Dataflow Analysis: Dataflow Frameworks
Outline of Today’s Class

- Catch up, four classical dataflow problems
- Dataflow frameworks
  - Lattices
  - Transfer functions
  - Worklist algorithm

Reading:
- Dragon Book, Chapter 9.2 and 9.3
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)

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

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

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 $U$

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

E.g., if \(\text{out}(2) = \{(x,1), (y,2)\}\) and \(\text{out}(3) = \{(x,3)\}\) and 2 and 3 are predecessors of 4
\[
in(4) = \{(x,1), (x,3), (y,2)\}\]
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*/*y*; 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. \( (y > x) \)
   - T
     - \( z = y \)
     - \( x = z \)
   - F
     - \( z = y \times y \)
     - \( x = z \)

\( i_{in}(1) = 2 \)
\( i_{in}(2) = 4 \)
\( i_{in}(3) = 2 \)
\( i_{in}(4) = 2 \times 3 \)
\( i_{in}(5) = 2 \times 3 \)
\( i_{in}(6) = 2 \times 3 \)
\( i_{in}(7) = 2 \times 3 \)

\( out(1) = 2 \)
\( out(2) = 4 \)
\( out(3) = 2 \times 3 \)
\( out(4) = 2 \times 3 \)
\( out(5) = 2 \times 3 \)
\( out(6) = 2 \times 3 \)
\( out(7) = 2 \times 3 \)

\( i_{in}(4) = \frac{9}{e^2} \)
Live Uses of Variables (Live)

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

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

\[ \text{out}_{LV}(j) = \{ u \in \text{in}_{LV}(i) \mid i \text{ is a successor of } j \} \]

Q: What are the primitive dataflow facts?
Q: What is $\text{gen}_{LV}(j)$?
Q: What is $\text{kill}_{LV}(j)$?
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 U

\[
\text{kill}_{LV}(j) = \{x\} \\
\text{gen}_{LV}(j) = \{y, z\} \\
\text{kill}_{LV}(j) = \{x\} \\
\text{gen}_{LV}(j) = \{y, z\}
\]
Live Uses of Variables

Backward, may dataflow problem
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 \( n \).
Problem 3. Available Expressions (Avail)

- Problem statement: For every node $n$, compute the set of expressions that are available at $n$
Avail Enables Global Common Subexpression Elimination

\[ z = a \times b \]
\[ r = 2 \times z \]
\[ q = a \times b \]
\[ u = a \times b \]
\[ z = u / 2 \]
\[ w = a \times b \]
Avail Enables Global Common Subexpression Elimination

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 $\bigcap$

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

```
j : x = x op y
kill_{AE}(j) = (x \ op \ -), (-\ op \ x)
gen_{AE}(j) = \emptyset
```
Available Expressions (Avail)

in(i1) \rightarrow i1
in(i2) \rightarrow i2
in(i3) \rightarrow i3

\text{Forward, must dataflow problem}

j \leftarrow \text{in(j)}
\text{x=y+z}
Example

\[ y = a + b \]

\[ x = a \times b \]

\[ \text{if } y \leq a \times b \]

\[ a = a + 1 \]

\[ x = a \times b \]

\[ \text{goto 3} \]

\[ \text{...} \]
Note on Homework

1. \( x = x + b \)
2. \( y = x + 1 \)
3. \( x = x + y \)

\( B_1 \)

\( \text{LUV:} \)
\( \text{Kill} (B_1) = \sum x_i y_i \)
\( \text{Gen} (B_1) = \exists (x, b^3) \)

\( RD \)
\( \text{Kill} (B_1) = \text{All definitions of } y \text{ and } x \)
\( \text{Gen} (B_1) = \exists (y, 11) , (x, 12) \)

AE:
Very Busy Expressions

- 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 $y$.

```
X = ...
Y = ...
t1 = X \text{ op } Y
X = ...
Y = ...
t1 = X \text{ op } Y
X = ...
Y = ...
t1 = X \text{ op } Y
```
Problem 4. Very Busy Expressions (*VeryB*)

- Problem Statement: For each node \( n \), compute the set of expressions that are very busy on exit from \( n \)

\[
j: x = y + z
\]

- Q: What is the data?
- Q: What are the equations?
- Q: What is \( \text{gen}_{\text{VB}}(i) \)?
- Q: What is \( \text{kill}_{\text{VB}}(i) \)?
- Q: What is the merge operator?
Very Busy Expressions (VeryB)

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

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

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

$$out_{VB}(j) = \{ \bigcap in_{VB}(i) \mid i \text{ is successor of } j \}$$
Very Busy Expressions

Backward, must dataflow problem

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

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

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

\[ \text{out}_{VB}(i3) \]
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., \( \text{deviceId} = \text{telephony\_mgr\_getDeviceId}() \);
  - or \( i: x = y \text{ op } z \) and a tainted \( (y,j) \) or a tainted \( (z,k) \) reaches program point \( i \)

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

1. \( x = \text{read()} \)
2. \( y = 1 \)
3. \( x \geq 2 \)
4. \( y = x \times y \)
5. \( x = x - 1 \)
6. \( \text{goto 3} \)
7. \( z = y - 1 \)
Outline of Today’s Class

- Catch up
- Dataflow frameworks
  - Lattices
  - Transfer functions
  - Worklist algorithm

Reading:
- Dragon Book, Chapter 9.2 and 9.3
## Dataflow Problems

<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><strong>Backward Problems</strong></td>
<td>Live Uses of Variables</td>
<td>Very Busy Expressions</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 $n$)
Dataflow equations have same form (from now on, we’ll focus on forward problems):

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

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

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

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

- 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
Dataflow equation at node $j$ is 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 properties of the property space
  - Property space is a lattice
  - Choice of lattice settles merge operator
- We generalize and study properties of the transfer function space
  - Functions are monotone or distributive
- We generalize and study properties of the worklist algorithm that computes a solution
Lattices

- 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
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 \( \text{glb} \) 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 \( \text{lub} \) 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 \(L\).
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

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

$\text{lub}(l_1, l_2) = l_1 \cup l_2$
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 $s_1 \lor s_2$, is set union $s_1 \cup s_2$

- $J = (N_1, \text{gcd}, \text{lcm})$
  - Partial order is integer divide on $N_1$
  - $\text{lub}(n_1,n_2)$ denoted $n_1 \lor n_2$ is $\text{lcm}(n_1,n_2)$
  - $\text{glb}(n_1,n_2)$ denoted $n_1 \Lambda n_2$ is $\text{gcd}(n_1,n_2)$

($N_1$ 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., $\emptyset \leq \{a\} \leq \{a,b\} \leq \{a,b,c\}$
- 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

- Lattice properties lead to certain properties of the worklist algorithm (standard way of solving dataflow problems)
Dataflow Lattices: 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. \(x = a \times b\)
- 2. if \(y \leq a \times b\)
- 3. \(a = a + 1\)
- 4. \(x = a \times b\)
- 5. goto 3

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

CSCI 4450/6450, A Milanova
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 the 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
  - $\mathbb{D}$ 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 $\emptyset$, and a 1 being $\mathbb{D}$
  - Lattice satisfies the *Ascending Chain Condition*
Example: *Avail* Lattice

- Property space is the lattice of the subsets
  - $\mathbb{D}$ 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 $\mathbb{D}$, and a 1 being $\emptyset$
- Lattice satisfies *Ascending Chain Condition*
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 (also the maximal fixpoint algorithm or the fixpoint iteration algorithm)
Outline of Today’s Class

- Catch up
- Dataflow frameworks
  - Lattices
  - Transfer functions
  - Worklist algorithm

- 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
**Monotonicity Property**

- \( F: L \rightarrow L \) is monotone if and only if:
  1. \( a, b \) in \( L \), \( f \) in \( F \) then \( a \leq b \iff f(a) \leq f(b) \)
  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)
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 classical Reach distributive?
- Yes

To show distributivity:
For each $j$: $( ( X_1 \cup X_2 ) \cap \text{pres}(j) ) \cup \text{gen}(j) =$
$( (X_1 \cap \text{pres}(j)) \cup \text{gen}(j) ) \cup ( (X_2 \cap \text{pres}(j)) \cup \text{gen}(j) )$

$( ( X_1 \cup X_2 ) \cap \text{pres}(j) ) \cup \text{gen}(j) =$
$( ( X_1 \cap \text{pres}(j) ) \cup ( X_2 \cap \text{pres}(j) ) ) \cup \text{gen}(j) =$
$( (X_1 \cap \text{pres}(j)) \cup \text{gen}(j) ) \cup ( (X_2 \cap \text{pres}(j)) \cup \text{gen}(j) )$
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 \( F: L \rightarrow L \) is monotone

Thus, we can make use of a generic solution procedure, known as the **worklist algorithm**.
/* 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 ≠ ∅ 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}

\( D = \) 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
Termination Argument

Why does the algorithm terminate?

Sketch of argument:

A node $j$ is placed on the worklist only if the $\text{out}(i)$ of a predecessor $i$ changes. Monotonicity of $f$ ensures that $\text{in}^k(i) \leq \text{in}^{k+1}(i)$ and $\text{out}^k(i) \leq \text{out}^{k+1}(i)$. $\text{in}(i)$ and $\text{out}(j)$ sets and in $L$ and $L$ satisfies the Ascending Chain Condition; therefore, there is only a finite number of times each $\text{out}(i)$ 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” $out(i)$ and not added successor $j$ to worklist or otherwise $in(j)$ would have been recomputed to account for new $out(i)$; This is impossible.
Theorem: Worklist algorithm computes the least solution of the dataflow equations.

Historically though, this solution is called the maximal fixpoint solution (MFP).

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

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

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