Dataflow Analysis Frameworks

Data-flow Analysis

- Many data-flow equations have the same structure
  - → RD(B) = $\cup$ (GEN(Bi) $\cup$ [RD(Bi) - KILL(Bi)])
  - → LV(B) = $\cup$ (GEN(Bi) $\cup$ [LV(Bi) - KILL(Bi)])
  - → AVAIL(B) = $\cap$ (GEN(Bi) $\cup$ [AVAIL(Bi) - KILL(Bi)])
  - → VBE(B) = $\cap$ (GEN(Bi) $\cup$ [VBE(Bi) - KILL(Bi)])
  - → CONST(B) = $\cap$ (GEN(Bi) $\cup$ [CONST(Bi) - KILL(Bi)])

    Where Bi ε PRED(B) or SUCC(B) depending on problem

- What do data-flow problems have in common?

→ Meet operator $\wedge$ to merge results	$\cup$, $\cap$
→ Propagation functions to model basic blocks	GEN, KILL
→ Direction	forward, backward
→ Best case and worst case values	$\top$ $\bot$

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 — $\wedge$
  - → A set of propagation functions — $L \rightarrow L$

- Benefits of using framework
  - → Defines properties needed to guarantee correctness, convergence
  - → Can describe convergence speed and precision of results
  - → Can reuse code to solve other problems

Data-flow Lattices

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

Data-flow Lattices

- How does this relate to data-flow analysis?
  - → Choose a semi-lattice L to represent facts
  - → Attach to each element of L a meaning
    - Each a e L is a distinct set of known facts
  - → For each basic block n, associate a propagation / transfer function
    - $f_n : L \rightarrow L$ models behavior of n
  - → Propagate facts around control flow graph

- Example for AVAIL
  - → Semi-lattice L is 2^E, where
    - E is set of all expressions
    - $\wedge$ is $\cap$
    - $\top$ is $\emptyset$
    - $\perp$ is E
  - → For a node n, fn has the form
    - $f_n(x) = GEN_n \cup (x - KILL_n)$

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
  - $x \le y$ implies $f(x) \le f(y)$
    - i.e., "a smaller or equal" input to the same function will always give a "a smaller or equal" output
- Equivalently
  - $f(x \wedge y) \le f(x) \wedge f(y)$
    - 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
  - If lattice elements can drop information a finite number of times

Quality of Solution

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