Class Analysis
Announcements

- HW2
  - Post question on Submitty
    - Setup, please do set this up as soon as possible!
    - Starter code, class analysis framework and worklist algorithm
    - Soot
Outline of Today’s Class

- Class analysis
- Class Hierarchy Analysis (CHA)
- Rapid Type Analysis (RTA)

- HW2 class analysis framework

- XTA analysis family (next week)
- 0-CFA (next week)
Your Homework

- A bunch of flow-insensitive, context-insensitive analyses for Java
  - RTA in HW2, other in later homework
  - Simple property space
  - Simple transfer functions
    - E.g., in fact, RTA gets rid of most CFG nodes, processes just 3 kinds of statements (i.e., CFG nodes)

- Millions of lines of code in seconds
Class Analysis

Problem statement: What are the classes of objects that a (Java) reference variable may refer to at runtime?

- Class Hierarchy Analysis (CHA)
- Rapid Type Analysis (RTA)
- XTA family of analyses
- 0-CFA
- Points-to Analysis (PTA)
Applications of Class Analysis

- Call graph construction
  - At virtual call $r.m()$, what methods may be called? (Assuming $r$ is of static type $A$.)
    - $r : E, D$
    - $r.m() : B.m()$

Call graph
- Nodes are methods
- Edges represent calling relationships
- Notion of methods reachable from `main`
Applications of Class Analysis

- **Virtual call** resolution
  - If analysis proves that a virtual call has a single target, it can replace it with a **direct call**
  - An OOPSLA’96 paper by Holzle and Driesen reports that C++ programs spend 5% of their time in dispatch code. For “all virtual”, it is 14%
public abstract class BoolExp {
    public boolean evaluate(Context c);
}

public class Constant extends BoolExp {
    private boolean constant;
    public boolean evaluate(Context c) {
        return constant;
    }
}

public class VarExp extends BoolExp {
    private String name;
    public boolean evaluate(Context c) {
        return c.lookup(name);
    }
}
public class AndExp extends BoolExp {
    private BoolExp left;
    private BoolExp right;

    public AndExp(BoolExp left, BoolExp right) {
        this.left = left;
        this.right = right;
    }

    public boolean evaluate(Context c) {
        return left.evaluate(c) && right.evaluate(c);
    }
}

"Ground truth given"
public class OrExp extends BoolExp {
    private BoolExp left;
    private BoolExp right;

    public OrExp(BoolExp left, BoolExp right) {
        this.left = left;
        this.right = right;
    }

    public boolean evaluate(Context c) {
        return left.evaluate(c) || right.evaluate(c);
    }
}
main() {
    Context theContext = new …
    BoolExp x = new VarExp("X");
    BoolExp y = new VarExp("Y");
    BoolExp exp = new AndExp(
        new Constant(true), new OrExp(x, y) );
    theContext.assign(x, true);
    theContext.assign(y, false);
    boolean result = exp.evaluate(theContext);
}
Call Graph Example (Partial)

main

exp.evaluate

AndExp.evaluate

left.evaluate  right.evaluate

Constant.evaluate  OrExp.evaluate

VarExp.evaluate

left.evaluate  right.evaluate
Attributed to Dean, Grove and Chambers:

Jeff Dean, David Grove, and Craig Chambers, “Optimization of OO Programs Using Static Class Hierarchy Analysis”, ECOOP’95

Simplest way of inferring information about reference variables --- just look at class hierarchy
In Java, if a reference variable r has type A, r can refer only to objects that are **concrete subclasses** of A. Denoted by **SubTypes(A)**.

- Note: refers to Java subtype, not true subtype
- Note: **SubTypes(A)** notation due to Tip and Palsberg (OOPSLA’00)

At virtual call site r.m(), we can find what methods may be called based on the hierarchy information.
Example

```java
public class A {
    public static void main() {
        A a;
        D d = new D();
        E e = new E();
        if (...) a = d; else a = e;
        a.m();
    }
}
```

```java
public class B extends A {
    public void foo() {
        G g = new G();
    }
}
```

... // no other creation sites or calls in the program
Example

```java
public class A {
    public static void main() {
        A a;
        D d = new D();
        E e = new E();
        if (...) a = d; else a = e;
        a.m();
    }
}

public class B extends A {
    public void foo() {
        G g = new G();
    }
}
```

```
SubTypes(A) = { A, B, C, D, E, G }
SubTypes(B) = { B, G }
```
Example

public class A {
    public static void main() {
        A a;
        D d = new D();
        E e = new E();
        if (...) a = d; else a = e;
        a.m();
    }
}

public class B extends A {
    public void foo() {
        G g = new G();
    }
}

Example:

SubTypes(StaticType(a)) = SubTypes(A) = { A, B, C, D, E, G }

a.m():

main
    A.m
    B.m
    C.m
    G.m
$R$ denotes the set of reachable methods

1. $\{ \text{main} \} \subseteq R$  // Algo: initialize $R$ with $\text{main}$

2. for each method $m \in R$, each virtual call $y.n(z)$ in $m$, each class $C$ in $\text{SubTypes}(\text{StaticType}(y))$ and $n'$, where $n' = \text{resolve}(C,n)$
   
   $\{ n' \} \subseteq R$  // Algo: add $n'$ to $R$

   (Practical concerns: must consider direct calls too!)
Rapid Type Analysis (RTA)

- Due to Bacon and Sweeney
  - David Bacon and Peter Sweeney, “Fast Static Analysis of C++ Virtual Function Calls”, OOPSLA ’96

- Improves on CHA

- Expands calls only if it has seen an instantiated object of the appropriate type!
Example

```java
public class A {
    public static void main() {
        A a;
        D d = new D();
        E e = new E();
        if (...) a = d; else a = e;
        a.m();
    }
}

public class B extends A {
    public void foo() {
        G g = new G();
    }
}
```

Example

RTA starts at `main`.
Records that `D` and `E` are instantiated.
At call `a.m()` looks at all CHA targets.
Expands only into target `C.m()`!
Never reaches `B.foo()`, never records `G` as being instantiated.

CSCI 4450/6450, A Milanova
RTA

$R$ is the set of reachable methods

$I$ is the set of instantiated types

1. $\{ \text{main} \} \subseteq R$  // Algo: initialize $R$ with main

2. for each method $m \in R$ and each new site new $C$ in $m$
   
   $\{ C \} \subseteq I$  // Algo: add $C$ to $I$; schedule
   // “successor” constraints
3. for each method \( m \in R \), each **virtual call** \( y.n(z) \) in \( m \), each class \( C \) in \( \text{SubTypes(StaticType(y))} \cap I \), and \( n' \), where \( n' = \text{resolve}(C,n) \)

\[
\{ n' \} \subseteq R \quad // \text{Algo: add target } n' \text{ to } R, \text{ if not already there. Schedule “successors”}
\]
Comparison

class A {
public:
    virtual int foo() { return 1; };
};
class B: public A {
public:
    virtual int foo() { return 2; };
    virtual int foo(int i) { return i+1; };
};
void main() {
    B* p = new B;
    int result1 = p->foo(1);
    int result2 = p->foo();
    A* q = p;
    int result3 = q->foo();
}

CHA resolves `result2` to `B.foo()`; however, it does not resolve `result3`.

RTA resolves `result3` to `B.foo()` because only `B` has been instantiated.
```cpp
class A {
public:
    virtual int foo() { return 1; }
};

class B: public A {
public:
    virtual int foo() { return 2; }
    virtual int foo(int i) { return i+1; }
};

void main() {
    void* x = (void*) new A;
    B* q = (B*) x;
    int result3 = q->foo();
}
```
main() {
    Context theContext = new ...;
    BoolExp x = new VarExp("X");
    BoolExp y = new VarExp("Y");
    BoolExp exp = new AndExp(
        new Constant(true), new OrExp(x, y));
    theContext.assign(x, true);
    theContext.assign(y, false);
    boolean result = exp.evaluate(theContext);
}

exp: {VarExpr, Constant, OrExp, AndExp}

exp.evaluate() = \{ And.evaluate(), Or.Evaluate(),
                Constant.toEvaluator(), Var.evaluate() \}
HW2 Class Analysis Framework

- Big picture

[Diagram]

- Override hook methods to collect necessary transfer function info
  - You need hooks for Allocation (x=new A;), Virtual Call (x=y.m(2))
  and direct call (e.g. Sw(1)).
- Define classes to represent transfer functions (i.e. Constraints)

Set Constraints map =
- main: Alloc, Alloc1, VCall
- A.m: Alloc1, VCall
- Sw = DCall, Alloc1, Alloc2
Let’s take a moment (or two, or more) to go over HW2 class analysis framework

- **Hooks**
  - E.g., `void allocStmt(SootMethod enclMethod, int allocSiteId, Node lhs, Node alloc)`

- **Transfer functions, i.e., Constraints**
  - Add Constraint classes for certain statements
  - E.g., `class Alloc implements Constraint { ... }`
  - sootConstraints map
  - resolve function