Class Analysis, conclusion

1

Announcements

- Quiz 2
- HW2
  - Post question on Submitty
    - I'm assuming you all have framework set locally
    - Starter code, class analysis framework and worklist algorithm
    - Soot
  - There are already many useful posts

2

Outline of Today's Class

- Rapid Type Analysis (RTA), last time
- HW2, Class analysis framework questions?
- The XTA analysis family
- 0-CFA
- Points-to analysis (PTA)

Class Analysis

- Problem statement: What are the **classes** of objects that a (Java) **reference** variable may refer to?
- Applications
  - Call graph construction
    - Nodes are method
    - Edges represent calls
    - Notion of methods reachable from **main**
  - Virtual call resolution

RTA, A Declarative Specification

R is the set of reachable methods

I is the set of instantiated types

1. **{** main **}** $\sqsubseteq$ **R** // Algo: initialize **R** with **main**

2. for each method **m** $\in$ **R** and
   each new site **new C** in **m**
   { C } $\sqsubseteq$ **I** // Algo: add **C** to **I;** schedule
   // "successor" constraints

RTA, A Declarative Specification

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

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

Worklist Algorithm for Flow-Insensitive Analysis

- Flow-insensitive, context-insensitive analysis

S = … /* initialize S, typically to empty, which is 0 of lattice */
W = { f_1, … f_n } /* initialize W with transfer functions in **main** */
while W ≠ Ø do {
  remove f_j from W
  S = f_j(S) /* f_j never "kills" */
  if S changed
    W = W U Successors
/* Successors includes all affected transfer functions; easy safe approximation for us: include all f_j's in reachable methods */
}

XTA Analysis Family

- Due to Tip and Palsberg
  - Frank Tip and Jens Palsberg, "Scalable Propagation-Based Call Graph Construction Algorithms", OOPSLA '00
- Generalizes RTA
- Improves on RTA by keeping more info
  - What if we kept sets per method and per field rather than a "blob" **I**?

XTA

R is the set of reachable methods

S_m is the set of types that flow to method **m**

S_f is the set of types that flow to field **f**

1. **{** main **}** $\sqsubseteq$ **R**

2. for each method **m** $\in$ **R** and
   each new site **new C** in **m**
   { C } $\sqsubseteq$ S_m

9

9

XTA

3. for each method **m** $\in$ **R**,
each virtual call **y.n(z)** in **m**,
each class **C** in **SubTypes(StaticType(y))**$\cap S_m$
and **n'**, where **n' = resolve(C,n)**

   { n' } $\sqsubseteq$ **R** // add **n'** to **R** if not already there

   { C } $\sqsubseteq$ $S_{n'}$ // add **C** to $S_{n'}$ if not already there

   S_m $\cap$ **SubTypes(StaticType(p))** $\sqsubseteq$ $S_{n'}$

   $S_{n'}$ $\cap$ **SubTypes(StaticType(ret))** $\sqsubseteq$ S_m

(**p** denotes the parameter of **n'**, and **ret** denotes the return of **n'**)

10

10

XTA

4. for each method **m** $\in$ **R**,
each field read **x = y.f** in **m**

$S_f \sqsubseteq S_m$

5. for each method **m** $\in$ **R,**
each field write **x.f = y** in **m**

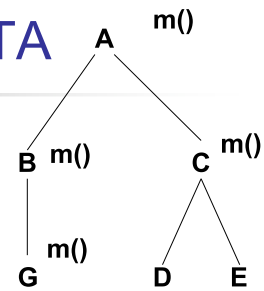
$S_m \cap \textbf{SubTypes(StaticType(f))} \sqsubseteq S_f$

Practical Concerns

- Multiple parameters
- Direct calls
  - either static invoke calls or
  - special invoke calls
- Array reads and writes!
- Static fields

- See Tip and Palsberg for more

Example: RTA vs. XTA




```
public class Main {
   public static void main() {
      n1();
      n2();
   }
   static void n1() {
      A a1 = new B();
      a1.m();
   }
   static void n2() {
      A a2 = new C();
      a2.m();
   }
}
```


Boolean Expression Hierarchy: RTA vs. XTA vs. "Ground Truth"

```
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) {
     private BoolExp l = this.left;
     private BoolExp r = this.right;
     return l.evaluate(c) && r.evaluate(c);
   }
}
```

Boolean Expression Hierarchy: RTA vs. XTA vs. "Ground Truth"

```
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) {
    private BoolExp l = this.left;
    private BoolExp r = this.right;
    return l.evaluate(c) || r.evaluate(c);
  }
}
```


Boolean Expression Hierarchy: RTA vs. XTA vs. "Ground Truth"

```
main() {
  Context theContext = new Context();
  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);
}
```

Outline of Today's Class

- Rapid Type Analysis (RTA), last time
- HW2, Class analysis framework questions?
- The XTA analysis family
- 0-CFA
- Points-to analysis (PTA)

0-CFA

- Described in Tip and Palsberg's paper
- 0-CFA stands for 0-level Control Flow Analysis, where "0-level" stands for context-insensitive analysis
  - Will see 1-CFA, 2-CFA, … k-CFA later
- Improves on XTA by storing even more information about flow of class types

0-CFA

R is the set of reachable methods

S_v is the set of types that flow to variable **v**

S_f is the set of types that flow to field **f**

1. **{** main **}** ⊑ **R**

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

   { C } ⊑ S_x

19

19

0-CFA

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

   { n' } ⊑ **R**

   { C } ⊑ S_{this}

   S_z ∩ **SubTypes(StaticType(p))** ⊑ S_p

   S_{ret} ∩ **SubTypes(StaticType(x))** ⊑ S_x

(**this** is the implicit parameter of **n'**, **p** is the parameter of **n'**, and **ret** is the return of **n'**)

20

20

0-CFA

4. for each method **m** $\in$ **R**,
each field read **x = y.f** in **m**

$S_f \cap$ **SubTypes(StaticType(x))** $\sqsubseteq S_x$

5. for each method **m** $\in$ **R,**
each field write **x.f = y** in **m**

$S_y \cap$ **SubTypes(StaticType(f))** $\sqsubseteq S_f$

0-CFA

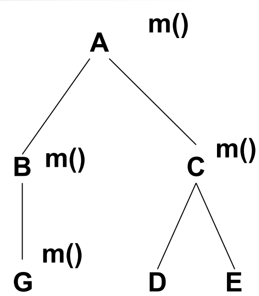
6. for each method **m** $\in$ **R**,
each assignment **x = y** in **m**

$S_y \cap$ **SubTypes(StaticType(x))** $\sqsubseteq S_x$

Example: XTA vs. 0-CFA

```
public class Main {
   public static void main() {
      A a1 = new B();
      a1.m();

      A a2 = new C();
      a2.m();
   }
}
```



Boolean Expression Hierarchy: XTA vs. 0-CFA

```
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) {
       private BoolExp l = this.left;
       private BoolExp r = this.right;
       return l.evaluate(c) && r.evaluate(c);
   }
}
```

Boolean Expression Hierarchy: XTA vs. 0-CFA

```
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) {
    private BoolExp l = this.left;
    private BoolExp r = this.right;
    return l.evaluate(c) || r.evaluate(c);
  }
}
```


Boolean Expression Hierarchy: XTA vs. 0-CFA

```
main() {
  Context theContext = new Context();
  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);
}
```

Outline of Today's Class

- Rapid Type Analysis (RTA), last time
- HW2, Class analysis framework questions?
- The XTA analysis family
- 0-CFA
- Points-to analysis (PTA)

PTA

- Widely referred to as Andersen's points-to analysis for Java
- Improves on 0-CFA by storing information about **objects**, not classes
  - A a1 = new A(); // $\mathbf{o_1}$
  - A a2 = new A(); // $\mathbf{o_2}$

PTA

R is the set of reachable methods

Pt(v) is the set of objects that **v** may point to

Pt(o.f) is the set of objects that field **f** of object **o** may point to

1. **{** main **}** $\sqsubseteq$ **R**

2. for each method **m** $\in$ **R** and
   each new site **i: x = new C** in **m**

   { o_i } $\sqsubseteq$ Pt(x) // instead of C, we have o_i

29

29

PTA

class_of(o) returns the class of object **o**

3. for each method **m** $\in$ **R**,
each virtual call **x = y.n(z)** in **m**,
each class **o_i** in **Pt(y)**
and **n'**, where **n' = resolve(class_of(o_i),n)**

   { n' } $\sqsubseteq$ R

   { o_i } $\sqsubseteq$ Pt(this)

   Pt(z) $\cap$ SubTypes(StaticType(p)) $\sqsubseteq$ Pt(p)

   Pt(ret) $\cap$ SubTypes(StaticType(x)) $\sqsubseteq$ Pt(x)

(**this** is the implicit parameter of **n'**, **p** is the parameter of **n'**, and **ret** is the return of **n'**)

30

30

PTA

4. for each method **m** $\in$ **R**,
each field read **x = y.f** in **m**
   for each object **o** $\in$ **Pt(y)**
      Pt(o.f) $\cap$ **SubTypes(StaticType(x))** $\sqsubseteq$ **Pt(x)**

5. for each method **m** $\in$ **R,**
each field write **x.f = y** in **m**
   for each object **o** $\in$ **Pt(x)**
      Pt(y) $\cap$ **SubTypes(StaticType(f))** $\sqsubseteq$ **Pt(o.f)**

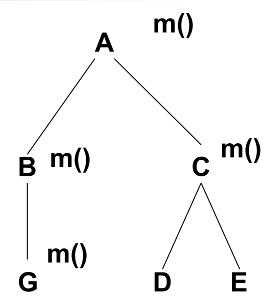
PTA

6. for each method **m** $\in$ **R**,
each assignment stmt **x = y** in **m**
      Pt(y) $\cap$ **SubTypes(StaticType(x))** $\sqsubseteq$ **Pt(x)**

Example: 0-CFA vs. PTA

```
public class Main {
   public static void main() {
      X x1 = new X();    // o1
      A a1 = new B();   // o2
      x1.f = a1;  // o1.f points to o2
      A a2 = x1.f;  // a2 points to o2
      a2.m();

      X x2 = new X();    // o3
      A a3 = new C();   // o4
      x2.f = a3;  // o3.f points to o4
      A a4 = x2.f; // a4 points to o4
      a4.m();
   }
}
```



Big Picture

- All fit into our monotone dataflow framework!
- Flow-insensitive, context-insensitive
  - Compute single solution S
- Algorithms differ mainly in "size" of S
  - RTA: only 2 kinds of statements; Lattice?
  - XTA: expands to all statements; Lattice?
  - 0-CFA: all statements; Lattice?
  - PTA (Points-to analysis): all statements; Lattice elements are points-to graphs

The Big Picture