Optimising Java programs with pure functions: Milestone 1 presentation

Andrew Walbran

VUW

May 15, 2008

Pure methods

- Return value depends only on state of arguments (including `this`)
- Does not alter any state

```
public class Example {
    private int x;
    public static int k;

    public int mult(int y) { \\Pure
        return x * y;
    }

    public set(int a) { \\Not pure, alters state
        x = a;
    }

    \\Not pure, depends on global state
    public int multk(int y) {
        return x * y * k;
    }
}
```

Loop invariant code motion

```
MyCollection collection = ...
for (int i = 0; i < collection.size(); ++d) {
    System.out.println(collection.get(i));
}
```

If `MyCollection.size()` is pure and nothing in the loop body alters the object referenced by `collection`, then the call can be moved out of the loop as the return value will not change.

```
MyCollection collection = ...
int size = collection.size();
for (int i = 0; i < size; ++d) {
    System.out.println(collection.get(i));
}
```

Achievements

- Literature review
- JKit parser stuff for annotations
- Drawing flowgraphs with Graphviz
- Constant folding
- Moving `@moveable` method calls
- `Array.length`
- Constructing examples of aliasing problems

Read my milestone report.

Array.length code movement

```
public class LoopArray {
  public static void main(String[] args) {
    int[] array = new int[5];

    int j = 0;
    while (j < 5) {
      int[] innerArray = new int[j];
      System.out.println("j = " + j);
      System.out.println("array length " + array.length);
      System.out.println("innerArray length "
        + innerArray.length);
      ++j;
    }
  }
}
```

Figure: LoopArray.java: program to test factoring of Array.length

Array.length code movement

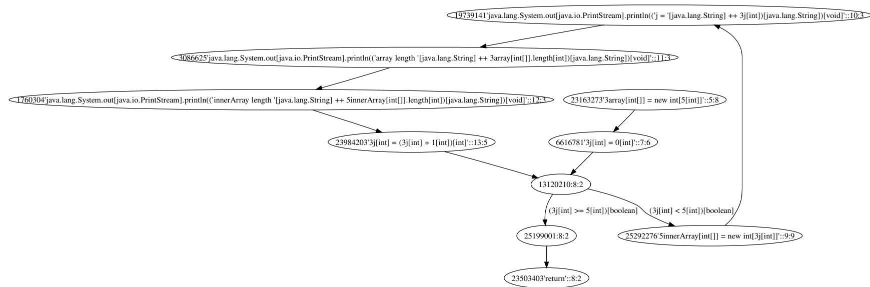

Figure: Flow graph for LoopArray before any code movement

Array.length code movement

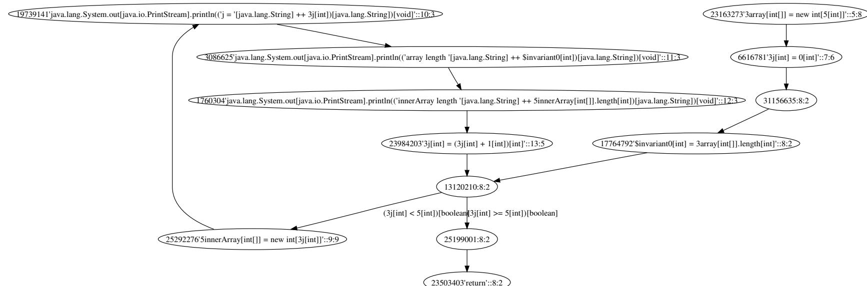

Figure: Flow graph for LoopArray after movement of `Array.length` references where safe

Problems

```
public class SimpleChanged {
    public static void main(String[] argv) {
        TestCollection list = new TestCollection();

        ...

        for (int i = 0; i < list.size(); ++i) {
            if (i % 2 == 0) {
                list.add();
            }
        }
    }
}
```

Figure: Simple case that should not be optimised, as the object is modified

Problems

```
public class AliasChanged {
    public static void main(String[] argv) {
        TestCollection list = new TestCollection();
        TestCollection alias = list;

        ...

        for (int i = 0; i < list.size(); ++i) {
            if (i % 2 == 0) {
                alias.add();
            }
        }
    }
}
```

Figure: Case that should not be optimised, as the object is modified via a local alias

Problems

```
public class IndirectChanged {
  public static void main(String[] argv) {
    TestCollection list = new TestCollection();
    Changer changer = new Changer(list);
    ...
    for (int i = 0; i < list.size(); ++i) {
      if (i % 2 == 0) changer.change();
    }
  }
}
class Changer {
  TestCollection list;
  public Changer(TestCollection list) {this.list = list;}
  public void change() { list.add(); }
}
```

Figure: More complicated case that should not be optimised, as the object is modified via an alias inside another class

Problems

```
public class ProxyChanged {
  public static void main(String[] argv) {
    TestCollection list = new TestCollection();
    CollectionProxy proxy = new CollectionProxy(list);
    ...
    for (int i = 0; i < proxy.size(); ++i) {
      if (i % 2 == 0) list.add();
    }
  } }
class CollectionProxy {
  private final TestCollection target;
  public CollectionProxy(TestCollection target) {
    this.target = target; }
  public @Pure int size() { return target.size(); }
  ...
}
```

Figure: Another case that should not be optimised, as the object inside the proxy is modified

Aliasing is a pain

Possible approaches:

- Whole-program analysis
  - Infeasible
  - Impossible in general, due to separate compilation and dynamic loading
- Ownership?
  - Not ready yet
- Immutability
  - Workable, but limiting
- More special annotations
  - Requires more work of programmer