CS 480
Fall 2015

Mike Lam, Professor

Data-Flow Analysis

Optimization is Hard

- Problem: it's hard to analyze code and make guarantees about all possible executions
  - Inputs differ
  - Consider this code:

```
int *p; cin >> p; *p = 42;
```

- Optimization tradeoff: work vs. speedup
  - "Better than naïve" is fairly easy
  - "Optimal" is impossible
  - Real world: somewhere in between
    - Better speedups with more static analysis
    - Usually worth the added compile time

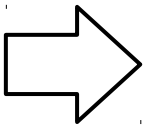
Control-Flow Graphs

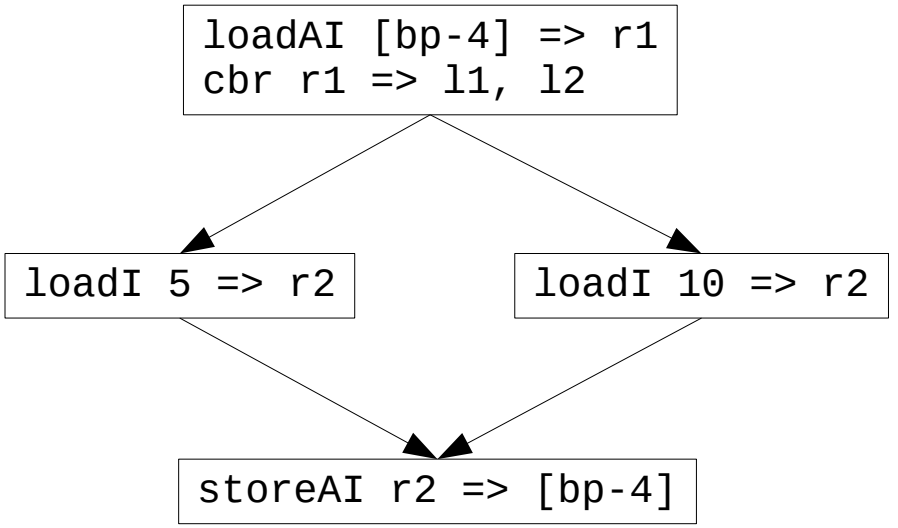
- Linear IRs (e.g., ILOC) don't easily expose control flow
  - This makes analysis and optimization difficult
- Basic blocks
  - "Maximal-length sequence of branch-free code"
  - "Atomic" code sequences
  - Instructions that always execute together
- Control-flow graph (CFG)
  - Nodes/vertices for basic blocks
  - Edges for control transfer
    - Branches (explicit) or fallthrough (implicit)

Control-Flow Graphs

- Conversion: linear IR to CFG
  - Find *leaders* (initial instruction of a basic block) and build blocks
  - Add edges between blocks based on branches and fallthrough
  - Complicated by jump-to-address instructions

```
  loadAI [bp-4] => r1
  cbr r1 => l1, l2
l1:
  loadI 5 => r2
  jump l3
l2:
  loadI 10 => r2
l3:
  storeAI r2 => [bp-4]
```





Static CFG Analysis

- Single block analysis is easy
- Trees are also relatively easy
- General CFGs, not so much
  - Which branch of a conditional will execute?
  - How many times will a loop execute?
- How do we handle this?
  - One method: iterative data-flow analysis

Data-Flow Analysis

- Define properties of interest on basic blocks
- Define formula describing how those properties change within a basic block
- Run an iterative update algorithm until the properties converge for all basic blocks
- Types of data-flow analysis
  - Dominance
  - Liveness
  - Available expressions
  - Reaching definitions
  - Anticipable expressions

Dominance

- Block A *dominates* block B if A lies on every path from the entry block to B
  - Conversely, B *postdominates* block A if B lies on every path from A to any exit

$$DOM(n) = \{n\} \cup \left(\bigcap_{m \in preds(n)} DOM(m) \right)$$

Liveness

- Variable v is *live* at point p if there is a path from p to a use of v with no intervening assignment to v

$$LIVEOUT(n) = \bigcup_{m \in succs(n)} (UEVAR(m) \cup (LIVEOUT(m) \cap VARKILL(m)))$$