**Data-flow Analysis Frameworks**

- Can use same framework to solve these data-flow problems
  - Local GEN, KILL information for each basic block
  - Initial values for data-flow solutions
  - Iterate through nodes in CFG until values stabilize
- Data-flow framework has three components
  - Set of values $L$
  - Operator for combining values $\land$
  - A set of propagation functions $L \rightarrow L$
- Benefits of using framework
  - Define properties needed to guarantee correctness, convergence
  - Can describe convergence speed and precision of results
  - Can reuse code to solve other problems

**Data-flow Analysis**

- Many data-flow equations have the same structure
  - $RD(B) = \{ GEN(B) \cup (RD(B) \setminus KILL(B)) \}$
  - $LV(B) = \{ GEN(B) \cup (LV(B) \setminus KILL(B)) \}$
  - $AVAIL(B) = \cap (GEN(B) \cup (AVAIL(B) \setminus KILL(B)))$
  - $VE(B) = \cap (GEN(B) \cup (VE(B) \setminus KILL(B)))$
  - $CONST(B) = \cap (GEN(B) \cup (CONST(B) \setminus KILL(B)))$
  
  Where $B \in PRE(B)$ or $SUCC(B)$ depending on problem

- What do data-flow problems have in common?
  - Meet operator $\land$ to merge results
  - Propagation functions to model basic blocks
  - Direction:
    - Forward, backward
    - Best case and worst case values

**Data-flow Lattices**

- A lattice consists of a set of values $L$ and a meet operator $\land$
  - For every $a, b, c \in L$
    - $a \land a = a$ (idempotent)
    - $a \land b = b \land a$ (commutative)
    - $(a \land b) \land c = a \land (b \land c)$ (associative)
  - $\land$ imposes a partial order on $L$
  - $a \land b = a \land b$
  - $a \land b = a \land b$ and $a \land b$
  - A lattice may have a top element
    - $\top \land a = a$
  - A lattice may have a bottom element
    - $\bot \land 0 = \bot$

**Iterative Solver**

- What about loops?
  - Circular dependences between basic blocks
  - Can initialize and solve repeatedly
- Termination:
  - Goal is for solutions to converge to a fixed point
  - Can stop once answer stops changing
  - Is this guaranteed?
Monotonicity

- A data-flow analysis framework is monotone if
  \[ x \leq y \] implies \( f(x) \leq f(y) \)
  i.e., "a smaller or equal" input to the same function will always give
  a "smaller or equal" output
- Equivalently
  \[ f(x \land y) \leq f(x) \land f(y) \]
  i.e., if result of merging inputs then applying \( f \) is "smaller or equal" to applying \( f \) individually then merging result
- Intuitively, monotonicity means "smaller" input will not yield "larger" output
- Monotone frameworks are guaranteed to converge and terminate
  \[ \text{If lattice elements can drop information a finite number of times} \]

Quality of Solution

- Possible solutions
  \[ \text{Perfect solution} \]
  \[ \text{Meet over real paths taken during program execution} \]
  \[ \text{Meet-over-all-paths (MOP)} \]
  \[ \text{Meet over potential paths in control flow graph} \]
  \[ \text{Maximal-fixed-point (MFP)} \]
  \[ \text{Solution from iterative framework} \]
- Properties
  \[ \text{In general, MFP \& MOP \& perfect solution} \]
  \[ \text{In some sense, MOP is the best feasible solution} \]
  \[ \text{MFP is unique, regardless of order of propagation} \]
  \[ \text{A framework is distributive if } f(x \land y) = f(x) \land f(y) \]
  \[ \text{MFP = MOP for distributive framework} \]