Dataflow Analysis: Non-distributive Analyses, Approximations, and Intro to Soot

Announcements

- HW1 due
- HW2 will be out tonight
- Get started with your Submitty git repos, Soot and Jimple, etc
- We won’t cover class analysis (RTA, XTA, etc.) today

Outline of Today’s Class

- One more note on Worklist algorithm
- Non-distributive analyses
  - Constant propagation
  - Points-to analysis
- Analysis scope and approximations
- Introduction to Soot

Worklist Algorithm for Forward Dataflow Problems

```plaintext
/* Initialize to initial values; 1 is entry node of CFG */
in(1) = InitialValue;
out(1) = \text{false};
for m := 2 to n do
  in(m) = 0;
  out(m) = \text{false};
W := \{2, \ldots, n\} /* put every node but 1 on the worklist */
while W ≠ \emptyset do {
  remove j from W
  in(j) = \bigcup \{ out(i) | i \text{ is predecessor of } j \}
  out(j) = \text{false}
  if out(j) changed then
    W = W U \{ k | k \text{ is successor of } j \}
}
```

Example. Reach with Bitvectors

Initialization

```plaintext
00000 (i,1),(k,1)
\text{pres: } 00000 11111 11111 10001 10001 01110
\text{gen: } 11000 00000 00000 00100 00010 00001
```

```plaintext
00000 (i,1),(k,1)
\text{pres: } 00000 11111 11111 10001 10001 01110
\text{gen: } 11000 00000 00000 00100 00010 00001
```

```plaintext
00000 (i,1),(k,1)
\text{pres: } 00000 11111 11111 10001 10001 01110
\text{gen: } 11000 00000 00000 00100 00010 00001
```
### Constant Propagation (Simple)

- **Problem statement:** What variables always hold constant values at a given program point.

- **Example:**
  1. \( x = 1 \) if \( b > 0 \)
  2. \( y = z + w \)
  3. \( z = 10 \times x \)
  4. \( x = \) not a constant!

### Aside: Defining an Analysis

- **If property space has desired properties**
  - is a lattice \( L \), \( \leq \) that satisfies the *Ascending Chain Condition*
  - merge operator \( V \) is the join of \( L \)
  - Function space \( F: L \rightarrow L \) is monotone then analysis fits the monotone dataflow framework and can be solved by the worklist algorithm
Constant Propagation: Property Space

- Associate one of the following values with variable \( x \) at each program point

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (or T)</td>
<td>( x ) is NOT a constant</td>
</tr>
<tr>
<td>C</td>
<td>( x ) has constant value C</td>
</tr>
<tr>
<td>0 (or ( \perp ))</td>
<td>( x ) is unknown</td>
</tr>
</tbody>
</table>

Constant Propagation: Lattice

- Lattice \( L_x \) is the product lattice of \( L_x \)
- Dataflow lattice \( L \) is the product lattice of \( L_x \)
- \( l_1, l_2 \) in \( L \), \( l_1 \leq l_2 \) iff \( l_1 \leq l_2 \) for every variable \( x \)
- \( l_1 \lor l_2 \) amounts to \( l_1 \lor l_2 \) for every variable \( x \)
- Merge operator is join of \( L \)
- Does the product lattice satisfy the ACC?

Example

1. \( in(1) \) is \( T \)
2. \( x=1 \), \( y=2 \)
3. \( x=2 \), \( y=1 \)
4. \( z=x+y \)
5. \( w=10 \times z \)

Constant Propagation: Transfer Functions

- \( j: x = V \text{ Op } V' \)
- \( f_j: \text{ kill } x \to \text{ val} \), generate \( x \to C \)
- \( j: x = y \)
- \( f_j: \text{ kill } x \to \text{ val} \), add \( x \to \text{ val} \), s.t. \( y \to \text{ val} \) in \( in(j) \). \( \text{val} \) and \( \text{val} ' \) are one of
  - \( C \): constant
  - \( T \): top (not a constant)

Constant Propagation: Transfer Functions

- \( j: x = V \text{ Op } V' \)
- \( f_j: \text{ kill } x \to \text{ val} \), generate \( x \to C \)
- \( j: x = y \)
- \( f_j: \text{ kill } x \to \text{ val} \), add \( x \to \text{ val} \), s.t. \( y \to \text{ val} \) in \( in(j) \). \( \text{val} \) and \( \text{val} ' \) are one of
  - \( C \): constant
  - \( T \): top (not a constant)

Product Lattice

- E.g., \( <x=1, y=1, z=T>, <x=1, y=2, z=3> \), etc. are lattice elements
- E.g., \( <x=1, y=2, z=T> \leq <x=T, y=2, z=T> \)
- E.g., \( <x=1, y=3, z=T> \lor <x=T, y=2, z=T> = <T, T, T> \)
Not Distributive! A Counter Example

- \( f_4(f_2(f_1(T))) \) implies \( z \rightarrow 3 \)
- \( f_4(f_3(f_1(T))) \) implies \( z \rightarrow 3 \)

Thus, MOP at 5
\[ f_4(f_2(f_1(T))) \lor f_4(f_3(f_1(T))) \]
implies \( z \rightarrow 3 \)

MFP at 5 implies \( z \rightarrow T \)
(i.e., \( z \) is NOT a const)

More Product Lattices

- Problem statement: Is integer variable \( x \) odd or even at program point \( n \)?

\[ L_x: \]

\[ \text{odd} \quad \text{even} \]

Points-to Analysis

- Problem statement: What memory locations may a pointer variable point to?

- Many applications!
  - Enables compiler optimization
    1. \( a = 1; \)
    2. \( *p = b; \)
    3. \( s = a*a; \)
  - Static debugging tools, static taint analysis tools

Points-to Analysis: Example

Example 1:
```c
int a, b;
int *p1, *p2;
p1 = &a;
p2 = p1;
*p2 = 1;
```

Example 2:
```c
int a, b = 15;
int *p1, *p2;
iint **p3;
p3 = &p1;
p1 = &a;
p2 = *p3;
*p2 = b;
```

Points-to Analysis: Syntax

- Assume the following 4 simple statements
  1. address taken \( p = \&q \)
  2. propagation \( p = q \)
  3. indirect read \( p = \&q \)
  4. indirect write (update) \( *p = q \)

- One can transform any program into a sequence of statements of these kinds
Points-to Analysis: Property Space

- Lattice $L \subseteq$ Lattice of the subsets over all edges $p \rightarrow q$ where $p$ and $q$ are program variables
- ... or in simpler terms, lattice elements are points-to graphs, e.g.,
- $V$ is points-to graph union
- $0$ of $L$ is empty graph
- $1$ of $L$ is complete graph

Spring 18 CSCI 4450/6450, A. Milanova

Points-to Analysis: Transfer Functions

1. $f_{p \& q}$: "kill" all points-to edges from $p$, and "generate" a new points-to edge from $p$ to $q$
2. $f_{p = q}$: "kill" all points-to edges from $p$; "generate" new points-to edges from $p$ to every $x$, such that $q$ points to $x$ in incoming points-to graph
3. $f_{p = *q}$: "kill" all points to edges from $p$; "generate" new points-to edges from $p$ to every $x$, s.t. there is $y$ where $q$ points to $y$ and $y$ points to $x$ in incoming points-to graph
4. $f_{p = *q}$: Do not kill! Can you think of a reason why? "Generate" new points-to edges from every $y$ to every $x$, such that $p$ points to $y$ and $q$ points to $x$

The Problem with Updates

- Updates (4) $f_{p = *q}$ (also known as destructive updates) ... are a pain
- If we drop (4) from our language we get
  1. $p = &a$ $p = \text{cons}(a,\text{null})$
  2. $p = q$ $p = q$
  3. $p = *q$ $p = \text{car}(q)$
- Research problems!

Spring 18 CSCI 4450/6450, A. Milanova

Points-to Analysis is Monotone

To argue monotonicity we must show that if $Pt_1$ is $\subseteq$ (subset of) $Pt_2$, then $f(Pt_1) \subseteq f(Pt_2)$ for each transfer function $f$

1. $Pt_1 \subseteq Pt_2$ then $f_{p \& q}(Pt_1) \subseteq f_{p \& q}(Pt_2)$
2. $Pt_1 \subseteq Pt_2$ then $f_{p = q}(Pt_1) \subseteq f_{p = q}(Pt_2)$
3. $Pt_1 \subseteq Pt_2$ then $f_{p = *q}(Pt_1) \subseteq f_{p = *q}(Pt_2)$
4. $Pt_1 \subseteq Pt_2$ then $f_{p = *q}(Pt_1) \subseteq f_{p = *q}(Pt_2)$

Points-to Analysis is Not Distributive

- but it is not distributive!
- Because of updates!

Spring 18 CSCI 4450/6450, A. Milanova
MFP vs. MOP for Points-to

1. $z := x$
2. $q := x, \quad q := y$
3. $q := z, \quad q := w$
4. $p = q$

$\text{in}_{PT}(4) = \text{out}_{PT}(3)$
$\text{out}_{PT}(4) = f_{\text{same}}(\text{in}_{PT}(4))$
$\text{in}_{PT}(5) = \text{out}_{PT}(4)$

MFP
MOP?
Ø
Ø

Outline of Today's Class

- One brief note on the Worklist algorithm
- Non-distributive analyses
  - Constant propagation
  - Points-to analysis
- Analysis scope and approximations
- Introduction to Soot

Final Note

- HW2 will be up tonight or tomorrow morning
  - Go ahead and clone your repo in Eclipse
  - Run Soot on the toy programs and get used to Jimple

- Next time:
  - Analysis scope and approximations
  - Analysis for object-oriented programs
    - CHA, RTA, XTA, PTA (points-to analysis)