Dataflow Analysis: Dataflow Frameworks
Outline of Today’s Class

- Catch up
- Dataflow frameworks
  - Lattices
  - Transfer functions (next time)
  - Worklist algorithm (next time)

- Reading:
  - Dragon Book, Chapter 9.2 and 9.3
Dataflow Analysis

1. Control-flow graph (CFG):
   - $G = (N, E, 1)$
   - Nodes are basic blocks

2. Data

3. Dataflow equations
   \[ \text{out}(j) = (\text{in}(j) - \text{kill}(j)) \cup \text{gen}(j) \]
   (\text{gen} and \text{kill} are parameters)

4. Merge operator $V$
   \[ \text{in}(j) = V \text{out}(i) \]
   \(i\) is predecessor of \(j\)
Problem 1. Reaching Definitions (Reach)

- Problem statement: for each CFG node \( n \), compute the set of definitions \( (x,k) \) that reach \( n \)

- First, define data (i.e., the dataflow facts) to propagate
  - Primitive dataflow facts are definitions \( (x,k) \)
  - Reach propagates sets of definitions, e.g., \( \{(i,1),(p,4)\} \)
Reaching Definitions (Reach)

- Next, define the dataflow equations (i.e., effect of code at node \( j \) on incoming dataflow facts)

\[ j: x = y + z \]

\[
\begin{align*}
\text{kill}(j): & \text{ all definitions of } (x, _) \\
\text{gen}(j): & \text{ this definition of } x, (x, j)
\end{align*}
\]

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

E.g., if \( \text{in}(4) = \{ (x, 1), (y, 2), (x, 3) \} \)

Node 4 is: \( x = y + z \)

Then \( \text{out}(4) = \{ (y, 2), (x, 4) \} \)
Reaching Definitions (*Reach*)

- Next, define the merge operator $V$ (i.e., how to combine data from incoming paths)
- For *Reach*, $V$ is the set union $\bigcup$

$$in(j) = \{ \bigcup \text{out}(i) \mid i \text{ is predecessor of } j \}$$

E.g., if $out(2) = \{(x,1), (y,2)\}$ and $out(3) = \{(x,3)\}$ and 2 and 3 are predecessors of 4

$$in(4) = \{(x,1), (x,3), (y,2)\}$$
Reach: Dataflow Equations

1. \( x = 5 \) 
   \[ \text{in}(1) = \emptyset \]
   \[ \text{out}(1) = (\text{in}(1) - D_x) \cup \{(x,1)\} \]

2. \( y = 1 \) 
   \[ \text{in}(2) = \text{out}(1) \]
   \[ \text{out}(2) = (\text{in}(2) - D_y) \cup \{(y,2)\} \]

3. \( x \geq 2 \)
   \[ \text{in}(3) = \text{out}(2) \cup \text{out}(6) \]
   \[ \text{out}(3) = \text{in}(3) \]

4. \( y = x \times y \)
   \[ \text{in}(4) = \text{out}(3) \]
   \[ \text{out}(4) = (\text{in}(4) - D_y) \cup \{(y,4)\} \]

5. \( x = x - 1 \)
   \[ \text{in}(5) = \text{out}(4) \]
   \[ \text{out}(5) = (\text{in}(5) - D_x) \cup \{(x,5)\} \]

6. goto 3
   \[ \text{in}(6) = \text{out}(5) \]
   \[ \text{out}(6) = \text{in}(6) \]

7. ...
   \[ \text{in}(7) = \text{out}(3) \]
Reach: Solution of Equations

1. $x = 5$
   - $\text{in}(1) = \emptyset$
   - $\text{out}(1) = \{(x,1)\}$

2. $y = 1$
   - $\text{in}(2) = \{(x,1)\}$
   - $\text{out}(2) = \{(x,1), (y,2)\}$

3. $x \geq 2$

4. $y = x \times y$
   - $\text{in}(3) = \{(x,1),(x,5),(y,2),(y,4)\}$
   - $\text{out}(3) = \{(x,1),(x,5),(y,2),(y,4)\}$

5. $x = x - 1$
   - $\text{in}(4) = \{(x,1),(x,5),(y,2),(y,4)\}$
   - $\text{out}(4) = \{(x,1),(x,5),(y,4)\}$

6. $\text{goto 3}$
   - $\text{in}(5) = \{(x,1),(x,5),(y,4)\}$
   - $\text{out}(5) = \{(x,5),(y,4)\}$

7. ...
   - $\text{in}(6) = \{(x,5),(y,4)\}$
   - $\text{in}(7) = \{(x,1),(x,5),(y,2),(y,4)\}$
Reaching Definitions

Forward, may dataflow problem
Problem 2. Live Uses of Variables (Live)

- We say that a variable $x$ is “live on exit from node $j$” if there is a live use of $x$ on exit from $j$ (recall the definition of “live use of $x$ on exit from $j$”)

- Problem statement: for each node $n$, compute the set of variables that are live on exit from $n$.

1. $x=2$; 2. $y=4$; 3. $x=1$; if $(y>x)$ then 5. $z=y$; else 6. $z=y^2$; 7. $x=z$.

What variables are live on exit from statement 3? Statement 1?
Live Example

1. \( x=2 \)

2. \( y=4 \)

3. \( x=1 \)

4. \( (y>x) \)

5. \( z=y \)

6. \( z=y\times y \)

7. \( x=z \)
Live Uses of Variables (*Live*)

- **Data**
  - Primitive facts: variables \( x \)
  - Propagates sets: \( \{x, y, z\} \)

- **Dataflow equations. At** \( j: x = y+z \)
  - \( \text{kill}_{LV}(j): \{x\} \)
  - \( \text{gen}_{LV}(j): \{y, z\} \)

- **Merge operator**: set union \( \cup \)
Live Uses of Variables (Live)

Problem statement: for each node $n$, compute the set of variables that are live on exit from $n$.

$$in_{LV}(j) = (out_{LV}(j) - kill_{LV}(j)) \cup gen_{LV}(j)$$

$$out_{LV}(j) = \{ \cup in_{LV}(i) \mid i \text{ is a successor of } j \}$$

Q: What are the primitive dataflow facts?
Q: What is $gen_{LV}(j)$?
Q: What is $kill_{LV}(j)$?
Problem 2: Live Uses of Variables

Backward, may dataflow problem

What are the primitive dataflow facts? Variables, e.g., $x, y, z$. Equations act on sets of variables.
Available Expressions

- An expression $x \ op \ y$ is available at program point $n$ if every path from entry to $n$ evaluates $x \ op \ y$, and there are NO subsequent assignments to $x$ or $y$ after evaluation and prior to reaching $n$. 

```
1

x op y
x = ...
|   |
x | op x
\|   \|
y = ...

x op y
x = ...
|   |
x | op x
\|   \|
y = ...

n
```

Spring 21 CSCI 4450/6450, A Milanova
Problem 3. Available Expressions (Avail)

Problem statement: For every node $n$, compute the set of expressions that are available at $n$. 

```
x op y
x = ...
y = ...
```

```
x op x
x = ...
y = ...
```

```
x op y
x = ...
y = ...
```

```
1
```

```
x op y
x = ...
y = ...
```

```
x op x
x = ...
y = ...
```

```
x op y
x = ...
y = ...
```

```
n
```
Avail Enables Global Common Subexpressions
Avail Enables Global Common Subexpressions

Can we eliminate \( w = a \times b \)?
Available Expressions (Avail)

Data?

- Primitive dataflow facts are expressions, e.g., \( x+y, a*b, a+2 \)
- Analysis propagates sets of expressions, e.g., \( \{x+y, a*b\} \)

Dataflow equations at \( j \): \( x = y \ op \ z \) ?

- \( \text{out}_{AE}(j) = (\text{in}_{AE}(j) - \text{kill}_{AE}(j)) \cup \text{gen}_{AE}(j) \)
- \( \text{kill}_{AE}(j) \): all expressions with operand \( x \): \( (x \ op \ _), (_ \ op \ x) \)
- \( \text{gen}_{AE}(j) \): new expression: \( \{ (y \ op \ z) \} \)
Available Expressions (Avail)

- Merge operator?
  - For Avail, it is set intersection $\bigcap$

$$in_{AE}(j) = \{ \bigcap out_{AE}(i) \mid i \text{ is predecessor of } j \}$$
Example

1. \( y = a + b \)
2. \( x = a \times b \)
3. \( \text{if } y \leq a \times b \)
4. \( a = a + 1 \)
5. \( x = a \times b \)
6. \( \text{goto 3} \)
7. ...
What are the primitive dataflow facts?
Expressions, e.g., $x+y$, $a*b$. Equations act on sets of expressions.
Problem 4: Very Busy Expressions (VeryB)

An expression $x \text{ op } y$ is very busy at node $n$, if along EVERY path from $n$ to the end of the program, we come to a computation of $x \text{ op } y$ BEFORE any redefinition of $x$ or $y$. 

$$
\begin{align*}
\text{X} &= \ldots \\
\text{Y} &= \ldots \\
\text{t1} &= \text{X op Y} \\
\text{X} &= \ldots \\
\text{Y} &= \ldots \\
\text{t1} &= \text{X op Y} \\
\text{X} &= \ldots \\
\text{Y} &= \ldots \\
\text{t1} &= \text{X op Y}
\end{align*}
$$
Very Busy Expressions (*VeryB*)

- **Data?**
  - Primitive dataflow facts are expressions, e.g., \(x + y, a \times b\)
  - Analysis propagates sets of expressions, e.g., \(\{x + y, a \times b\}\)

- **Dataflow equations at** \(j\): \(x = y \operatorname{op} z\)?
  - \(\text{in}_{\text{VB}}(j) = (\text{out}_{\text{VB}}(j) - \text{kill}_{\text{VB}}(j)) \cup \text{gen}_{\text{VB}}(j)\)
  - \(\text{kill}_{\text{VB}}(j)\): all expressions with operand \(x\):
    - \((x \operatorname{op} \_), (\_ \operatorname{op} x)\)
  - \(\text{gen}_{\text{VB}}(j)\): new expression: \(\{(y \operatorname{op} z)\}\)
Very Busy Expressions (VeryB)

- Merge operator?
  - For VeryB, it is set intersection \( \bigcap \)

\[
\text{out}_{\text{VB}}(j) = \{ \bigcap \text{in}_{\text{VB}}(i) \mid i \text{ is successor of } j \}
\]
Very Busy Expressions

Backward, must dataflow problem

\[ j \]
\[ \text{out}_{VB}(j) \]

\[ i1 \]
\[ \text{out}_{VB}(i1) \]

\[ i2 \]
\[ \text{out}_{VB}(i2) \]

\[ i3 \]
\[ \text{out}_{VB}(i3) \]
Another Example: Taint Analysis

- A definition \( i: x = \ldots (x,i) \) is \textit{tainted} if
  - \( i: x = \text{tainted\_source}() \) is designated as a taint source
    - e.g., `deviceId=telephony_mgr.getDeviceId();`
  - or \( i: x = y \text{ op } z \) and a tainted \((y,j)\) or a tainted \((z,k)\) reaches \( i \)

- Problem statement: for each node \( n \), compute the set of tainted definitions that reach \( n \).
Example: Taint Analysis (explicit flow)

1. x = read()

2. y = 1

3. x >= 2

4. y = x * y

5. x = x - 1

6. goto 3

7. z = y - 1
Outline of Today’s Class

- Catch up
- Dataflow frameworks
  - Lattices
  - Transfer functions (next time)
  - Worklist algorithm (next time)
- Reading:
  - Dragon Book, Chapter 9.2 and 9.3
<table>
<thead>
<tr>
<th></th>
<th>May Problems</th>
<th>Must Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Forward Problems</strong></td>
<td>Reaching Definitions</td>
<td>Available Expressions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Backward Problems</strong></td>
<td>Live Uses of Variables</td>
<td>Very Busy Expressions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Similarities

- Analyses operate over similar property spaces
- In all cases, analysis operates over a finite set $D$ of primitive dataflow facts
  - Reach: $D$ is the set of all definitions in the program: e.g., $\{(x,1), (y,2), (x,4), (y,5)\}$
  - Avail and VeryB: $D$ is the set of all arithmetic expressions: e.g., $\{a+b, a*b, a+1\}$
  - Live: $D$ is the set of all variables e.g., $\{x, y, z\}$
- Solution at node $n$ is a subset of $D$ (e.g., a definition either reaches $n$ or it does not reach $n$)
Similarities

- Dataflow equations have the same form (from now on, we’ll focus on forward problems):

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

  \[
  \text{in}(j) = \{ V \text{ out}(i) | i \text{ is predecessor of } j \}
  \]

  \text{pres}(j) \text{ is the complement of } \text{kill}(j)

- A note: what makes the 4 classical problems special is that sets \text{kill}(j)/\text{pres}(j) and \text{gen}(j) do not depend on \text{in}(j)

- Set union and set intersection can be implemented as logical OR and AND respectively
The dataflow equation at node $j$ is defined in terms of a transfer function. It takes $\text{in}(j)$ as argument and produces $\text{out}(j)$ as result:

$$\text{out}(j) = f_j(\text{in}(j))$$
Dataflow Frameworks

- We generalize and study the properties of the property space
  - Property space is a lattice
  - Choice settles merge operator
- We generalize and study the properties of the transfer function space
  - Functions are monotone or distributive
- We generalize and study the properties of the worklist algorithm that computes a solution
Lattice Theory

- **Partial ordering** (denoted by $\leq$ or $\subseteq$)
  - Relation between pairs of elements
  - Reflexive $a \leq a$
  - Anti-symmetric $a \leq b$ and $b \leq a \implies a = b$
  - Transitive $a \leq b$ and $b \leq c \implies a \leq c$

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

We don’t necessarily need 0 or 1 element
D = \{a,b,c\}
The poset is $2^D$, $\leq$ is set inclusion

Spring 21 CSCI 4450/6450, A Milanova
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 \textit{meet} (denoted by \( \wedge \) or \( \cap \)) 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 \textit{join} (denoted by \( \vee \) or \( \sqcup \)) of \( l_1 \) and \( l_2 \).
Definition of a Lattice \((\mathbb{L}, \land, \lor)\)

- A lattice \(\mathbb{L}\) 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 \(\mathbb{L}\), then \(a\) is the 0 element of \(\mathbb{L}\).
- If there is \(a\) such that \(x \leq a\) for every \(x\) in \(\mathbb{L}\), then \(a\) is the 1 element of \(\mathbb{L}\).
A Poset but Not a Lattice

There is no \( \text{lub}(e_3, e_4) \) in this poset so it is not a lattice.

Suppose we add the \( \text{lub}(e_3, e_4) \), is it a lattice?
Is This Poset a Lattice

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

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

- $J = (N_1, \gcd, \lcm)$
  - Partial order is integer divide on $N_1$
  - $\text{lub}(n_1, n_2)$ denoted $n_1 \vee n_2$ is $\lcm(n_1, n_2)$
  - $\text{glb}(n_1, n_2)$ denoted $n_1 \Lambda n_2$ is $\gcd(n_1, n_2)$
  ($N_1$ denotes natural numbers starting at 1)
Chain

- 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 in Dataflow Analysis

- Lattices define property space
- Lattices entail properties of the standard dataflow analysis solution procedure (the worklist algorithm, which we will study shortly)
Dataflow Lattices: \textit{Reach}

D = 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. $x=a\cdot b$

2. \textbf{if} $y \leq a\cdot b$

3. $a=a+1$

4. $x=a\cdot b$

5. goto 3

Spring 21 CSCI 4450/6450, A Milanova
Dataflow Lattices: *Avail*

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

Poset is $2^D$, $\leq$ 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
Dataflow Frameworks

Equations:

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

\[ \text{i in pred}(j) \]

where:

- \( \text{in}(j), \text{out}(j) \) are elements of a property space
- \( f_j \) is the transfer function associated with node \( j \)
- \( V \) is the merge operator
The property space must be:

1. A lattice $L, \leq$
2. $L$ satisfies the Ascending Chain Condition
   Requires that all ascending chains are finite
Dataflow Frameworks (cont.)

- The 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 the merge operator must be the join of $L$, a \textit{may} problem entails that $\leq$ is \textit{subset} and a \textit{must} problem entails that $\leq$ is \textit{superset}
Example: *Reach* Lattice

- 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 $\cup$, as needed for *Reach*, which is a *may* problem
- Lattice has $0$ being $\emptyset$, and $1$ being $D$
- Lattice satisfies the *Ascending Chain Condition*
**Reach Lattice**

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

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

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

Spring 21 CSCI 4450/6450, A Milanova
Example: *Avail* Lattice

- Property space is the lattice of the subsets where
  - $D$ is the set of all expressions in the program
  - $\leq$ is superset
    - *join* of the lattice is set intersection, as needed for *Avail*, which is a *must* problem

- Lattice has 0 being $D$, and 1 being $\emptyset$
- Lattice satisfies *Ascending Chain Condition*
Dataflow Lattices: *Avail*

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

Poset is \(2^D\), \(\leq\) 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