Class Analysis

1

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

2

Flow Insensitivity

- Flow-insensitive analysis discards CFG edges and computes a single solution S
- A "declarative" definition, i.e., specification:
  - Least solution S of equations S = $f_j(S) \vee S$
  - Points-to analysis is an example where such a solution makes sense!

E.g., Flow-Sensitive vs. Flow-Insensitive Constant Propagation

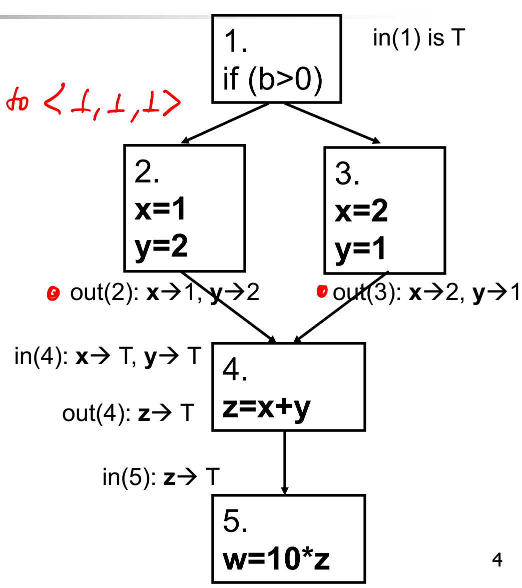
Assume we can initialize S to <⊥,⊥,⊥>

1. **x=1**
2. **y=2**
3. **x=2**
4. **y=1**
5. **z=x+y**
6. **w=10*z**

<1,⊥,⊥>

...

S = <T,T,T>



Example

- There are variations, but typically, in a flow-insensitive analysis f_j refer to statement j, not basic block j
- IRs mitigate flow insensitivity

x y z w
S = ⟨⊥, ⊥, ⊥, ⊥⟩

```
1. x=1
2. x=2
3. y=1
4. z=x+y
5. w=10*z
```

S = ⟨T, 1, T, T⟩

x1 x2 y z w
S = ⟨⊥, ⊥, ⊥, ⊥, ⊥⟩

```
1. x1=1
2. x2=2
3. y=1
4. z=x2+y
5. w=10*z
```

S = ⟨1, 2, 1, 3, 30⟩

Flow Insensitivity

- An "operational" definition. A worklist algorithm: 0 of the lattice

```
S = 0, W = { 1, 2, … n } /* all nodes */
while W ≠ Ø do {
  remove j from W
  S = fj(S) V S
  if S changed then
    W = W U { k | k is "successor" of j }
}
```

if S changed add all f_i to W

- "successor" is not CFG successor nodes, but more generally, nodes **k** whose transfer function $\mathbf{f_k}$ may be affected as a result of the change in S by **j**

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 analyses 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?

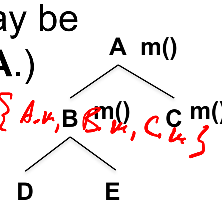
- Class Hierarchy Analysis (CHA)
- Rapid Type Analysis (RTA)
- XTA family of analyses
- 0-CFA
- Points-to Analysis (PTA)

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: {A, B, C, D, E} r.m() = {A.m, B.m, C.m}

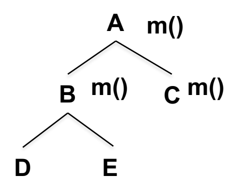
r: {D, E} r.m() = {B.m}



- Call graph
  - Nodes are methods
  - Edges represent calling relationships m1 → m2
  - 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%

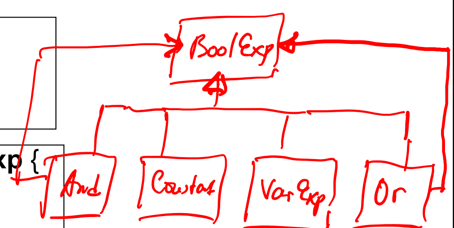


Boolean Expression Hierarchy

```
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);
    }
}
```



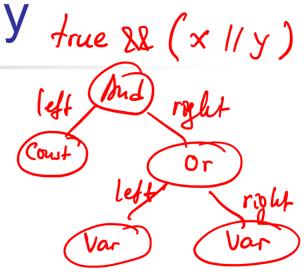
Boolean Expression Hierarchy

```
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);
   }
}
```


Boolean Expression Hierarchy

```
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);
  }
}
```

A Client of the Boolean Expression Hierarchy




```
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);
}
```

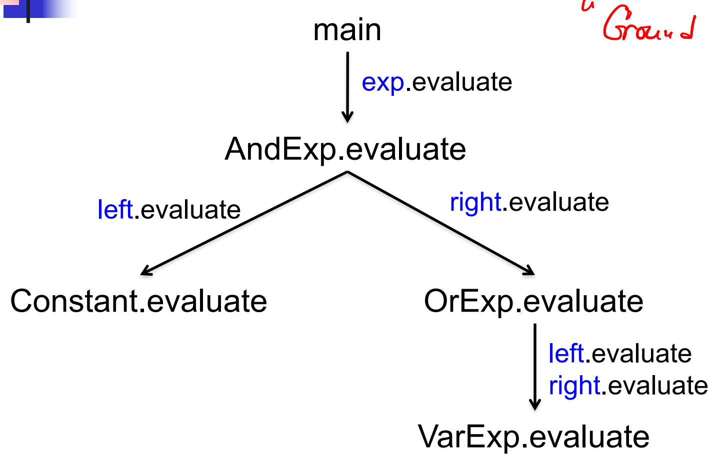
exp: {AndExp}

At runtime, exp can refer to an object of class AndExp,
but it cannot refer to objects of class OrExp, Constant or VarExp!

15

Call Graph Example (Partial)

"Ground Truth"



16

Class Hierarchy Analysis (CHA)

- 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

A r;

CSCI 4450/6450, A Milanova 17

17

Class Hierarchy Analysis (CHA)

- 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 using the hierarchy information

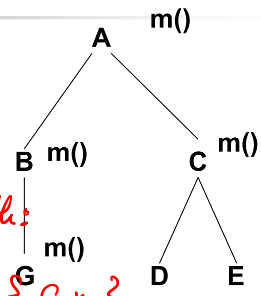
CSCI 4450/6450, A Milanova 18

18

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();
   }
} … // no other creation sites or calls in the program
```

Ground truth: a: {D, E} a.m(): {C.m}

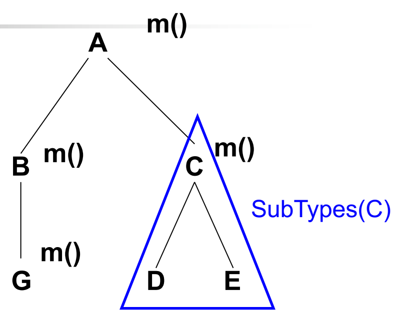


19

19

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();
   }
} …
```



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

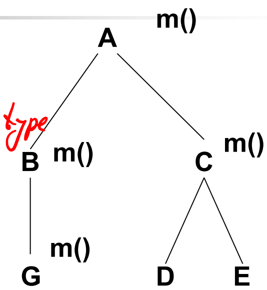
SubTypes(B) = **{ B, G }**

20

20

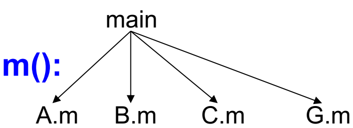
Example

```
public class A {
   public static void main() {
        A a;   Static type, also declared type
        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();
   }
} …
```



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

a.m():



21

21

CHA as Reachability Analysis

R denotes the set of reachable methods

1. { main } ⊑ **R**   // Algo: initialize **R** with **main**

2. for each method **m** ∈ **R**,
each virtual call **y.n(z)** in **m**,
each class **C** in **SubTypes(StaticType(y))** and
n', where **n'** = **resolve(C,n)**

   { **n'** } ⊑ **R**   // Algo: add **n'** to **R**

   (Practical concerns: must consider direct calls too!)

22

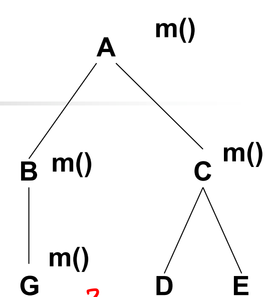
22

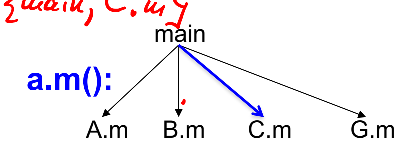
Rapid Type Analysis (RTA)

- 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

```
public class A {
   public static void main() {
        A a;
Alloc   D d = new D();     I = {D, E}
Alloc   E e = new E();     R = {main, C.m}
        if (…) a = d; else a = e;
VCall   a.m();
   }
}
public class B extends A {
   public void foo() {
        G g = new G();
   }
}
```





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.

RTA

R is the set of reachable methods

I is the set of instantiated types

1. **{** main **}** ⊑ **R**   // Algo: initialize **R** with **main**

2. for each method **m** ∈ **R** and
   each new site **new C** in **m**

   { C } ⊑ **I**   // Algo: add **C** to **I;** schedule
   // "successor" constraints

CSCI 4450/6450, A Milanova   25

25

RTA

3. for each method **m** ∈ **R**,
each virtual call **y.n(z)** in **m**,
each class **C** in **SubTypes(StaticType(y))** ∩ **I,**
and **n'**, where **n'** = **resolve(C,n)**

   { n' } ⊑ **R** // Algo: add target **n'** to **R**, if not already
   // there. Schedule "successors"

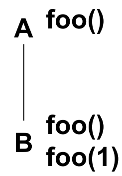
CSCI 4450/6450, A Milanova   26

26

Comparison

Bacon-Sweeny, OOPSLA' 96

```
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.

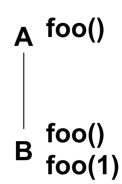
27

27

Caveat

Bacon-Sweeny, OOPSLA' 96

```
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();
}
```



28

28

RTA Example with Boolean Expression Hierarchy

```
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);
}
```

R = { main, VarExp.evaluate, … }
I = { VarExp, AndExp, Constant, OrExp }

{ VarExp, AndExp, OrExp, Constant }

for each C = cone(BoolExp) ∩ I
ds' = resolve(C, evaluate())
// { VarExp.evaluate, … }
change = true

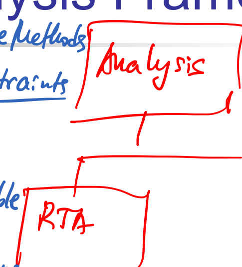
29

HW2 Class Analysis Framework

- Big picture

readableMethods
sootConstraints

worklist "hooks"

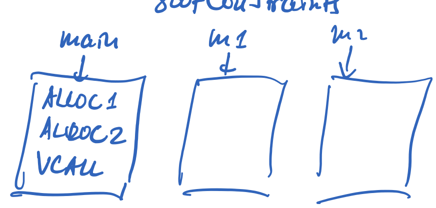


define hooks

1. Run soot
→ Soot traverses all CHA-reachable methods
→ Creates this sootConstraints map

2. Run worklistSolve

sootConstraints



30

HW2 Class Analysis Framework

- 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