Dataflow Analysis: Classical Analysis for Object-oriented Programs, cont.
Announcements

- HW1, Quiz 1 and 2 graded

- HW2 out
  - We’ve added some posts covering some common setup issues
  - Post question on
    - Setup, please do set this up as soon as possible!
    - Starter code, class analysis framework and worklist algorithm
    - Soot
Outline of Today’s Class

- Will go over HW1 Problems 2 and 3
- Class analysis framework questions?
- Rapid Type Analysis (RTA)
- 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
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
In Java, if a reference variable \( r \) has type \( A \), \( r \) can refer only to objects that are concrete subclasses of \( A \). Denoted by \textbf{SubTypes}(A).

- Note: refers to Java subtype, not true subtype
- Note: \textbf{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.
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!
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.
RTA

$R$ 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 $C$ 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 $n' = \text{resolve}(C, n)$

\[
\{ n' \} \subseteq R \quad // \text{Algo: add target } n' \text{ to } R, \text{ if not already there. Schedule “successors”}
\]
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 storing more precise information about flow of class types
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 \( n' = \text{resolve}(C,n) \)

\[
\begin{align*}
\{ n' \} & \subseteq R \quad \text{\( \text{//} \) add \( n' \) to \( R \) if not already there} \\
\{ C \} & \subseteq S_{n'} \quad \text{\( \text{//} \) add \( C \) to \( S_{n'} \) if not already there} \\
S_m \cap \text{SubTypes}(\text{StaticType}(p)) & \subseteq S_{n'} \\
S_{n'} \cap \text{SubTypes}(\text{StaticType}(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 $m$
\[
S_m \cap \text{SubTypes}($\text{StaticType}(f)) \subseteq S_f
\]
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();
    }
}
```

![Class Diagram](diagram.png)
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

- Will go over HW1 Problems 2 and 3
- Class analysis framework questions?
- Rapid Type Analysis (RTA)
- 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 next time

- 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. \{ \text{main} \} \subseteq R

2. for each method \( m \in R \) and each new site \( x = \text{new} \ C \text{ in } m \)
   \{ \text{C} \} \subseteq S_x
0-CFA

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 \( n' \))
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 $m$

$$S_y \cap \text{SubTypes(StaticType}(x)) \subseteq S_x$$
public class A {
    public static void main() {
        A a1 = new B();
        a1.m();

        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);
    }
}
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:
XTA vs. 0-CFA
Outline of Today’s Class

- Will go over HW1 Problems 2 and 3
- Class analysis framework questions?
- Rapid Type Analysis (RTA)
- The XTA analysis family
- 0-CFA
- Points-to analysis (PTA)
Andersen’s Points-to Analysis

- Commonly attributed to Lars Andersen [1994]
  - “Andersen’s points-to analysis for C”
- More approximation than our earlier formulation: don’t ever “kill”; maintain a single points-to graph for all program points
- Flow-insensitive, context-insensitive analysis

- Formulated in terms of subset constraints
- Solvable by the worklist algorithm
Andersen’s Points-to Analysis

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

1. \( p = \&a \{ a \} \subseteq \text{pts}(p) \)
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 \textcolor{blue}{\text{worklist-like algorithm}} to compute least solution of these constraints
Andersen’s Points-to Analysis: Examples

Example 1:

\begin{align*}
p1 &= &a \\
p2 &= p1 \\
*p2 &= 1
\end{align*}
Example 2:

\[
\begin{align*}
p_3 &= \&p_1 \\
p_1 &= \&a \\
\ldots \\
q &= p_3 \\
r &= \ast q \\
p_1 &= \&b
\end{align*}
\]
a = &x;
p = &a

if (...) {
    q = &b;
    *p = q;
}

else {
    q = &c;
    *p = q;
}
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 \( i \): \( x = \text{new C in } m \)
   \( \{ o_i \} \subseteq Pt(x) \) // 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 $\text{Pt}(y)$ and $n'$, where $n' = \text{resolve}(\text{class_of}(o_i), n)$

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

(*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 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 \( m \in \mathbb{R} \), 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
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
  - Least solution of $S = f_j(S) \lor S$
- 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

- All fit into monotone dataflow framework
- Flow-insensitive, context-insensitive
  - Least solution of $S = f_j(S) \lor S$
- 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:  
\[ A \mathrm{\quad B \quad C \quad D} \]

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

\[ A \quad B \quad C \quad D \ldots \]

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

\[ A \quad B \quad C \quad D \ldots \]

PTA:  
\[ o_1:A \quad o_2:A \quad o_3:B \quad o_4:B \quad o_5:C \quad o_6:D \ldots \]

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