Homework 5

Michael Faes

Compiler Design FS ‘18

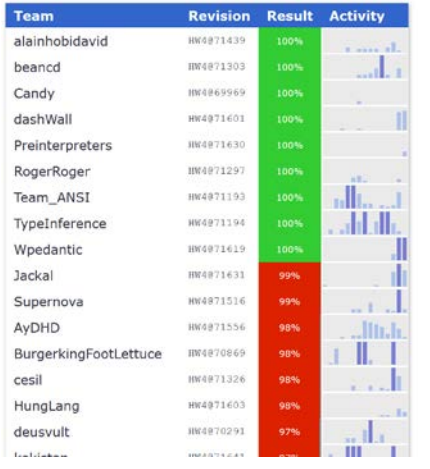
HW5 Overview

New! Project-like homework that lasts until the end of the semester!

Goal: Implement various optimizations to speed up Javali programs

The fun part: There will be a competition:

Grade for HW5 is partially based on how well your optimizations work compared to others!



Team	Revision	Result	Activity
alainhobidavid	HW4871439	100%	
beancd	HW4871303	100%	
Candy	HW4869969	100%	
dashWall	HW4871601	100%	
Preinterpreters	HW4871630	100%	
RogerRoger	HW4871297	100%	
Team_ANSI	HW4871193	100%	
TypeInference	HW4871194	100%	
Wpedantic	HW4871619	100%	
Jackal	HW4871631	99%	
Supernova	HW4871516	99%	
AyDHD	HW4871556	98%	
BurgerkingFootLettuce	HW4870869	98%	
cesil	HW4871326	98%	
HungLang	HW4871603	98%	
deusvult	HW4870291	97%	

Phase I

Optimizations in the AST

- Copy propagation
- Constant folding
- …

Tested using AST interpreter, by counting number of executed “expressions”

Due: May 22, 10:00

Phase II

Any kind of optimizations, including in generated code

- Null-check elimination
- Array-bounds-check elimin.
- Method dispatch optimiz.
- (Register allocation)
- …

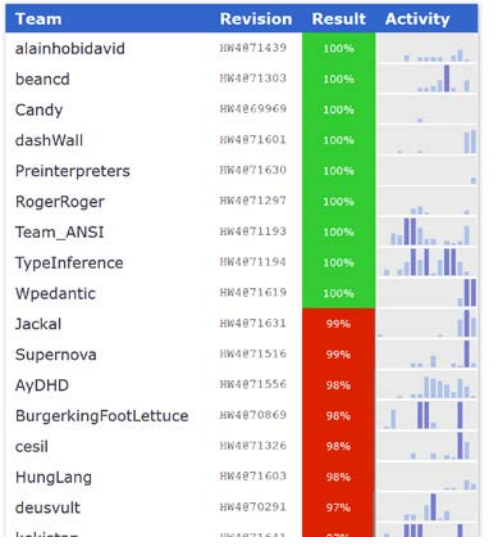
Tested by counting number of executed assembly instrs.

Due: June 1, 23:59

Grading

Optimizations are tested based on suite of benchmark programs.

- Source code not available to you, but we will give hints about what they do
- When you commit, programs will be compiled and executed, and result shown on website
- You can experiment with different approaches!



Team	Revision	Result	Activity
alainhobidavid	HW4871439	100%	
beancd	HW4871303	100%	
Candy	HW4869969	100%	
dashWall	HW4871601	100%	
Preinterpreters	HW4871630	100%	
RogerRoger	HW4871297	100%	
Team_ANSI	HW4871193	100%	
TypeInference	HW4871194	100%	
Wpedantic	HW4871619	100%	
Jackal	HW4871631	99%	
Supernova	HW4871516	99%	
AyDHD	HW4871556	98%	
BurgerkingFootLettuce	HW4870869	98%	
cesil	HW4871326	98%	
HungLang	HW4871603	98%	
deusvult	HW4870291	97%	

Correctness is still paramount!

- Compilers that produce fast but incorrect code will receive lower grades than correct ones

What's in it (the Template) for You?

1. Complete & functional code generator (solution of HW4)

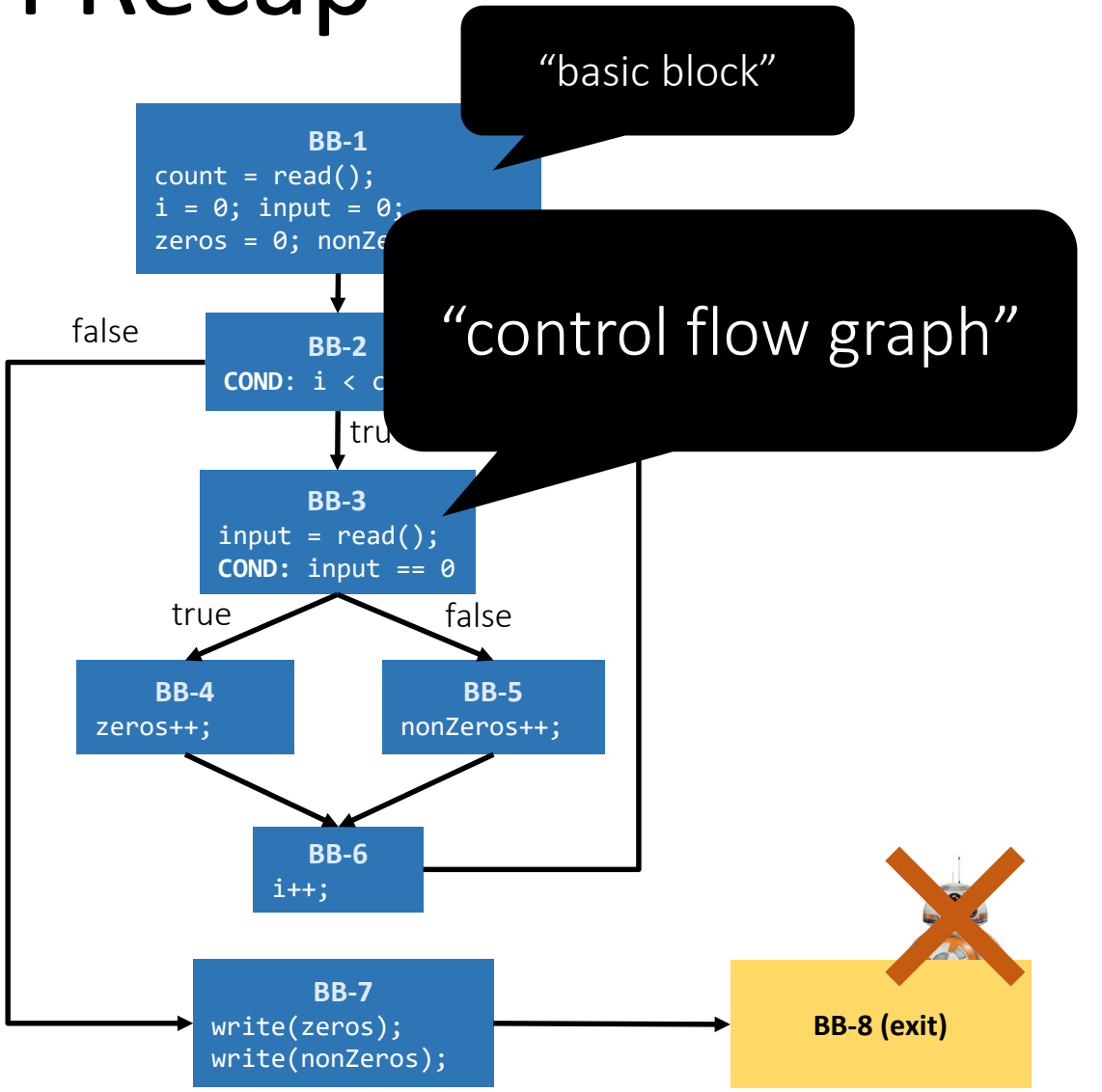
2. Control flow graph construction

3. Framework for dataflow analysis

Control Flow Graph Recap

```
class Main {
   void main() {
      int count, i, input;
      int zeros, nonZeros;
      count = read();
      i = 0; input = 0;
      zeros = 0; nonZeros = 0;

      while(i < count) {
         input = read();
         if(input == 0) {
            zeros = zeros+1;
         } else {
            nonZeros = nonZeros+1;
         }
         i = i + 1;
      }
      write(zeros);
      write(nonZeros);
   }
}
```



(actual block numbering is different)

Control Flow Graph in the Template

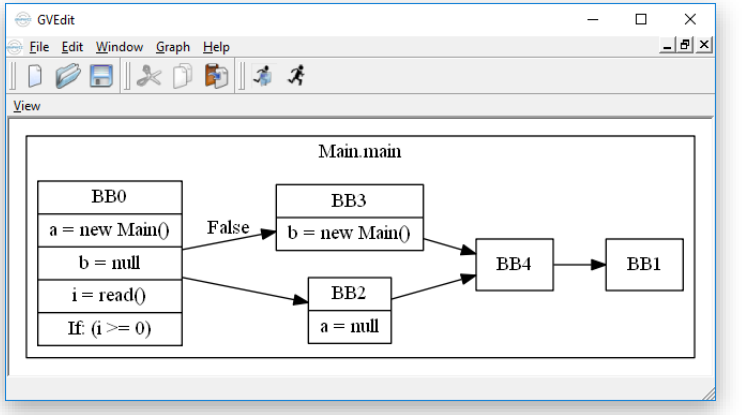
New classes in `cd.ir` package: `ControlFlowGraph` and `BasicBlock`

- Plus a new `cfg` field in MethodDecl

Class `cd.transform.CfgBuilder` that constructs CFG from AST of a method

In “debug mode”, compiler outputs the CFG into a `.cfg.dot` file

- Already the case for testing
- Use **GraphViz** to visualize



Dataflow Analysis “Framework”

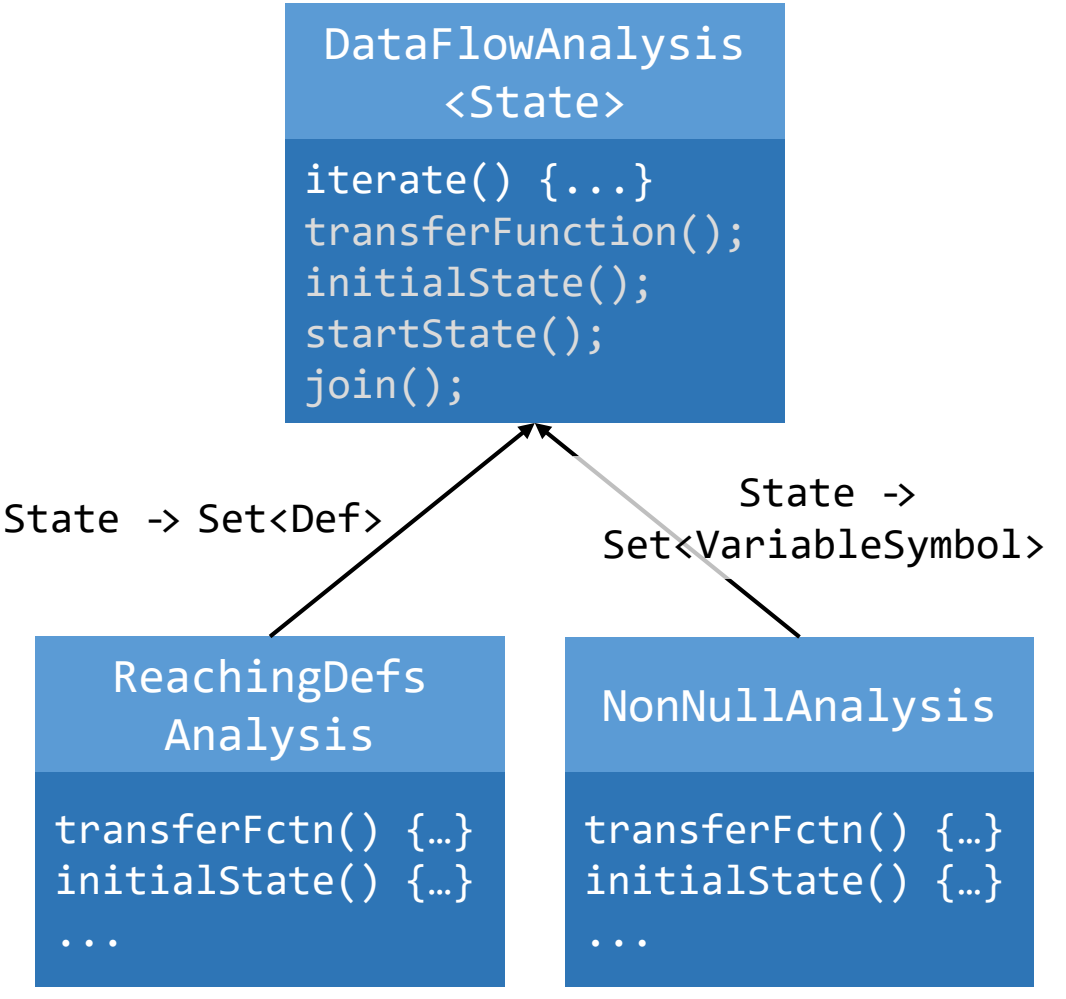
Many optimizations can be expressed as **dataflow analysis**

Template contains a framework that provides common funct.

- I.e. the **fixed-point iteration**

Look at `iterate()` method in `DataFlowAnalysis`

- Based on abstract methods `transferFunction()`, …



`DataFlowAnalysis` Class

```
public abstract class DataFlowAnalysis<State> {

   protected final ControlFlowGraph cfg;
   private Map<BasicBlock, State> inStates;
   private Map<BasicBlock, State> outStates;

   public DataFlowAnalysis(ControlFlowGraph cfg) {
      this.cfg = cfg;
   }

   /** Subclasses should call this in the constructor, after initialization */
   protected void iterate() {
      // here's the interesting stuff
      // ...
   }

   protected abstract State transferFunction(BasicBlock block, State inState);
   protected abstract State initialState();
   protected abstract State startState();
   protected abstract State join(Set<State> states);
   ...
}
```

Example: Reaching Defs

```
public class ReachingDefsAnalysis
    extends DataFlowAnalysis<Set<Def>> {

    public ReachingDefsAnalysis
            (ControlFlowGraph cfg) {
        super(cfg);
        // TODO
    }

    protected Set<Def> initialState() {
        // TODO
    }

    ...

    public static class Def {
        ...
    }
}
```

Implement simple methods:

- `initialState()`, `startState()`: return initial def sets for blocks
- `join()`: merges two def sets where control flow joins

Implement `transferFunction()`

- Uses *gen* and *kill* sets to compute effect of BB on state
- Compute *gen* and *kill* sets in `ReachingDefsAnalysis` constr. to make f.-p. iteration efficient!

Statement Granularity Information

To be useful for optimizations, need nullness info before each statement

```
x = this.foo();
x.bar();
x = new X();
x.baz();
```

Computed in additional **local analysis** after fixed-point interation:

for each *basic block*
    state = inState(*basic block*)
    for each *stmt* **in** *basic block*
        stateBefore(*stmt*) = *state*
        state = update *state* based on *stmt*
    stateBeforeCondition(*basic block*) = *state*

Questions?