- Reading:
  - Dragon Book, Chapter 9.2 and 9.3
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
  \[
  f_{n_k}(\ldots f_{n_2}(f_1(\text{Initial Value})))
  \]

- The MOP at entry of \( n \) is
  \[
  V f_{n_k}(\ldots f_{n_2}(f_1(\text{Initial Value})))
  \]
  over all paths \( p \) from 1 to \( n \)
MOP vs. MFP

- MOP is an abstraction of the best solution computable with dataflow analysis
  - It is a common assumption in dataflow analysis that *all program paths are executable*
- 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 $X$ overestimates 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: Reach

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

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

1. \(x = a \cdot b\)
2. if \(y \leq a \cdot b\)
3. \(a = a + 1\)
4. \(x = a \cdot b\)
5. goto 3
Safe Solutions: \textit{Avail}

U = all expressions: \{a*b, a+1, y*z\}

Poset is $2^U$, $\leq$ is the superset relation $\supseteq$

1. $x := a*b$
2. if $y*z \leq a*b$
3. $a := a+1$
4. $x := a*b$
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 is what we call noise)
  - Unfortunately, for most problems even the MOP (an approximation itself!) is undecidable

- MOP ≤ X ≤ Y: X is more precise than Y
  - Usually, we can compare two solutions X and Y
  - But, for most problems, we have no way of knowing the “ground truth”
Next class: real analyses

- Next time: non-distributive analyses
  - Constant propagation
  - Pointer analysis