CSC2/455 Software Analysis and Improvement
Foundations of Data Flow Analysis - II

Sreepathi Pai

URCS

February 18, 2019
Outline

Review

Proofs

Constant Propagation

Postscript
Outline

Review

Proofs

Constant Propagation

Postscript
Part I of Foundations

- Methods to solve dataflow analysis equations
  - IDEAL
  - Meet over paths (MOP)
  - Maximum Fixed Point (MFP)
  - \( \text{IDEAL} \subseteq \text{MOP} \subseteq \text{MFP} \)
- (Semi)lattice-based framework
  - \((D, V, \land, F)\)
  - Monotone framework
- Greatest Lower Bound
  - If \( z = x \land y \), then \( z \leq x \) and \( z \leq y \)
A given \((D, V, \land, F)\) is monotone if for all \(x, y \in V\), and \(f \in F\):

- \(x \leq y \rightarrow f(x) \leq f(y)\)
- equivalently, \(x \leq y \rightarrow f(x \land y) \leq f(x) \land f(y)\)
- The proof of equivalence is in the textbook.

In addition, the framework is *distributive* if:

- \(f(x \land y) = f(x) \land f(y)\)

Note that these properties do not necessarily arise automatically, \(F\) must be designed to have these properties

- And proofs must be written to show that \(F\) does.
- We’ll see this for a complicated example today.
General Iterative Algorithm

```python
forwards(IN, OUT, meet, top, v_entry, f_transfer)
    OUT[entry] = v_entry

    for each basic block B except ENTRY:
        OUT[B] = top

    do {
        for each basic block B except ENTRY:
            # this calculates the meet over predecessors, \( \backslash \backslash p \) OUT[p]
            IN[B] = reduce(meet, [OUT[p] for p in B.predecessors])
            OUT[B] = f_transfer(IN[B])
    } while(some OUT changes value)
```

▶ Does this calculate the solution to the dataflow problem?
▶ Does this algorithm terminate?
▶ Does this algorithm calculate the *maximum* fixed point – i.e. the most precise solution admissible?
This class

- Proofs that answer these three questions
- Relationships between IDEAL, MOP and MFP in terms of the framework
- Examples of:
  - a non-distributive framework (from Dragon 9.4, Constant Propagation)
  - lattices containing infinite values
  - possibly some proof writing exercises (from Dragon 9.3)
Proof #1

The iterative algorithm computes the solution to the dataflow problem.

- The iterative algorithm performs an unbounded number of iterations as long as IN and OUT change.
- *When it terminates*, IN and OUT have not changed for an iteration.
- The values of IN and OUT therefore satisfy the equations.
  - Hence they are solutions of the dataflow problem.
Proof #2

The iterative algorithm terminates (i.e. converges to a fix point).

- When we apply the $\land$ operator, we obtain the glb
  - i.e. $z = x \land y$ and $z \leq x$ and $z \leq y$
- Since the framework is monotone:
  - $f(x) \leq f(y)$ if $x \leq y$
  - i.e. OUT values are no greater than the IN values
- At each step, these values decrease or remain the same
  - When they all remain the same, we terminate
- If values decrease, recall the lattice has finite height
  - Implies a finite number of steps before we reach $\bot$
  - $x \land \bot = \bot$ and $f(\bot) = \bot$ (i.e once a value becomes $\bot$, it no longer changes)
  - We terminate in this case as well
Proof #3

The fixed point solution computed by the iterative algorithm is the *maximum* fixed point.

**Proof** By induction, for forward analyses (BASIS) After the first iteration, values of $\text{IN}[B]$ and $\text{OUT}[B]$ are $\leq$ their initial values.

- At initialization, $\text{OUT}[B]$ is $\top$ for all blocks $B$ except ENTRY
- After the first iteration, in a monotone framework, all values will be $\leq$ those at initialization by definitions of the $\land$ and transfer functions
Proof #3: Inductive step

Assume that:

- \( \text{IN}[B]^k \leq \text{IN}[B]^{k-1} \)
- \( \text{OUT}[B]^k \leq \text{OUT}[B]^{k-1} \)

Show that:

- \( \text{IN}[B]^{k+1} \leq \text{IN}[B]^k \)
- \( \text{OUT}[B]^{k+1} \leq \text{OUT}[B]^k \)
To obtain $\text{IN}[B]$ we must apply $\land$ to all $\text{OUT}[P]$

- $P$ is a predecessor of $B$
- This implies $\text{IN}[B] \leq \text{OUT}[P]$ ($\land$ yields glb)
- From our inductive hypothesis, $\text{OUT}[P]^k \leq \text{OUT}[P]^{k-1}$
- applying $\land$ on both sides over all $P$, $\text{IN}[B]^{k+1} \leq \text{IN}[B]^k$

Now, $\text{OUT}[B] = f(\text{IN}[B])$

- In the monotone framework, $f(x) \leq f(y)$ when $x \leq y$
- We have shown $\text{IN}[B]^{k+1} \leq \text{IN}[B]^k$
- Therefore, after applying $f$ to both sides, by monotonicity, we have $\text{OUT}[B]^{k+1} \leq \text{OUT}[B]^k$
Properties of the IDEAL solution

- Any solution greater than IDEAL is incorrect (or unsafe)
- Any solution less than or equal to IDEAL is conservative\(^1\), or safe.

To see why, consider IDEAL solution \( x = p_1 \land p_2 \land \ldots \land p_n \):

- How can we obtain a value \( z = p_1 \land \ldots \) greater than \( x \)?
- How can we obtain a value \( y = p_1 \land \ldots \) less than \( x \)?

(recall the relationship between the results of the meet operator and its operands)

\(^1\)In the English sense
Relationship between IDEAL and MOP

- MOP considers a superset of all executable paths
  - MOP solution $y = p_1 \land p_2 \land \ldots \land p_n \land p_{n+1} \ldots$
- What is the relationship between MOP ($y$) and IDEAL ($z$)?
Relationship between MOP and MFP

- **MOP**[$B_4$] = 
  \[(f_{B_3} \circ f_{B_1}) \land (f_{B_3} \circ f_{B_2})\)(v_{entry})\]

- **IN**[$B_4$] = 
  \[f_{B_3}(f_{B_1}(v_{entry}) \land f_{B_2}(v_{entry}))\]
In a distributive framework, \( \text{MOP} = \text{MFP} \)

\[ \text{MOP}[B_4] = ((f_{B_3} \circ f_{B_1}) \land (f_{B_3} \circ f_{B_2}))(v_{\text{entry}}) \]

\[ \text{IN}[B_4] = f_{B_3}(f_{B_1}(v_{\text{entry}}) \land f_{B_2}(v_{\text{entry}})) \]

If \( f(x \land y) = f(x) \land f(y) \) (i.e. distributive):

\[ \text{IN}[B_4] = f_{B_3}(f_{B_1}(v_{\text{entry}})) \land f_{B_3}(f_{B_2}(v_{\text{entry}})) \]

If the framework is distributive, then \( \text{MOP solution} = \text{MFP solution} \)

Otherwise by monotonicity \( \text{MFP} \leq \text{MOP} \)

In either case,

\[ \text{MFP} \leq \text{MOP} \leq \text{IDEAL} \]

So all methods produce "safe" solutions
Analyses so far

- Live variable analysis
- Available Expressions
- Reaching Definitions
- These are all distributive (implies monotonicity)
- Their lattices contain a finite number of values
- Their lattices have finite height
Constant Propagation

- Does this variable have a constant value at this point in the program?
  - Used to perform constant folding (i.e. evaluate constant expressions at compile time)

- Data flow analysis framework
  - Direction?
  - Values?
  - Meet operator?
  - Transfer function?
Constant Propagation

- Direction: Forward
- Values:
  - UNDEF: variable is undefined so far
  - $c$: variable is constant value $c$
  - NAC: variable is not a constant
- Meet operators and transfer functions are slightly more complicated.
Meet for Constant Propagation

- $\text{UNDEF} \land v = ?$
- $\text{NAC} \land v = ?$
- $c \land c = ?$
- $c_1 \land c_2 = ? \ (c_1 \neq c_2)$
Meet for Constant Propagation

- UNDEF $\land v = v$
  - UNDEF is $\top$
- NAC $\land v = $ NAC
  - NAC is $\bot$
- $c \land c = c$
- $c_1 \land c_2 = $ NAC

What does the lattice for constant propagation look like?
The lattice for constant propagation
The Transfer Function

-\( \text{OUT}[s] = f(\text{IN}[s]) \) for a statement \( s \)
  - Slightly easier to understand if we use statements instead of basic blocks
- Observe that non-assignment statements do not change values
  - \( f \) is simply the identify function \( f(x) = x \) for such statements
- What about assignment statements?
  - \( x = c \), where \( x \) is a variable, and \( c \) is a constant
  - \( x = y + z \), where + is any binary operator
  - \( x = *y \) or \( x = f(...) \), where \( f \) is a function call
The Transfer Function - II

- Note that **IN** (and **OUT**) are maps (i.e. dictionaries)
  - From variables to their current dataflow values (**UNDEF**, c, or **NAC**)
  - Let's call this map $m$, so that $m(x)$ returns the dataflow value for variable $x$

- $x = c$, changes $m(x) \leftarrow c$

- $x = y + z$, where $+$ is any binary operator (not just addition)
  - $m(x) \leftarrow m(y) + m(z)$ if $m(y)$ and $m(z)$ are constants
  - $m(x) \leftarrow \text{NAC}$ if either $m(y)$ or $m(z)$ is **NAC**
  - $m(x) \leftarrow \text{UNDEF}$ otherwise

- $x = *y$ or $x = f(...)$, $m(x) \leftarrow \text{NAC}$ (conservatively)

- Note that $m(v) \leftarrow m(v)$ for all $v \neq x$
  - i.e. the other values of the map remain unchanged

Note that I use slightly different notation than the textbook, which uses $m'$ on the LHS
Is this monotonic?

Is \( \text{OUT}[s] \leq \text{IN}[s] \) for every \( s \)?

- For the two cases below, it is “surely … monotone”:
  - \( m(x) \leftarrow c \)
  - \( m(x) \leftarrow \text{NAC} \)

- What about \( x = y + z \)?
  - Need to show that \( m(x) \) does not get greater as \( m(y) \) (and/or) \( m(z) \) get smaller
  - Show by case analysis and symmetry
\[ x = y + z \text{ as } m(z) \text{ gets smaller} \]

<table>
<thead>
<tr>
<th></th>
<th>( m(y) )</th>
<th>( m(z) )</th>
<th>output ( m(x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNDEF</td>
<td>UNDEF</td>
<td>NAC</td>
<td>UNDEF</td>
</tr>
<tr>
<td></td>
<td>( c_2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>NAC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( c_1 )</td>
<td>UNDEF</td>
<td>NAC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( c_2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>NAC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAC</td>
<td>UNDEF</td>
<td>NAC</td>
<td>NAC</td>
</tr>
<tr>
<td></td>
<td>( c_2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>NAC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### $x = y + z$ as $m(z)$ gets smaller (answers)

<table>
<thead>
<tr>
<th>$m(y)$</th>
<th>$m(z)$</th>
<th>Output $m(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNDEF</td>
<td>UNDEF $c_2$ NAC</td>
<td>UNDEF NAC</td>
</tr>
<tr>
<td>$c_1$</td>
<td>UNDEF $c_2$ NAC</td>
<td>UNDEF $c_1 + c_2$ NAC</td>
</tr>
<tr>
<td>NAC</td>
<td>UNDEF $c_2$ NAC</td>
<td>NAC NAC NAC</td>
</tr>
</tbody>
</table>
Is it distributive?

ENTRY

B1:
\[ x = 2 \]
\[ y = 3 \]

B2:
\[ x = 3 \]
\[ y = 2 \]

B3:
\[ z = x + y \]

EXIT
**MOP solution**

- **Path 1** ($x = 2; \ y = 3; \ z = x + y$)
  - $m(z) = 5$, so $z$ is a constant
- **Path 2** ($x = 3; \ y = 2; \ z = x + y$)
  - $m(z) = 5$, so $z$ is a constant
- **Meet over Path 1 and Path 2**
  - $m(z) = 5 \land 5$, so $z$ is a constant
MFP solution

- At end of block $B_1$
  - $m(x) = 2$ and $m(y) = 3$
- At end of block $B_2$
  - $m(x) = 3$ and $m(y) = 2$
- Meet before block $B_3$
  - $m(x) = 2 \land 3$ (i.e. case $c_1 \land c_2$)
  - $m(y) = 3 \land 2$
- Conclusion?
Constant Propagation is not distributive

- For constant propagation, in most non-trivial programs
  - MFP < MOP
Outline

Review

Proofs

Constant Propagation

Postscript
References

- Chapter 9 of the Dragon book
  - Section 9.3, 9.4