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

- HW3 is out
  - I will go over HW2 and provide feedback tomorrow
Outline of Today’s Class

- Rapid Type Analysis (RTA) (last time)
- The XTA analysis family (last time)
- 0-CFA
- Points-to analysis (PTA)
- Class analysis framework and HW2?
Example: RTA in HW2

```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();
    }
}
```

A \rightarrow B \rightarrow C \rightarrow G \rightarrow D \rightarrow E
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);
    }
}
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)
- The XTA analysis family (last time)
- 0-CFA
- Points-to analysis (PTA)
- Class analysis framework and HW2?
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 \( x = \text{new} \ C \) in \( m \)

\( \{ C \} \subseteq S_x \)
3. for each method $m \in R$, each virtual call $x = y \cdot n(z)$ 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 *ret* is the return of $n'$)
4. for each method \( m \in R \),
each field read \( x = y.f \) in \( m \)
\[
S_f \cap \text{SubTypes(StaticType(x))} \subseteq S_x
\]

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

Spring 21 CSCI 4450/6450, A Milanova
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);
    }
}
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);
    }
}
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)
- The XTA analysis family (last time)
- 0-CFA
- Points-to analysis (PTA)
- Class analysis framework and HW2?
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. $p = *q \quad$ for each $x$ in $\text{pts}(q)$. $\text{pts}(x) \subseteq \text{pts}(p)$
4. $*p = q \quad$ for each $x$ in $\text{pts}(p)$. $\text{pts}(q) \subseteq \text{pts}(x)$

Use worklist-like algorithm to compute least solution of these constraints
Andersen’s Points-to Analysis: Examples

Example 1:

```c
p1 = &a
p2 = p1
*p2 = 1
```
Example 2:

\[ p3 = \&p1 \]
\[ p1 = \&a \]

\[ \ldots \]
\[ q = p3 \]
\[ r = \ast q \]
\[ p1 = \&b \]
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

```java
A a1 = new A(); // o1
A a2 = new A(); // o2
```
PTA

\( R \) 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. \( \{ \text{main} \} \subseteq R \)

2. for each method \( m \in R \) 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 \( n' = \text{resolve}(\text{class}_of(o_i),n) \)

\[
\{ n' \} \subseteq R
\]

\[
\{ o_i \} \subseteq Pt(this)
\]

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

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

(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 read $x = y.f$ in $m$

for each object $o \in \text{Pt}(y)$

$\text{Pt}(o.f) \cap \text{SubTypes(StaticType}(x)) \subseteq \text{Pt}(x)$

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

for each object $o \in \text{Pt}(x)$

$\text{Pt}(y) \cap \text{SubTypes(StaticType}(f)) \subseteq \text{Pt}(o.f)$
6. for each method $m \in R$, each assignment $x = y$ in $m$

$$\text{Pt}(y) \cap \text{SubTypes(StaticType}(x)) \subseteq \text{Pt}(x)$$
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();
    }
}
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
Big Picture

RTA:    I

Types:  A B C D

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

A B C D \ldots

0-CFA:  v_1, v_2, \ldots v_n

A B C D \ldots

PTA:    v_1, v_2, \ldots v_n

o_1:A o_2:A o_3:B o_4:B o_5:C o_6:D \ldots
Outline of Today’s Class

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