Dataflow Analysis: Class Analysis (conclusion)
Announcements

- Quiz 2

- HW2
  - Post question on Submitty
    - Setup, please do set this up as soon as possible!
    - Starter code, class analysis framework and worklist algorithm
    - Soot
  - There are already some useful posts
Announcements

Office hours change
- Mondays 4-5pm on Webex
- Fridays 4-5pm on Webex
- Mondays and Thursdays 2pm - in SAGE 3713
- or Wednesdays 3:30pm - 4:30pm
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)
Your Homework

- A bunch of flow-insensitive, context-insensitive analyses for Java
  - RTA, XTA, and optionally other
  - Simple property space
  - Simple transfer functions
    - E.g., in fact, RTA gets rid of most CFG nodes, processes just 2 kinds of nodes
- Millions of lines of code in seconds
“Classical” Points-to Analysis (Imperative, i.e., “operational”)

- Flow-insensitive, context-insensitive analysis
  - Makes sense for points-to analysis

\[ \text{Pt} = 0 /* initialize solution to empty points-to graph */ \]

\[ \text{F} = \{ f_1, f_2, \ldots, f_n \} /* all transfer functions, without “kills“, including ones for implicit assignments */ \]

\[ \text{W} = \{ f_1, f_2, \ldots, f_n \} \]

while \( W \neq \emptyset \) do {
    remove \( f_j \) from \( W \)
    \[ \text{Pt} = f_j(\text{Pt}) \]
    if \( \text{Pt} \) changed then
        \[ \text{W} = \text{W} \cup \text{F} /* Safe to add all transfer functions! */ \]
}
“Classical” Points-to Analysis (vs. Declarative)

- Known as Andersen’s Points-to Analysis

\( \text{pts}(p) \) denotes the points-to set of \( p \)

1. \( p = &a \quad \{ a \} \subseteq \text{pts}(p) \)
2. \( p = q \quad \text{pts}(q) \subseteq \text{pts}(p) \)
3. \( p = *q \quad \text{for each } x \text{ in } \text{pts}(q). \quad \text{pts}(x) \subseteq \text{pts}(p) \)
4. \( *p = q \quad \text{for each } x \text{ in } \text{pts}(p). \quad \text{pts}(q) \subseteq \text{pts}(x) \)

Use \textit{worklist-like algorithm} to compute least solution of these constraints
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

CSCI 4450/6450, A Milanova
RTA

\( R \) is the set of reachable methods
\( I \) is the set of instantiated types

1. \{ main \} \subseteq R // Algo: initialize \( R \) with \texttt{main}

2. for each method \( m \in R \) and each new site \texttt{new C} in \( m \)
   \{ C \} \subseteq I // Algo: add \texttt{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 $\text{SubTypes}(\text{StaticType}(y)) \cap I$, and $n'$, where $n' = \text{resolve}(C,n)$

$$\{ n' \} \subseteq R \quad // \quad \text{Algo: add target } n' \text{ to } R, \text{ if not already there. Schedule "successors"}$$
Let’s take a moment (or two) to go over HW2 class analysis framework
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. $\{\text{main}\} \subseteq R$

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

\[
\{ n' \} \subseteq R \quad // \text{add } n' \text{ to } R \text{ if not already there}
\]

\[
\{ C \} \subseteq S_{n'} \quad // \text{add } C \text{ to } S_{n'} \text{ if not already there}
\]

$S_m \cap \text{SubTypes}(\text{StaticType}(p)) \subseteq S_{n'}$

$S_{n'} \cap \text{SubTypes}(\text{StaticType}(\text{ret})) \subseteq S_m$

($p$ 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
Example: RTA vs. XTA

```java
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();
    }
}
```

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: 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);
    }
}
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 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
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 \} \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 \( n' = \text{resolve}(C,n) \):

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

(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();
    }
}
```
```java
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);
    }
}
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

$\mathbb{R}$ is the set of reachable methods

$\text{Pt}(v)$ is the set of objects that $v$ may point to

$\text{Pt}(o.f)$ is the set of objects that field $f$ of object $o$ may point to

1. \{ main \} $\subseteq \mathbb{R}$

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

   \{ $o_i$ \} $\subseteq \text{Pt}(x)$ // 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 $n' = resolve(\text{class}\_\text{of}(o_i), n)$

$$\{ n' \} \subseteq R$$

$$\{ o_i \} \subseteq Pt(this)$$

$$Pt(z) \cap \text{SubTypes}(\text{StaticType}(p)) \subseteq Pt(p)$$

$$Pt(ret) \cap \text{SubTypes}(\text{StaticType}(x)) \subseteq Pt(x)$$

(this is the implicit parameter of $n'$, $p$ is the parameter of $n'$, and $\text{ret}$ is the return of $n'$)
4. for each method \( m \in R \), each field \( x = y.f \) 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 \( x.f = y \) 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();    // o_1
        A a1 = new B();   // o_2
        x1.f = a1;  // o_1.f points to o_2
        A a2 = x1.f; // a2 points to o_2
        a2.m();

        X x2 = new X();    // o_3
        A a3 = new C();   // o_4
        x2.f = a3; // o_3.f points to o_4
        A a4 = x2.f; // a4 points to o_4
        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:

Types: A B C D

<table>
<thead>
<tr>
<th>I</th>
</tr>
</thead>
</table>

XTA:

<table>
<thead>
<tr>
<th>S_{m1}</th>
<th>S_{m2}</th>
<th>...</th>
<th>S_{mk}</th>
<th>S_{f1}</th>
<th>...</th>
<th>S_{fk}</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td></td>
<td>C</td>
<td>D</td>
<td></td>
<td>D</td>
</tr>
</tbody>
</table>

0-CFA:

<table>
<thead>
<tr>
<th>v_1, v_2, ...</th>
<th>v_n</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
</tbody>
</table>

PTA:

<table>
<thead>
<tr>
<th>v_1, v_2, ...</th>
<th>v_n</th>
</tr>
</thead>
<tbody>
<tr>
<td>o_1:A</td>
<td>o_2:A</td>
</tr>
</tbody>
</table>