Dataflow Analysis Frameworks

Data-flow Analysis

• Many data-flow equations have the same structure
  → RD(B) = ∪ ( GEN(Bi) ∪ [ RD(Bi) - KILL(Bi) ] )
  → LV(B) = ∪ ( GEN(Bi) ∪ [ LV(Bi) - KILL(Bi) ] )
  → AVAIL(B) = ∩ ( GEN(Bi) ∪ [ AVAIL(Bi) - KILL(Bi) ] )
  → VBE(B) = ∩ ( GEN(Bi) ∪ [ VBE(Bi) - KILL(Bi) ] )
  → CONST(B) = ∩ ( GEN(Bi) ∪ [ CONST(Bi) - KILL(Bi) ] )
  Where Bi ∈ PRED(B) or SUCC(B) depending on problem

• What do data-flow problems have in common?
  → Meet operator ∧ to merge results
  → Propagation functions to model basic blocks
  → Direction forward, backward
  → Best case and worst case values

CS430
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
  - 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 \( \land \)
  - 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 \( L \)
    - \( a \geq b \iff a \land b = b \)
    - \( a > b \iff a \geq b \text{ and } a \neq b \)
  - A lattice may have a top element
    - \( \top \land a = a \)
  - A lattice may have a bottom element
    - \( \bot \land a = \bot \)
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 $a \in L$ 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 $\emptyset$
    • $\bot$ is $E$
  → For a node $n$, $f_n$ has the form
    • $f_n(x) = \text{GEN}_n \cup (x - \text{KILL}_n)$

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
  \[ x \leq y \implies f(x) \leq f(y) \]
  i.e., "a smaller or equal" input to the same function will always give a "a smaller or equal" output

- Equivalently
  \[ f(x \land y) \leq f(x) \land f(y) \]
  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
  - **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, \( \text{MFP} \leq \text{MOP} \leq \text{perfect solution} \)
  - In some sense, \( \text{MOP} \) is the best feasible solution
  - \( \text{MFP} \) is unique, regardless of order of propagation
  - A framework is distributive if \( f(x \land y) = f(x) \land f(y) \)
    - \( \text{MFP} = \text{MOP} \) for distributive framework