Speeding Up Data-Flow Analysis

Last time
− Various optimizations that use data-flow analysis

Today
− Speeding up data-flow analysis

Solving the Data-flow Equations

Algorithm

\[
\text{for each node } n \text{ in CFG} \\
\quad \text{in}^*[n] = \text{in}[n] \\
\quad \text{out}^*[n] = \text{out}[n] \\
\quad \text{in}[n] = \text{in}[n] \cup (\text{out}[n] - \text{def}[n]) \\
\quad \text{out}[n] = \bigcup_{s \in \text{succ}[n]} \text{in}[s] \\
\text{until } \text{in}^*[n] = \text{in}[n] \text{ and out}^*[n] = \text{out}[n] \text{ for all } n
\]

\{ initialize solutions \}
\{ save current results \}
\{ solve data-flow equations \}
\{ test for convergence \}

This is iterative data-flow analysis (for liveness analysis)

Time Complexity (liveness)

Consider a program of size \( N \)
− Has \( N \) nodes in the flow graph and at most \( N \) variables
− Each live-in or live-out set has at most \( N \) elements
− Each set-union operation takes \( O(N) \) time
− The for loop body
  − constant # of set operations per node
  − \( O(N) \) nodes \( \Rightarrow \) \( O(N^2) \) time for the loop
− Each iteration of the repeat loop can only make the set larger
− Each set can contain at most \( N \) variables \( \Rightarrow \) \( 2N^2 \) iterations

Worst case: \( O(N^4) \)
Typical case: 2 to 3 iterations with good ordering & separability
\( \Rightarrow \) \( O(N) \) to \( O(N^2) \)

Efficiency (from lattice lecture)

Parameters
− \( n \): Number of nodes in the CFG
− \( k \): Height of lattice
− \( t \): Time to execute one flow function

Complexity
− \( O(nkt) \)

Relation to previous slide
− \( n = N \)
− \( k = N^2 \) // used as worst-case for repeat loop
− \( t = N \)
Ways to improve data-flow analysis efficiency

Node ordering
- Visit nodes in an ordering that most efficiently propagates change

Bitvectors
- Use the tuple of lattices concept to implement the data-flow sets as bit-vectors

Worklist
- Only visit nodes where the input data-flow sets have changed

Basic Blocks
- Group statements with straightforward control-flow

Others
- Structural or interval analysis
- Slotwise analysis
- SSA

Example (liveness)

Data-flow Equations for Liveness
\[
\text{in}[n] = \text{use}[n] \cup (\text{out}[n] - \text{def}[n])
\]
\[
\text{out}[n] = \bigcup_{s \in \text{succ}[n]} \text{in}[s]
\]

Improving Performance
Consider the \((3 \rightarrow 4)\) edge in the graph:
\[
\text{out}[4] \text{ is used to compute } \text{in}[4]
\]
\[
\text{in}[4] \text{ is used to compute } \text{out}[3] \ldots
\]
So we should compute the sets in the order: \text{out}[4], \text{in}[4], \text{out}[3], \text{in}[3], \ldots

The order of computation should follow the direction of flow

Example (cont)

Data-flow Equations for Liveness
\[
\text{in}[n] = \text{use}[n] \cup (\text{out}[n] - \text{def}[n])
\]
\[
\text{out}[n] = \bigcup_{s \in \text{succ}[n]} \text{in}[s]
\]

Iterating Through the Flow Graph Backwards

Converges much faster!
Solving the Data-flow Equations (reprise)

Algorithm

for each node n in CFG
in\[n\] = ∅; out\[n\] = ∅

repeat

for each node n in CFG in reverse post dfs order
in\[n\] = in\[n\]
out\[n\] = out\[n\]

out\[n\] = ∪ \in\[s\]
in\[n\] = use\[n\] ∪ (out\[n\] – def\[n\])

until in\[n\] = in\[n\] and out\[n\] = out\[n\] for all n

Initialize solutions
Save current results
Solve data-flow equations
Test for convergence

Effects on complexity

Repeat loop
- Conservative upper bound is that each iteration will only make one set larger by one element.
- A better approximation for separable analyses is height of the lattice (k) times the depth of the graph (Knuth 1971, depth is 2.75 on average).
- Now repeat loop complexity is k instead of N^2.

Representation of sets
- For dense sets, use a bit vector representation
- Use the tuple of lattices concept to implement the data-flow sets as bit-vectors.
- Now time to execute one flow function (t) is N/wordsize.
- For sparse sets, use a sorted list (e.g., linked list)

Worklist
- Only visit nodes where the input data-flow sets have changed.
- Does not change complexity, but makes it unnecessary for convergence check iteration in IDFA.

Basic Blocks

Basic blocks
- Decrease the size of the CFG by merging nodes that have a single predecessor and a single successor into basic blocks

Concepts

Efficient Data-Flow Analysis
- Complexity analysis
- Node ordering
- Bit vector implementation
- Basic blocks
Next Time

Reading
- Ch 18.1, 19.5

Lecture
- Control dependence
- Loops
- Dominators