I. Meet Operator

- Properties of the meet operator
  - Commutative: \( x \land y = y \land x \)
  - Idempotent: \( x \land x = x \)
  - Associative: \( x \land (y \land z) = (x \land y) \land z \)
  - There is a Top element \( T \) such that \( x \land T = x \)

- Meet operator defines a partial ordering on values
  - \( x \leq y \) if and only if \( x \land y = x \)
  - Transitivity: if \( x \leq y \) and \( y \leq z \) then \( x \leq z \)
  - Antisymmetry: if \( x \leq y \) and \( y \leq x \) then \( x = y \)
  - Reflexivity: \( x \leq x \)

A Unified Framework

- Data flow problems are defined by
  - Domain of values: \( V \)
  - Meet operator \( (V, \land, \top) \), initial value
  - A set of transfer functions \( (V \to V) \)

- Usefulness of unified framework
  - To answer questions such as correctness, precision, convergence, speed of convergence for a family of problems
    - If meet operators and transfer functions have properties \( X \), then we know \( Y \) about the above.
  - Reuse code

Partial Order

- Example: let \( V = \{ x \mid x \subseteq \{ d_1, d_2, d_3 \} \}, \land = \cap \)

- Top and Bottom elements
  - Top \( T \) such that: \( x \land T = x \)
  - Bottom \( \perp \) such that: \( x \land \perp = \perp \)

- Values and meet operator in a data flow problem define a semi-lattice:
  - There exists a \( T \), but not necessarily a \( \perp \)
  - \( x, y \) are ordered: \( x \leq y \) then \( x \land y = x \)
  - What if \( x \) and \( y \) are not ordered?
    - \( x \leq y \), \( x \land y \), and if \( w \leq x, w \leq y \), then \( w \leq x \land y \)
**One vs. All Variables/Definitions**

- Lattice for each variable: e.g. intersection
- **Lattice for three variables:**
  \[
  \begin{array}{cccc}
  x_{111} & x_{101} & x_{011} & x_{001} \\
  x_{110} & x_{100} & x_{010} & x_{000} \\
  \end{array}
  \]

**Descending Chain**

- Definition
  - The **height** of a lattice is the largest number of \( \times \) relations that will fit in a descending chain.
  \[ x_0 \times x_1 \times x_2 \times \ldots \]
- Height of values in reaching definitions?
- **Important property: finite descending chain**
  - Can an infinite lattice have a finite descending chain?
- **Example: Constant Propagation/Folding**
  - To determine if a variable is a constant
- **Data values**
  - undef, -1, 0, 1, 2, ..., not-a-constant

**II. Transfer Functions**

- **Basic Properties** \( f : V \rightarrow V \)
  - Has an identity function
    - There exists an \( f \) such that \( f(x) = x \), for all \( x \).
  - Closed under composition
    - if \( f_1, f_2 \in F \), then \( f_1 \cdot f_2 \in F \)

**Monotonicity**

- A framework \((F, V, \sqcap)\) is **monotone** if and only if
  - \( x \sqsubseteq y \) implies \( f(x) \sqsubseteq f(y) \)
    - i.e. a “smaller or equal” input to the same function will always give a “smaller or equal” output
- **Equivalently**, a framework \((F, V, \sqcap)\) is **monotone** if and only if
  - \( f(x \sqcap y) \leq f(x) \sqcap f(y) \)
    - i.e. merge input, then apply \( f \) is small than or equal to apply the transfer function individually and then merge the result
**Example**

- **Reaching definitions:** $f(x) = \text{Gen} \cup (x - \text{Kill})$, $x = \cup$
  - Definition 1:
    - $x_1 \leq x_2$, $\text{Gen} \cup (x_1 - \text{Kill}) \leq \text{Gen} \cup (x_2 - \text{Kill})$
  - Definition 2:
    - $(\text{Gen} \cup (x_1 - \text{Kill})) \cup (\text{Gen} \cup (x_2 - \text{Kill})) = (\text{Gen} \cup ((x_1 \cup x_2) - \text{Kill}))$
- **Note:** Monotone framework does not mean that $f(x) \leq x$
  - e.g., reaching definition for two definitions in program
  - suppose: $f_1, \text{Gen}, (d_1, d_2); \text{Kill}, = \emptyset$

- If input(second iteration) $\leq$ input(first iteration)
  - result(second iteration) $\leq$ result(first iteration)

**Distributivity**

- A framework $(F, V, \land)$ is **distributive** if and only if
  - $f(x \land y) = f(x) \land f(y)$
  - i.e. merge input, then apply $f$ is equal to apply the transfer function individually then merge result
- **Example:** Constant Propagation

```
a = 2
b = 3
c = a + b
```

**Meet-Over-Paths (MOP)**

- **Err in the conservative direction**
  - **Meet-Over-Paths (MOP):**
    - For each node $n$:
      - $\land f_p(T)$, for all possibly executed paths $p$ reaching $n$
      - $a$ path exists as long there is an edge in the code
      - consider more paths than necessary
      - $\text{MOP} = \text{Perfect-Solution} \land \text{Solution-to-Unexecuted-Paths}$
      - $\text{MOP} \leq \text{Perfect-Solution}$
      - Potentially more constrained, solution is small
        - hence conservative
      - It is not safe to be $> \text{Perfect-Solution}$!
      - **Desirable solution:** as close to MOP as possible

```
a = 3
b = 2
c = a + b
```

**III. Data Flow Analysis**

- **Definition**
  - Let $f_1, ..., f_n \in F$, where $f_i$ is the transfer function for node $i$
    - $f_p = f_{p_2} \land ... \land f_{p_k}$ where $p$ is a path through nodes $n_1, ..., n_k$
    - $f_p$ is identify function, if $p$ is an empty path

- **Ideal data flow answer:**
  - For each node $n$:
    - $\land f_p(T)$, for all possibly executed paths $p$ reaching $n$

```
if sqrt(y) >= 0
x = 0
x = 1
```

- Determining all possibly executed paths is **undecidable**
Solving Data Flow Equations

- Example: Reaching definitions
  - out[entry] = {}
  - Values = (subsets of definitions)
- Meet operator: \( \cup \)
  - \( \text{in}[b] = \cup \text{out}[p] \), for all predecessors \( p \) of \( b \)
- Transfer functions: \( \text{out}[b] = \text{gen}_b \cup (\text{in}[b] \setminus \text{kill}_b) \)
- Any solution satisfying equations = Fixed Point Solution (FP)

Iterative algorithm
- initializes \( \text{out}[b] \) to {}
- if converges, then it computes Maximum Fixed Point (MFP)
  - MFP is the largest of all solutions to equations

Properties:
- \( \text{FP} \leq \text{MFP} \leq \text{MOP} \leq \text{Perfect-solution} \)
- \( \text{FP}, \text{MFP} \) are safe

Partial Correctness of Algorithm

- If data flow framework is monotone, then if the algorithm converges, \( \text{IN}[b] \leq \text{MOP}[b] \)
- Proof: Induction on path lengths
  - Define \( \text{IN}[\text{entry}] = \text{OUT}[\text{entry}] \)
    and transfer function of \( \text{entry} \) = Identity function
  - Base case: path of length 0
    - Proper initialization of \( \text{IN}[\text{entry}] \)
  - If true for path of length \( k \), \( p_k = (n_1, \ldots, n_k) \), then true for path of length \( k+1 \), \( p_{k+1} = (n_1, \ldots, n_k, n_{k+1}) \)
    - Assume: \( \text{IN}[n_1] \leq \text{OUT}[n_1] \) ...
    - \( \text{IN}[n_k] \leq \text{OUT}[n_k] \)
    - \( \text{IN}[n_k] \leq \text{OUT}[n_k] \) ...

Precision

- If data flow framework is distributive, then if the algorithm converges, \( \text{IN}[b] = \text{MOP}[b] \)
- Monotone but not distributive: behaves as if there are additional paths

\[
\begin{align*}
a &= 2 \\
b &= 3 \\
c &= a + b
\end{align*}
\]

\[
\begin{align*}
a &= 3 \\
b &= 2 \\
c &= a + b
\end{align*}
\]

Additional Property to Guarantee Convergence

- Data flow framework (monotone) converges if there is a finite descending chain
  - For each variable \( \text{IN}[b], \text{OUT}[b] \), consider the sequence of values set to each variable across iterations:
    - if sequence for \( \text{in}[b] \) is monotonically decreasing
      - sequence for \( \text{out}[b] \) is monotonically decreasing
        - (\( \text{out}[b] \) initialized to T)
    - if sequence for \( \text{out}[b] \) is monotonically decreasing
      - sequence of \( \text{in}[b] \) is monotonically decreasing
IV. Speed of Convergence

- Speed of convergence depends on order of node visits
- Reverse "direction" for backward flow problems

Reverse Postorder

- Step 1: depth-first post order
  ```
  main() {
    count = 1;
    Visit(root);
  }

  Visit(n) {
    for each successor s that has not been visited
      Visit(s);
    PostOrder(n) = count;
    count = count+1;
  }
  ```

- Step 2: reverse order
  ```
  For each node i
  rPostOrder = NumNodes - PostOrder(i)
  ```

Depth-First Iterative Algorithm (forward)

```
input: control flow graph CFG = (N, E, Entry, Exit)
/* Initialize */
/* Initialize */
out[entry] = init_value
For all nodes i
  out[i] = T
  Change = True
/* iterate */
While Change {
  Change = False
  For each node i in rPostOrder {
    in[i] = \lambda(out[p]), for all predecessors p of i
    oldout = out[i]
    out[i] = f_i(in[i])
    if oldout \neq out[i]
      Change = True
  }
}
```

Speed of Convergence

- If cycles do not add information
  - information can flow in one pass down a series of nodes of increasing order number:
    - e.g., 1 \rightarrow 4 \rightarrow 5 \rightarrow 7 \rightarrow 2 \rightarrow 4 ...
    - passes determined by number of back edges in the path
      - essentially the nesting depth of the graph
      - Number of iterations = number of back edges in any acyclic path + 2
        - (2 are necessary even if there are no cycles)
  - What is the depth?
    - corresponds to depth of intervals for "reducible" graphs
    - in real programs: average of 2.75
A Check List for Data Flow Problems

• **Semi-lattice**
  – set of values
  – meet operator
  – top, bottom
  – finite descending chain?

• **Transfer functions**
  – function of each basic block
  – monotone
  – distributive?

• **Algorithm**
  – initialization step (entry/exit, other nodes)
  – visit order: rPostOrder
  – depth of the graph