IC637 Program Analysis

Lecture 5 Review

Minseok Jeon

2025 Fall

Pointer Analysis

- Pointer analysis computes the set of memory locations (objects) that a pointer variable may point to at runtime.

```
Object f(){
  x = new A(); //l1
  return x;
}

Object g(){
  y = new B(); //l2
  return y;
}

void main() {
  Object p = f();
  Object q = g();
}
```

$$x \to \{l_1\} \quad y \to \{l_2\} \quad p \to \{l_1\} \quad q \to \{l_2\}$$

cf) Flow Sensitivity

- A flow-sensitive analysis maintains abstract states separately for each program point: e.g.,

$$1 : x = A()//l_1 \qquad 1 : x \to \{l_1\}$$
$$2 : x = B()//l_2 \qquad 2 : x \to \{l_2\}$$

- Pointer analysis is often defined flow-insensitively

$$1 : x = A()//l_1$$
$$2 : x = B()//l_2 \qquad x \to \{l_1, l_2\}$$

Constraint-based Analysis

- Pointer analysis is expressed as subset constraints. The analysis is to compute the smallest solution of the constraints.

$$\begin{array}{lcl} x = A()//l_1 & & \{l_1\} \subseteq pts(x) \\ y = x & \Rightarrow & pts(x) \subseteq pts(y) \end{array}$$

- We use the Datalog language to express such constraints

Example

```
main(){ //m1
  a = A(); // l1
  b = B(); // l2
  c = a;
  a.f = b;
  d = c.f;
}
```

$\Rightarrow$

$$Alloc(a, l_1, m_1)$$
$$Alloc(b, l_2, m_1)$$
$$Move(c, a, m_1)$$
$$Store(a, f, b, m_1)$$
$$Load(d, c, f, m_1)$$

$\Rightarrow$

$$VarPointsTo(a, l_1)$$
$$VarPointsTo(b, l_2)$$
$$VarPointsTo(c, l_1)$$
$$FldPointsTo(l_1, f, l_2)$$
$$VarPointsTo(d, l_2)$$

Input Relations

- Input relations program (representation):

$$Alloc(var : V, heap : H, inMeth : M)$$
$$Move(to : V, from : V, inMeth : M)$$
$$Load(to : V, base : V, fld : F, inMeth : M)$$
$$Store(base : V, fld : F, from : V, inMeth : M)$$
$$FormalParam(method : M, arg : A, var : V)$$
$$FormalReturn(method : M, var : V)$$
$$ActualParam(invocation : I, arg : A, var : V)$$
$$ActualReturn(invocation : I, var : V)$$
$$\dots$$
$$VirtualCall(base : V, sig : S, invo : I, inMeth : M)$$
$$SpecialCall(base : V, invo : I, toMeth : M, inMeth : M)$$
$$StaticCall(invo : I, toMeth : M, inMeth : M)$$

V : the set of program variables
H : the set of heap locations
F : the set of fields
M : the set of method identifiers
S : the set of method signatures (names)
I : the set of instructions
T : the set of class types
$\mathbb{N}$: the set of natural numbers

Output Relations

- Output relations (analysis results):

$$VarPointsTo(var : V, heap : H)$$
$$FldPointsTo(baseH : H, fld : F, heap : H)$$
$$CallGraph(invo : I, meth : M)$$

Quiz

- Does pointer analysis always terminate? If so or not, why?

Analysis Rules

(1) $VarPointsTo(var,\ heap) \leftarrow$

$$Alloc(var,\ heap,\ inMeth), CallGraph(\_,\ inMeth)$$

(2) $VarPointsTo(to,\ heap) \leftarrow$

$$Move(to,\ from,\ inMeth),\ VarPointsTo(from,\ heap), CallGraph(\_,\ inMeth)$$

(3) $FldPointsTo(baseH,\ fld,\ heap) \leftarrow$

$$Store(base,\ fld,\ from,\ inMeth),\ VarPointsTo(from,\ heap),$$
$$VarPointsTo(base,\ baseH),\ CallGraph(\_,\ inMeth)$$

(4) $VarPointsTo(to,\ heap) \leftarrow$

$$Load(to,\ base,\ fld,\ inMeth),\ VarPointsTo(base,\ baseH),$$
$$FldPointsTo(baseH,\ fld,\ heap),\ CallGraph(\_,\ inMeth)$$

Analysis Rules

$$
\begin{aligned}
&(5)\ VarPointsTo(this,\ heap),\ CallGraph(invo,\ toMeth) \leftarrow \\
&\qquad VirtualCall(base,\ sig,\ invo,\ inMeth), VarPointsTo(base,\ BaseH), \\
&\qquad CallGraph(\_,\ inMeth), HeapType(BaseH,\ type), \\
&\qquad LookUp(type,\ sig,\ toMeth), ThisVar(toMeth,\ this)
\end{aligned}
$$

$$
\begin{aligned}
&(6)\ VarPointsTo(param',\ heap) \leftarrow \\
&\qquad CallGraph(invo,\ meth), FormalParam(meth,\ n,\ param'), \\
&\qquad ActualParam(invo,\ n,\ param), VarPointsTo(param,\ heap)
\end{aligned}
$$

$$
\begin{aligned}
&(7)\ VarPointsTo(return,\ heap) \leftarrow \\
&\qquad CallGraph(invo,\ meth), FormalReturn(meth,\ return'), \\
&\qquad ActualReturn(invo,\ return), VarPointsTo(return',\ heap)
\end{aligned}
$$

Analysis Rules

$$(8) CallGraph(invo,\ toMeth) \leftarrow$$
$$StaticCall(invo,\ toMeth,\ inMeth), CallGraph(\_,\ inMeth)$$

$$(9)\ VarPointsTo(this,\ heap),\ CallGraph(invo,\ toMeth) \leftarrow$$
$$SpecialCall(base,\ invo,\ toMeth,\ inMeth), CallGraph(\_,\ inMeth),$$
$$ThisVar(toMeth,\ this), VarPointsTo(base,\ heap)$$

Example 1 (Alloc)

- Compute the analysis result.

```
class A {
    int value;
    A(int value) {
        this.value = value;
    }
}
class B {
    String data;
    B(String data) {
        this.data = data;
    }
}
public class Alloc {
    public static void main(String[] args) {
        A objA1 = new A(10);
        A objA2 = new A(20);

        B objB1 = new B("Hello");
        B objB2 = new B("World");

    }
}
```

Example 2 (Move)

- Compute the analysis result.

```
class A {}

class B {}

public class Move {
    public static void main(String[] args) {
        A objA = new A();
        B objB = new B();

        Object v1;
        Object v2;

        if (args.length > 0) {
            v1 = objB;
            v2 = objA;
        } else {
            v1 = objA;
            v2 = objB;
        }
    }
}
```

Example 3 (Load)

- Compute the analysis result.

```
class A {
    Object fld;
}

class B {
    Object fld;
}

public class Load {
    public static void main(String[] args) {
        A a = new A();
        B b = new B();
        a.fld = b;
        b.fld = a;

        Object o1 = a.fld;
        Object o2 = b.fld;
        assert o1 != o2 : "o1 should not be the same object as o2";
    }
}
```

Example 4 (Store)

- Compute the analysis result.

```
class A {
    Object fld;
}

class B {
    Object fld;
}

public class Store {
    public static void main(String[] args) {
        A a = new A();
        a.fld = new B();
        ((B) (a.fld)).fld = new A();
    }
}
```

Example 5 (Static Call)

- Compute the analysis result.

```
class A {}

class B {}

public class StaticCall {
    static Object id(Object o) { return o; }
    public static void main(String[] args) {
        A objA1 = new A();
        B objB1 = new B();
        A v1 = (A) id(objA1);
        B v2 = (B) id(objB1);
    }
}
```

Example 6 (Special Call)

- Compute the analysis result.

```
class A {
    Object fld;
    A(Object value) {
        this.fld = value;
    }
}

class B {
    Object fld;
    B(Object value) {
        this.fld = value;
    }
}

class C{}

public class SpecialCall {
    static Object id(Object o) { return o; }
    public static void main(String[] args) {
        C objC = new C();
        A a = new A(objC);
        B b = new B(objC);
    }
}
```

Example 7 (Virtual Call)

- Compute the analysis result.

```
class A {}

class B {}

class C {
    Object id(Object v){
        return v;
    }
}

public class VirtualCall {
    public static void main(String[] args) {
        A a = new A();
        B b = new B();
        C c = new C();

        Object v1 = c.id(a);
        Object v2 = c.id(b);

        assert v1 != v2 : "v1 should not be the same object as v2";
    }
}
```