Class Analysis, conclusion
Announcements

- Quiz 2

- HW2
  - Post question on Submitty
    - I’m assuming you all have this set up locally
    - Starter code, class analysis framework and worklist algorithm
    - Soot
  - There are already many useful posts
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)
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 calling relationships
  - 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. \( \{ \text{main} \} \subseteq R \)  // Algo: initialize \( R \) with \text{main}

2. for each method \( m \in R \) and each new site \text{new C} in \( m \)

   \( \{ C \} \subseteq I \)  // Algo: add \text{C} to \text{I}; schedule

   // “successor” constraints
3. for each method $m \in R$, each virtual call $y.n(z)$ in $m$, each class $C$ in $\text{SubTypes}(\text{StaticType}(y)) \cap I$, and $n'$, where $n' = \text{resolve}(C,n)$

\[
\{ n' \} \subseteq R \quad // \text{Algo: add target } n' \text{ to } R, \text{ if not already there. Schedule “successors”}
\]
Worklist Algorithm for Flow-Insensitive Analysis

- Flow-insensitive, context-insensitive analysis

\[ S = \ldots /* \text{initialize } S, \text{ typically to empty, which is 0 of lattice } */ \]
\[ W = \{ f_1, \ldots, f_n \} /* \text{initialize } W \text{ with transfer functions in } \text{main } */ \]

while \( W \neq \emptyset \) do {
    remove \( f_j \) from \( W \)
    \[ S = f_j(S) /* f_j \text{ never “kills” } */ \]
    if \( S \) changed
        \[ W = W \cup \text{Successors} \]

/* \text{Successors includes all affected transfer functions; easy safe approximation for us: include all } f_j \text{’s in reachable methods } */
Questions on HW2 class analysis framework?
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”?
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 \} $\subseteq$ R

2. for each method $m \in R$ and each new site new $C$ in $m$

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

\[
\begin{align*}
\{ n' \} & \subseteq R \quad \text{:// add } n' \text{ to } R \text{ if not already there} \\
\{ C \} & \subseteq S_{n'} \quad \text{:// add } C \text{ to } S_{n'} \text{ if not already there} \\
S_m \cap \text{SubTypes}(\text{StaticType}(p)) & \subseteq S_{n'} \\
S_{n'} \cap \text{SubTypes}(\text{StaticType}(\text{ret})) & \subseteq S_m
\end{align*}
\]

($p$ denotes the parameter of $n'$, and $\text{ret}$ denotes the return of $n'$)
4. for each method \( m \in R \), each field read \( x = y.f \) in \( m \)

\[
S_f \subseteq S_m
\]

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

\[
S_m \cap \text{SubTypes(StaticType}(f)) \subseteq S_f
\]
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
public class A {
    public static void main() {
        n1();
        n2();
    }
    static void n1() {
        A a1 = new B();
        a1.m();
    }
    static void n2() {
        A a2 = new C();
        a2.m();
    }
}
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);
    }
}
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);
}

Boolean Expression Hierarchy:
RTA vs. XTA vs. “Ground Truth”
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 Palsbserg’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
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. $\{ \text{main} \} \subseteq R$

2. for each method $m \in R$ and each new site $x = \text{new} \ C$ in $m$

$\{ C \} \subseteq S_x$
3. for each method $m \in R$, each virtual call $x = y.n(z)$ in $m$, each class $C$ in $S_y$ and $n'$, where $n' = \text{resolve}(C,n)$

\[
\begin{align*}
\{ n' \} & \subseteq R \\
\{ C \} & \subseteq S_{\text{this}} \\
S_z \cap \text{SubTypes}(\text{StaticType}(p)) & \subseteq S_p \\
S_{\text{ret}} \cap \text{SubTypes}(\text{StaticType}(x)) & \subseteq S_x
\end{align*}
\]

(this is the implicit parameter of $n'$, $p$ is the parameter of $n'$, and $\text{ret}$ is the return of $n'$)
0-CFA

4. for each method \( m \in R \),
each field read \( x = y.f \) in \( m \)
\[
S_f \cap \text{SubTypes}(\text{StaticType}(x)) \subseteq S_x
\]

5. for each method \( m \in R \),
each field write \( x.f = y \) in \( m \)
\[
S_y \cap \text{SubTypes}(\text{StaticType}(f)) \subseteq S_f
\]
6. for each method \( m \in R \), each assignment \( x = y \) in \( m \):

\[
S_y \cap \text{SubTypes}(\text{StaticType}(x)) \subseteq S_x
\]
Example: XTA vs. 0-CFA

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

![Method Call Tree]

CSCI 4450/6450, A Milanova
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(); // o1
  - A a2 = new A(); // o2
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 \} \subseteq R

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

   \{ o_i \} \subseteq Pt(x) // instead of C, we have o_i
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 \( n' = \text{resolve}(\text{class}_of(o_i), n) \)

\[
\{ n' \} \subseteq R \\
\{ o_i \} \subseteq Pt(this) \\
Pt(z) \cap \text{SubTypes}(\text{StaticType}(p)) \subseteq Pt(p) \\
Pt(ret) \cap \text{SubTypes}(\text{StaticType}(x)) \subseteq Pt(x)
\]

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

\text{class}_of(o) \text{ returns the class of object } o
4. for each method \( m \in R \), each field read \( x = y.f \) in \( m \)
   for each object \( o \in Pt(y) \)
   \[ Pt(o.f) \cap \text{SubTypes}(\text{StaticType}(x)) \subseteq Pt(x) \]

5. for each method \( m \in R \), each field write \( x.f = y \) in \( m \)
   for each object \( o \in Pt(x) \)
   \[ Pt(y) \cap \text{SubTypes}(\text{StaticType}(f)) \subseteq Pt(o.f) \]
6. for each method \( m \in R \), each assignment stmt \( x = y \) in \( m \)

\[
\text{Pt}(y) \cap \text{SubTypes}(\text{StaticType}(x)) \subseteq \text{Pt}(x)
\]
Example: 0-CFA vs. PTA

```java
public class A {
    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();
    }
}
```
The 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

RTA: \[ I \]

Types: A B C D

XTA: \[ S_{m1} \quad S_{m2} \ldots \quad S_{mk} \quad S_{f1} \ldots \quad S_{fk} \]

A B C D

0-CFA: \[ v_1, v_2, \ldots \quad v_n \]

A B C D

PTA: \[ v_1, v_2, \ldots \quad v_n \]

o_1:A o_2:A o_3:B o_4:B o_5:C o_6:D