Lab 2:
Finding Redundant Null Checks and Extra Credit

CS 243, Winter 2013-2014

Null Checks

- Implicit in Java bytecode
- Explicit in Quads
- There can be many redundant ones

Reasons for Redundant Null Checks

- A common design pattern to reduce the compiler complexity: each pass has one clearly defined goal and does exactly that, not more.
- Example: copy propagation can generate dead code, but it delegates the cleanup to dead code elimination.
- Joeq has a simple pass for null check insertion and relies on the following redundant elimination passes.

Example

1 MOVE T1 String, T0 String
2 NULL_CHECK T-1 <g>, T1 String
3 MOVE T2 String, T1 String
4 NULL_CHECK T-1 <g>, T0 String
5 NULL_CHECK T-1 <g>, T1 String
6 NULL_CHECK T-1 <g>, T2 String

Goal

- Print out quad ids of redundant null checks
- This is an analysis, not a transformation.
  - i.e., You don't need to remove the redundant null checks

Inside optimize.tar.gz

- FindRedundantNullChecks.java: you need to fill out this.
- We provide a few dataflow analyses some of which may be useful for redundant null checks
  - Liveness.java, ConstantProp.java, and ReachingDefinitions.java
  - Flow.java, ReferenceSolver.java

Tests

- NullTest.java
- SkipList.java
- And one secret test case

Extra Credit

- Implement optimizations that speedup SkipList.java and QuickSort.java.
- Speedup is measured by the number of quads executed.
  - (SkipList quad count reduction + QuickSort reduction)/(the best reduction in class)*50
- Extra credit will be applied after all grades are curved.

Example

myth1:~/optimize$ bin/parun optimize.OptimizeHarness --optimize optimize.test.SkipList --run-main optimize.test.SkipList --run-param 20

14 6 21 … 28 14 17

Result of interpretation: Returned: null (null checks: 29376 quad count: 111014)

Fine Prints

- Optimizations must be ***sound***
  - i.e., safe, no false positive
  - One false positive → 0 extra credit
- You are not allowed to modify OptimizeHarness.java.
  - We will measure the quad count using exactly the below:
  - bin/parun optimize.OptimizeHarness --optimize optimize.test.SkipList --run-main optimize.test.SkipList --run-param 20
  - bin/parun optimize.OptimizeHarness --optimize optimize.test.QuickSort --run-main optimize.test.QuickSort --run-param 200

Hints for Extra Credit

- The most important thing: find out the biggest ***optimization opportunities*** (Amdahl's Law).
- Don't be too ambitious: you may not make it correct within a reasonable time.
  - e.g., If you want to implement PRE, first estimate "roughly" how long it will take.
- Transformations are harder than analyses.
- Control flow modification is harder than quad-level modification: may need to modify branch instructions to be consistent.

Quad Manipulation

- Removing quads
  - QuadIterator.remove
  - BasicBlock.removeQuad
- Creating quads
  - Operator.create
  - Quad constructors
- Adding quads
  - QuadIterator.add
- Changing operands
  - Operator.setDest, Oprator.setSrc
  - Quad.setOp1/2/3/4

ControlFlowGraph Manipulation

- Removing basic blocks
  - BasicBlock.removePredecessor, BasicBlock.removePredecessors
  - BasicBlock.removeSuccessor, BasicBlock.removeSuccessors
- Creating basic blocks
  - ControlFlowGraph.createBasicBlock
- Adding basic blocks
  - BasicBlock.addPredecessor
  - BasicBlock.addSuccessor