Theoretical Abstractions in Data Flow Analysis

Uday Khedker
(www.cse.iitb.ac.in/~uday)

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

August 2017
Part 1

About These Slides
Copyright

These slides constitute the lecture notes for CS618 Program Analysis course at IIT Bombay and have been made available as teaching material accompanying the book:

  (Indian edition published by Ane Books in 2013)

Apart from the above book, some slides are based on the material from the following books


*These slides are being made available under GNU FDL v1.2 or later purely for academic or research use*
Outline

- The need for a more general setting
- The set of data flow values
- The set of flow functions
- Solutions of data flow analyses
- Algorithms for performing data flow analysis
- Complexity of data flow analysis
Part 2

The Need for a More General Setting
### What We Have Seen So Far...

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Entity</th>
<th>Attribute at $p$</th>
<th>Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>Live variables</td>
<td>Variables</td>
<td>Use</td>
<td>Starting at $p$</td>
</tr>
<tr>
<td>Available expressions</td>
<td>Expressions</td>
<td>Availability</td>
<td>Reaching $p$</td>
</tr>
<tr>
<td>Partially available expressions</td>
<td>Expressions</td>
<td>Availability</td>
<td>Reaching $p$</td>
</tr>
<tr>
<td>Anticipable expressions</td>
<td>Expressions</td>
<td>Use</td>
<td>Starting at $p$</td>
</tr>
<tr>
<td>Reaching definitions</td>
<td>Definitions</td>
<td>Availability</td>
<td>Reaching $p$</td>
</tr>
<tr>
<td>Partial redundancy elimination</td>
<td>Expressions</td>
<td>Profitable hoistability</td>
<td>Involving $p$</td>
</tr>
</tbody>
</table>
The Need for a More General Setting

- We seem to have covered many variations
- Yet there are analyses that do not fit the same mould of bit vector frameworks
- We use an analysis called *Constant Propagation* to observe the differences

A variable \( v \) is a constant with value \( c \) at program point \( p \) if in every execution instance of \( p \), the value of \( v \) is \( c \)
An Introduction to Constant Propagation

\[ a = 1 \]
\[ b = 2 \]
\[ c = a + b \]

\[ c = a + b \]
\[ d = a \times b \]

\[ d = c - 1 \]
\[ a = 2 \]
\[ b = 1 \]
\[ c = a + b \]
An Introduction to Constant Propagation

\[ a = 1 \]
\[ b = 2 \]
\[ c = a + b \]

\[ c = a + b \]
\[ d = a \times b \]

\[ d = c - 1 \]
\[ a = 2 \]
\[ b = 1 \]
\[ c = a + b \]
An Introduction to Constant Propagation

\[ a = 1 \]
\[ b = 2 \]
\[ c = a + b \]

\[ c = a + b \]
\[ d = a \times b \]

\[ d = c - 1 \]
\[ a = 2 \]
\[ b = 1 \]
\[ c = a + b \]

\[ \langle a, b, c, d \rangle \]
\[ \downarrow \downarrow \downarrow \downarrow \]
\[ \langle ?, ?, ?, ? \rangle \]
\[ \langle 1, 2, 3, ? \rangle \]
An Introduction to Constant Propagation

\[ a = 1 \]
\[ b = 2 \]
\[ c = a + b \]

\[ n_1 \]

\[ c = a + b \]
\[ d = a \times b \]

\[ n_2 \]

\[ d = c - 1 \]
\[ a = 2 \]
\[ b = 1 \]
\[ c = a + b \]

\[ n_3 \]

\[ \langle a, b, c, d \rangle \]
\[ \langle ?, ?, ?, ? \rangle \]
\[ \langle 1, 2, 3, ? \rangle \]
\[ \langle 1, 2, 3, 2 \rangle \]

Execution Sequence

\[ n_1 \]
\[ n_2 \]
An Introduction to Constant Propagation

\[ a = 1 \]
\[ b = 2 \]
\[ c = a + b \]
\[ c = a + b \]
\[ d = a \times b \]
\[ d = c - 1 \]
\[ a = 2 \]
\[ b = 1 \]
\[ c = a + b \]

Execution Sequence

\[ \langle a, b, c, d \rangle \]
\[ \langle ?, ?, ?, ? \rangle \]
\[ \langle 1, 2, 3, ? \rangle \]
\[ \langle 1, 2, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
An Introduction to Constant Propagation

\[ a = 1 \]
\[ b = 2 \]
\[ c = a + b \]

\[ c = a + b \]
\[ d = a \times b \]

\[ d = c - 1 \]
\[ a = 2 \]
\[ b = 1 \]
\[ c = a + b \]

\[ \langle a, b, c, d \rangle \]
\[ \langle ?, ?, ?, ? \rangle \]
\[ \langle 1, 2, 3, ? \rangle \]
\[ \langle 1, 2, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
An Introduction to Constant Propagation

Execution Sequence

\[ \langle a, b, c, d \rangle \]
\[ \langle ?, ?, ?, ? \rangle \]
\[ \langle 1, 2, 3, ? \rangle \]
\[ \langle 1, 2, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]

\[ n_1 \]
\[ n_2 \]
\[ n_3 \]

\[ a = 1 \]
\[ b = 2 \]
\[ c = a + b \]

\[ n_2 \]
\[ c = a + b \]
\[ d = a \times b \]

\[ n_3 \]
\[ d = c - 1 \]
\[ a = 2 \]
\[ b = 1 \]
\[ c = a + b \]
An Introduction to Constant Propagation

\[\begin{align*}
  a &= 1 \\
  b &= 2 \\
  c &= a + b \\
  c &= a + b \\
  d &= a \times b \\
  d &= c - 1 \\
  a &= 2 \\
  b &= 1 \\
  c &= a + b
\end{align*}\]
An Introduction to Constant Propagation

Execution Sequence

\[ \langle a, b, c, d \rangle \]
\[ \downarrow \downarrow \downarrow \downarrow \]
\[ \langle ?, ?, ?, ? \rangle \]

\[ \langle 1, 2, 3, ? \rangle \]
\[ \downarrow \]
\[ \langle 1, 2, 3, 2 \rangle \]
\[ \downarrow \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \downarrow \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \downarrow \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \downarrow \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \downarrow \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \downarrow \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \downarrow \]
\[ \langle 2, 1, 3, 2 \rangle \]

\[ \ldots \]
An Introduction to Constant Propagation

### Summary Values
\[ \langle ?, ?, ?, ?, ? \rangle \]

### Execution Sequence
1. \( n_1 \):
   \[ \langle a, b, c, d \rangle \]
   \[ IN \]
   \[ \langle 1, 2, 3, ? \rangle \]

2. \( n_2 \):
   \[ \langle 1, 2, 3, 2 \rangle \]
   \[ OUT \]
   \[ \langle 1, 2, 3, ? \rangle \]

3. \( n_3 \):
   \[ \langle 2, 1, 3, 2 \rangle \]
   \[ \langle 2, 1, 3, ? \rangle \]

4. \( n_2 \):
   \[ \langle 2, 1, 3, 2 \rangle \]
   \[ \langle 2, 1, 3, ? \rangle \]

5. \( n_3 \):
   \[ \langle 2, 1, 3, 2 \rangle \]
   \[ \langle 2, 1, 3, ? \rangle \]

...
An Introduction to Constant Propagation

Summary Values

\( \langle ?, ?, ?, ? \rangle \)

\( \langle 1, 2, 3, ? \rangle \)

\( \langle a, b, c, d \rangle \)

Execution Sequence

IN

OUT

Aug 2017 IIT Bombay
An Introduction to Constant Propagation

Summary Values

\[ \langle ?, ?, ?, ? \rangle \]

\[ \langle 1, 2, 3, ? \rangle \]

\[ \langle \times, \times, 3, 2 \rangle \]

\[ \langle a, b, c, d \rangle \]

\[ \langle ?, ?, ?, ? \rangle \]

\[ \langle 1, 2, 3, ? \rangle \]

\[ \langle 1, 2, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]
An Introduction to Constant Propagation

Summary Values

\[ \langle ?, ?, ?, ? \rangle \]
\[ \langle 1, 2, 3, ? \rangle \]
\[ \langle \times, \times, 3, 2 \rangle \]

Execution Sequence

\[ \langle a, b, c, d \rangle \]
\[ \langle ?, ?, ?, ? \rangle \]
\[ \langle 1, 2, 3, ? \rangle \]
\[ \langle 1, 2, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
An Introduction to Constant Propagation

Summary Values

\[ \langle a, b, c, d \rangle \]

\[ \langle 1, 2, 3, ? \rangle \]

\[ \langle \times, \times, 3, 2 \rangle \]

\[ \langle \times, \times, 3, 2 \rangle \]

\[ \langle \times, \times, 3, 2 \rangle \]

\[ \langle a = 1, b = 2, c = a + b \rangle \]

\[ \langle d = a \times b \rangle \]

\[ \langle d = c - 1, a = 2, b = 1, c = a + b \rangle \]

Execution Sequence

\[ \langle a, b, c, d \rangle \]

\[ \langle 1, 2, 3, ? \rangle \]

\[ \langle 1, 2, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \ldots \]
An Introduction to Constant Propagation

Summary Values

- $a = 1$
- $b = 2$
- $c = a + b$

$n_1$

- $c = a + b$
- $d = a \times b$

$n_2$

- $d = c - 1$
- $a = 2$
- $b = 1$
- $c = a + b$

$n_3$

Execution Sequence

- $\langle a, b, c, d \rangle$
- $\langle 1, 2, 3, ? \rangle$
- $\langle 2, 1, 3, 2 \rangle$
- $\langle 2, 1, 3, 2 \rangle$
- $\langle 2, 1, 3, 2 \rangle$
- $\langle 2, 1, 3, 2 \rangle$

Aug 2017
An Introduction to Constant Propagation

Summary Values

\[ a = 1 \]
\[ b = 2 \]
\[ c = a + b \]
\[ \langle ?, ?, ?, ? \rangle \]

\[ n_1 \]

\[ c = a + b \]
\[ d = a \times b \]
\[ \langle \times, \times, 3, 2 \rangle \]
\[ \langle \times, \times, 3, 2 \rangle \]

\[ n_2 \]

\[ d = c - 1 \]
\[ a = 2 \]
\[ b = 1 \]
\[ c = a + b \]
\[ \langle 2, 1, 3, 2 \rangle \]

\[ n_3 \]

Desired Solution
Difference #1: Data Flow Values

- Tuples of the form \( \langle \eta_1, \eta_2, \ldots, \eta_k \rangle \) where \( \eta_i \) is the data flow value for \( i^{th} \) variable

Unlike bit vector frameworks, value \( \eta_i \) is not 0 or 1 (i.e. true or false). Instead, it is one of the following:

- \(?\) indicating that not much is known about the constantness of variable \( v_i \)
- \(\times\) indicating that variable \( v_i \) does not have a constant value
- An integer constant \( c_1 \) if the value of \( v_i \) is known to be \( c_1 \) at compile time
Difference #2: Dependence of Data Flow Values Across Entities

- In bit vector frameworks, data flow values of different entities are independent
Difference #2: Dependence of Data Flow Values Across Entities

- In bit vector frameworks, data flow values of different entities are independent
  - Liveness of variable $b$ does not depend on that of any other variable
  - Availability of expression $a \times b$ does not depend on that of any other expression
Difference #2: Dependence of Data Flow Values Across Entities

- In bit vector frameworks, data flow values of different entities are independent
  - Liveness of variable $b$ does not depend on that of any other variable
  - Availability of expression $a \times b$ does not depend on that of any other expression

- Given a statement $a = b \times c$, can the constantness of $a$ be determined independently of the constantness of $b$ and $c$?
Difference #2: Dependence of Data Flow Values Across Entities

- In bit vector frameworks, data flow values of different entities are independent
  - Liveness of variable $b$ does not depend on that of any other variable
  - Availability of expression $a \times b$ does not depend on that of any other expression
- Given a statement $a = b \times c$, can the constantness of $a$ be determined independently of the constantness of $b$ and $c$?

No
Difference #3: Confluence Operation

- Confluence operation \( \langle a, c_1 \rangle \sqcap \langle a, c_2 \rangle \)

<table>
<thead>
<tr>
<th>( \sqcap )</th>
<th>( \langle a, ? \rangle )</th>
<th>( \langle a, \times \rangle )</th>
<th>( \langle a, c_1 \rangle )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \langle a, ? \rangle )</td>
<td>( \langle a, ? \rangle )</td>
<td>( \langle a, \times \rangle )</td>
<td>( \langle a, c_1 \rangle )</td>
</tr>
<tr>
<td>( \langle a, \times \rangle )</td>
<td>( \langle a, \times \rangle )</td>
<td>( \langle a, \times \rangle )</td>
<td>( \langle a, \times \rangle )</td>
</tr>
<tr>
<td>( \langle a, c_2 \rangle )</td>
<td>( \langle a, c_2 \rangle )</td>
<td>( \langle a, \times \rangle )</td>
<td>If ( c_1 = c_2 ) ( \langle a, c_1 \rangle ) Otherwise ( \langle a, \times \rangle )</td>
</tr>
</tbody>
</table>

- This is neither \( \sqcap \) nor \( \sqcup \)

What are its properties?
Difference #4: Flow Functions for Constant Propagation

- Flow function for $r = a_1 * a_2$

\[
\langle a_1, c_1 \rangle, \langle a_2, c_2 \rangle \rightarrow r = a_1 * a_2
\]

<table>
<thead>
<tr>
<th>$mult$</th>
<th>$\langle a_1, ? \rangle$</th>
<th>$\langle a_1, \times \rangle$</th>
<th>$\langle a_1, c_1 \rangle$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle a_2, ? \rangle$</td>
<td>$\langle r, ? \rangle$</td>
<td>$\langle r, \times \rangle$</td>
<td>$\langle r, ? \rangle$</td>
</tr>
<tr>
<td>$\langle a_2, \times \rangle$</td>
<td>$\langle r, \times \rangle$</td>
<td>$\langle r, \times \rangle$</td>
<td>$\langle r, \times \rangle$</td>
</tr>
<tr>
<td>$\langle a_2, c_2 \rangle$</td>
<td>$\langle r, ? \rangle$</td>
<td>$\langle r, \times \rangle$</td>
<td>$\langle r, (c_1 * c_2) \rangle$</td>
</tr>
</tbody>
</table>

- This cannot be expressed in the form

\[
f_n(X) = \text{Gen}_n \cup (X - \text{Kill}_n)
\]

where $\text{Gen}_n$ and $\text{Kill}_n$ are constant effects of block $n$
Difference #5: Solution Computed by Iterative Method

\begin{align*}
  n_1 & : \\
  & a = 1 \\
  & b = 2 \\
  & c = a + b \\
  n_2 & : \\
  & c = a + b \\
  & d = a \times b \\
  n_3 & : \\
  & d = c - 1 \\
  & a = 2 \\
  & b = 1 \\
  & c = a + b
\end{align*}
Difference #5: Solution Computed by Iterative Method

Iteration #1

\[
\begin{align*}
\begin{array}{l}
\text{Iteration} \\
\#1
\end{array}
\end{align*}
\]

\[
\begin{array}{l}
\langle ?, ?, ?, ?, ? \rangle \\
\langle 1, 2, 3, ?, ? \rangle \\
\langle 1, 2, 3, ?, ? \rangle \\
\langle 1, 2, 3, 2 \rangle \\
\langle 1, 2, 3, 2 \rangle \\
\langle 2, 1, 3, 2 \rangle
\end{array}
\]

\[
\begin{array}{l}
\begin{array}{l}
n_1 \quad a = 1 \\
\quad b = 2 \\
\quad c = a + b
\end{array}
\end{array}
\]

\[
\begin{array}{l}
\begin{array}{l}
n_2 \quad c = a + b \\
\quad d = a \times b
\end{array}
\end{array}
\]

\[
\begin{array}{l}
\begin{array}{l}
n_3 \quad d = c - 1 \\
\quad a = 2 \\
\quad b = 1 \\
\quad c = a + b
\end{array}
\end{array}
\]
Difference #5: Solution Computed by Iterative Method


\[
\begin{align*}
\text{Iteration } \#1 & \quad \text{Iteration } \#2 \\
\begin{array}{l}
 a = 1 \\
 b = 2 \\
 c = a + b \\
\end{array} & \quad \begin{array}{l}
 a = 2 \\
 b = 1 \\
 c = a + b \\
\end{array} \\
\langle ?, ?, ?, ? \rangle & \quad \langle ?, ?, ?, ? \rangle \\
\langle 1, 2, 3, ? \rangle & \quad \langle 1, 2, 3, ? \rangle \\
\langle 1, 2, 3, ? \rangle & \quad \langle \times, \times, 3, 2 \rangle \\
\langle 1, 2, 3, 2 \rangle & \quad \langle \times, \times, \times, \times \rangle \\
\langle 1, 2, 3, 2 \rangle & \quad \langle \times, \times, \times, \times \rangle \\
\langle 2, 1, 3, 2 \rangle & \quad \langle 2, 1, 3, \times \rangle \\
\end{align*}
\]
Difference #5: Solution Computed by Iterative Method

<table>
<thead>
<tr>
<th>Iteration #1</th>
<th>Iteration #2</th>
<th>Iteration #3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a = 1 )</td>
<td>( 1, 2, 3, ? )</td>
<td>( 1, 2, 3, ? )</td>
</tr>
<tr>
<td>( b = 2 )</td>
<td>( 1, 2, 3, ? )</td>
<td>( 1, 2, 3, ? )</td>
</tr>
<tr>
<td>( c = a + b )</td>
<td>( 1, 2, 3, ? )</td>
<td>( 1, 2, 3, ? )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration #1</th>
<th>Iteration #2</th>
<th>Iteration #3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c = a + b )</td>
<td>( 1, 2, 3, ? )</td>
<td>( 1, 2, 3, ? )</td>
</tr>
<tr>
<td>( d = a \times b )</td>
<td>( 1, 2, 3, ? )</td>
<td>( 1, 2, 3, ? )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration #1</th>
<th>Iteration #2</th>
<th>Iteration #3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d = c - 1 )</td>
<td>( 1, 2, 3, 2 )</td>
<td>( 1, 2, 3, 2 )</td>
</tr>
<tr>
<td>( a = 2 )</td>
<td>( 1, 2, 3, 2 )</td>
<td>( 1, 2, 3, 2 )</td>
</tr>
<tr>
<td>( b = 1 )</td>
<td>( 1, 2, 3, 2 )</td>
<td>( 1, 2, 3, 2 )</td>
</tr>
<tr>
<td>( c = a + b )</td>
<td>( 2, 1, 3, 2 )</td>
<td>( 2, 1, 3, 2 )</td>
</tr>
</tbody>
</table>

Aug 2017
Difference #5: Solution Computed by Iterative Method

<table>
<thead>
<tr>
<th>n₁ ( a = 1 ) ( b = 2 ) ( c = a + b )</th>
<th>Iteration #1</th>
<th>Iteration #2</th>
<th>Iteration #3</th>
<th>Desired solution</th>
</tr>
</thead>
</table>
| \( n_2 \) \( c = a + b \) \( d = a \times b \) | \( 1, 2, 3, \_ \) \( \times, \_ , 3, 2 \) | \( 1, 2, 3, \_ \) \( \times, \_ , 3, \_ \) | \( 2, 1, 3, \_ \) \( 2, 1, 3, \_ \) | \( 1, 2, 3, ? \) \( 1, 2, 3, ? \) | }
| \( n_3 \) \( d = c - 1 \) \( a = 2 \) \( b = 1 \) \( c = a + b \) | \( 1, 2, 3, 2 \) \( \times, \times, \times, \times \) | \( 1, 2, 3, 2 \) \( \times, \times, \times, \times \) | \( 1, 2, 3, 2 \) \( \times, \times, \times, \times \) | \( 1, 2, 3, ? \) \( 1, 2, 3, ? \) |
### Difference #5: Solution Computed by Iterative Method

**Iteration #1**
- \(n_1\)
  - \(a = 1\)
  - \(b = 2\)
  - \(c = a + b\)
  - \(\langle ?, ?, ?, ?, ? \rangle\)

**Iteration #2**
- \(n_2\)
  - \(c = a + b\)
  - \(d = a \times b\)
  - \(\langle 1, 2, 3, ?, ? \rangle\)
  - \(\langle \times, \times, 3, 2 \rangle\)
  - \(\langle \times, \times, 3, \times \rangle\)

**Iteration #3**
- \(n_3\)
  - \(d = c - 1\)
  - \(a = 2\)
  - \(b = 1\)
  - \(c = a + b\)
  - \(\langle 1, 2, 3, 2 \rangle\)
  - \(\langle \times, \times, \times, \times \rangle\)
  - \(\langle \times, \times, \times, \times \rangle\)
  - \(\langle 1, 2, 3, 2 \rangle\)

**Desired solution**
- \(\langle 1, 2, 3, ? \rangle\)
- \(\langle 1, 2, 3, ? \rangle\)
- \(\langle 1, 2, 3, ? \rangle\)
- \(\langle 1, 2, 3, ? \rangle\)
Issues in Data Flow Analysis

- Data Flow Values
- Desired Solutions
- Acceptable Operations
- Practical Algorithms
Issues in Data Flow Analysis

- Representation
- Approximation: Partial Order, Lattices
Issues in Data Flow Analysis

- Representation
- **Approximation**: Partial Order, Lattices

- **Merge**: Commutativity, Associativity, Idempotence
- **Flow Functions**: Monotonicity, Distributivity, Boundedness, Separability
Issues in Data Flow Analysis

- Representation
- Approximation: Partial Order, Lattices

Data Flow
- Merge: Commutativity, Associativity, Idempotence
- Flow Functions: Monotonicity, Distributivity, Boundedness, Separability

Desired Solutions
- Existence, Computability
- Soundness, Precision

Acceptable Operations

Practical Algorithms
Issues in Data Flow Analysis

- Representation
- Approximation: Partial Order, Lattices

- Data Flow Values
- Desired Solutions
- Acceptable Operations
- Practical Algorithms

- Merge: Commutativity, Associativity, Idempotency
- Flow Functions: Monotonicity, Distributivity, Boundedness, Separability

- Existence, Computability
- Soundness, Precision

- Complexity, efficiency
- Convergence
- Initialization

Aug 2017
Part 3

Data Flow Values: An Overview
Data Flow Values: An Outline of Our Discussion

- The need to define the notion of abstraction
- Lattices, variants of lattices
- Relevance of lattices for data flow analysis
  - Partial order relation as approximation of data flow values
  - Meet operations as confluence of data flow values
- Constructing lattices
- Example of lattices
Part 4

A Digression on Lattices
Partially Ordered Sets

Sets in which elements can be compared and ordered

- **Total order.** Every element in comparable with every element (including itself)
- **Discrete order.** Every element is comparable only with itself but not with any other element
- **Partial order.** An element is comparable with some but not necessarily all elements
Partially Ordered Sets and Lattices

Partially ordered sets

Partial order $\sqsubseteq$ is reflexive, transitive, and antisymmetric
Partially Ordered Sets and Lattices

Partially ordered sets

Partial order $\sqsubseteq$ is reflexive, transitive, and antisymmetric

A lower bound of $x, y$ is $u$ s.t. $u \sqsubseteq x$ and $u \sqsubseteq y$

An upper bound of $x, y$ is $u$ s.t. $x \sqsubseteq u$ and $y \sqsubseteq u$
Partially Ordered Sets and Lattices

Partially ordered sets

Partial order $\sqsubseteq$ is reflexive, transitive, and antisymmetric

Lattices

Every non-empty finite subset has a greatest lower bound (glb) and a least upper bound (lub)
Partially Ordered Sets and Lattices

- Partially ordered sets
  - Partial order \( \sqsubseteq \) is reflexive, transitive, and antisymmetric

- Lattices
  - Every non-empty finite subset has a greatest lower bound (glb) and a least upper bound (lub)
  - glb must be related to all other lower bounds. Hence it must be unique
Partially Ordered Sets

Set \{1, 2, 3, 4, 6, 9, 12\} with \sqsubseteq \text{ relation as “divides” (i.e. } a \sqsubseteq b \text{ iff } a \text{ divides } b\)
Set \{1, 2, 3, 4, 6, 9, 12\} with \subseteq \text{ relation as “divides” (i.e. } a \subseteq b \text{ iff } a \text{ divides } b\)
Partially Ordered Sets

Set \{1, 2, 3, 4, 6, 9, 12\} with \red{\subseteq} relation as “divides” (i.e. \( a \red{\subseteq} b \) iff \( a \) divides \( b \))

\[
\begin{array}{c}
12 \\
\big/ \big/ \\
4 \quad 6 \\
\big/ \big/ \\
2 \quad 3 \\
\big/ \\
1
\end{array}
\]

Subset \{4, 9, 6\} and \{12, 9\} do not have an upper bound in the set
Lattice

Set \{1, 2, 3, 4, 6, 9, 12, 18, 36\} with \(\subseteq\) relation as “divides”
Complete Lattice

• Lattice: A partially ordered set such that every non-empty finite subset has a glb and a lub

Example: Lattice $\mathbb{Z}$ of integers under “less-than-equal-to” ($\leq$) relation
  ▶ All finite subsets have a glb and a lub
  ▶ Infinite subsets do not have a glb or a lub
Complete Lattice

- Lattice: A partially ordered set such that every non-empty finite subset has a glb and a lub

  Example: Lattice $\mathbb{Z}$ of integers under “less-than-equal-to” ($\leq$) relation
  - All finite subsets have a glb and a lub
  - Infinite subsets do not have a glb or a lub

- Complete Lattice: A lattice in which even $\emptyset$ and infinite subsets have a glb and a lub
**Complete Lattice**

- Lattice: A partially ordered set such that every non-empty finite subset has a glb and a lub

  Example: Lattice $\mathbb{Z}$ of integers under "less-than-equal-to" ($\leq$) relation
  - All finite subsets have a glb and a lub
  - Infinite subsets do not have a glb or a lub

- Complete Lattice: A lattice in which even $\emptyset$ and infinite subsets have a glb and a lub

  Example: Lattice $\mathbb{Z}$ of integers under $\leq$ relation with $\infty$ and $-\infty$
Complete Lattice

• Lattice: A partially ordered set such that every non-empty finite subset has a glb and a lub

Example: Lattice \( \mathbb{Z} \) of integers under “less-than-equal-to” (\( \leq \)) relation
  ▶ All finite subsets have a glb and a lub
  ▶ Infinite subsets do not have a glb or a lub

• Complete Lattice: A lattice in which even \( \emptyset \) and infinite subsets have a glb and a lub

Example: Lattice \( \mathbb{Z} \) of integers under \( \leq \) relation with \( \infty \) and \( -\infty \)
  ▶ \( \infty \) is the top element denoted \( \top \): \( \forall i \in \mathbb{Z}, \ i \leq \top \)
  ▶ \( -\infty \) is the bottom element denoted \( \bot \): \( \forall i \in \mathbb{Z}, \ \bot \leq i \)
$\mathbb{Z} \cup \{\infty, -\infty\}$ is a Complete Lattice

- Infinite subsets of $\mathbb{Z} \cup \{\infty, -\infty\}$ have a glb and lub
$\mathbb{Z} \cup \{\infty, -\infty\}$ is a Complete Lattice

- Infinite subsets of $\mathbb{Z} \cup \{\infty, -\infty\}$ have a glb and lub

- What about the empty set?
\( \mathbb{Z} \cup \{\infty, -\infty\} \) is a Complete Lattice

- Infinite subsets of \( \mathbb{Z} \cup \{\infty, -\infty\} \) have a glb and lub

- What about the empty set?
  - \( \text{glb}(\emptyset) \) is \( \top \)
\( \mathbb{Z} \cup \{\infty, -\infty\} \) is a Complete Lattice

- Infinite subsets of \( \mathbb{Z} \cup \{\infty, -\infty\} \) have a glb and lub

- What about the empty set?

  - \( \text{glb}(\emptyset) \) is \( \top \)
    
    Every element of \( \mathbb{Z} \cup \{\infty, -\infty\} \) is vacuously a lower bound of an element in \( \emptyset \)
    
    OR

    Every element in \( \emptyset \) is stronger than every element in \( \mathbb{Z} \cup \{\infty, -\infty\} \)
    
    (because there is no element in \( \emptyset \))
\( \mathbb{Z} \cup \{\infty, -\infty\} \) is a Complete Lattice

- Infinite subsets of \( \mathbb{Z} \cup \{\infty, -\infty\} \) have a glb and lub

- What about the empty set?
  
  \( \text{glb}(\emptyset) \) is \( \top \)
  
  Every element of \( \mathbb{Z} \cup \{\infty, -\infty\} \) is vacuously a lower bound of an element in \( \emptyset \)

  OR

  Every element in \( \emptyset \) is stronger than every element in \( \mathbb{Z} \cup \{\infty, -\infty\} \)

  (because there is no element in \( \emptyset \))

  The greatest among these lower bounds is \( \top \)
$\mathbb{Z} \cup \{\infty, -\infty\}$ is a Complete Lattice

- Infinite subsets of $\mathbb{Z} \cup \{\infty, -\infty\}$ have a glb and lub

- What about the empty set?

  - $\text{glb}(\emptyset)$ is $\top$
    Every element of $\mathbb{Z} \cup \{\infty, -\infty\}$ is vacuously a lower bound of an element in $\emptyset$
    OR
    Every element in $\emptyset$ is stronger than every element in $\mathbb{Z} \cup \{\infty, -\infty\}$
    (because there is no element in $\emptyset$)
    The greatest among these lower bounds is $\top$

  - $\text{lub}(\emptyset)$ is $\bot$
Operations on Lattices

- Meet (\(\sqcap\)) and Join (\(\sqcup\))
Operations on Lattices

- Meet ($\sqcap$) and Join ($\sqcup$)
  - $x \sqcap y$ computes the glb of $x$ and $y$
  
  $z = x \sqcap y \Rightarrow z \sqsubseteq x \land z \sqsubseteq y$
Operations on Lattices

- Meet (\( \cap \)) and Join (\( \cup \))
  - \( x \cap y \) computes the glb of \( x \) and \( y \)
    \[ z = x \cap y \Rightarrow z \subseteq x \land z \subseteq y \]
  - \( x \cup y \) computes the lub of \( x \) and \( y \)
    \[ z = x \cup y \Rightarrow z \supseteq x \land z \supseteq y \]
Operations on Lattices

- Meet ($\cap$) and Join ($\cup$)
  - $x \cap y$ computes the glb of $x$ and $y$
    $z = x \cap y \Rightarrow z \subseteq x \land z \subseteq y$
  - $x \cup y$ computes the lub of $x$ and $y$
    $z = x \cup y \Rightarrow z \supseteq x \land z \supseteq y$
  - $\cap$ and $\cup$ are commutative, associative, and idempotent
### Operations on Lattices

- **Meet (\( \sqcap \)) and Join (\( \sqcup \))**
  - \( x \sqcap y \) computes the glb of \( x \) and \( y \)
    - \( z = x \sqcap y \Rightarrow z \subseteq x \land z \subseteq y \)
  - \( x \sqcup y \) computes the lub of \( x \) and \( y \)
    - \( z = x \sqcup y \Rightarrow z \supseteq x \land z \supseteq y \)
  - \( \sqcap \) and \( \sqcup \) are commutative, associative, and idempotent

- **Top (\( \top \)) and Bottom (\( \bot \)) elements**
  
  \[
  \forall x \in L, \quad x \sqcap \top = x \\
  \forall x \in L, \quad x \sqcup \top = \top \\
  \forall x \in L, \quad x \sqcap \bot = \bot \\
  \forall x \in L, \quad x \sqcup \bot = x
  \]
Operations on Lattices

• Meet (∧) and Join (∨)
  ▶ $x \sqcap y$ computes the glb of $x$ and $y$
    $z = x \sqcap y \Rightarrow z \subseteq x \wedge z \subseteq y$
  ▶ $x \sqcup y$ computes the lub of $x$ and $y$
    $z = x \sqcup y \Rightarrow z \supseteq x \wedge z \supseteq y$
  ▶ $\sqcap$ and $\sqcup$ are commutative, associative, and idempotent

• Top (⊤) and Bottom (⊥) elements

$$
\forall x \in L, \ x \sqcap \top = x \\
\forall x \in L, \ x \sqcup \top = \top \\
\forall x \in L, \ x \sqcap \bot = \bot \\
\forall x \in L, \ x \sqcup \bot = x \\
$$

Greatest common divisor

$x \sqcap y = \text{gcd}(x, y)$
Operations on Lattices

- **Meet (⊔)** and **Join (⊓)**
  - \( x ⊓ y \) computes the glb of \( x \) and \( y \)
  - \( z = x ⊓ y \Rightarrow z ⊑ x \land z ⊑ y \)
  - \( x ⊔ y \) computes the lub of \( x \) and \( y \)
  - \( z = x ⊔ y \Rightarrow z ⊒ x \land z ⊒ y \)
  - \( ⊓ \) and \( ⊔ \) are commutative, associative, and idempotent

- **Top (⊤)** and **Bottom (⊥)** elements

\[
\begin{align*}
\forall x \in L, \ x \sqcap \top &= x \\
\forall x \in L, \ x \sqcup \top &= \top \\
\forall x \in L, \ x \sqcap \bot &= \bot \\
\forall x \in L, \ x \sqcup \bot &= x
\end{align*}
\]

- **Greatest common divisor**

\[
\begin{align*}
36 &= 3 \times 2 \times 3 \\
12 &= 3 \times 2 \times 2 \\
18 &= 3 \times 3 \times 2 \\
4 &= 2 \times 2 \\
6 &= 2 \times 3 \\
9 &= 3 \times 3 \\
2 &= 2 \\
3 &= 3 \\
1 &= 1
\end{align*}
\]

- **Lowest common multiple**

\[
\begin{align*}
x \sqcap y &= \gcd(x, y) \\
x \sqcup y &= \text{lcm}(x, y)
\end{align*}
\]
Partial Order and Operations

- For a lattice $\sqsubseteq$ induces $\sqcap$ and $\sqcup$ and vice-versa
- The choices of $\sqsubseteq$, $\sqcap$, and $\sqcup$ cannot be arbitrary
  They have to be
  - consistent with each other, and
  - definable in terms of each other
- For some variants of lattices, $\sqcap$ or $\sqcup$ may not exist
  Yet the requirement of its consistency with $\sqsubseteq$ cannot be violated
Finite Lattices are Complete

- Any given set of elements has a glb and a lub

```
Available Expressions Analysis

(⊤)
{e₁, e₂, e₃}
{e₁, e₂}  {e₁, e₃}  {e₂, e₃}
{e₁}  {e₂}  {e₃}
∅  (⊥)

Partially Available Expressions Analysis

(⊤)
∅
{e₁}  {e₂}  {e₃}
{e₁, e₂}  {e₁, e₃}  {e₂, e₃}
{e₁, e₂, e₃}
(⊥)
```
Lattice for May-Must Analysis

- There is no $\top$ among the natural values

<table>
<thead>
<tr>
<th>No</th>
<th>Must</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>May</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Interpreting data flow values</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Information does not hold along any path</td>
</tr>
<tr>
<td>Must</td>
<td>Information must hold along all paths</td>
</tr>
<tr>
<td>May</td>
<td>Information may hold along some path</td>
</tr>
</tbody>
</table>

- An artificial $\top$ can be added
Some Variants of Lattices

A poset $L$ is

- A **lattice** iff each non-empty finite subset of $L$ has a glb and lub
- A **complete lattice** iff each subset of $L$ has a glb and lub
- A **meet semilattice** iff each non-empty finite subset of $L$ has a glb
- A **join semilattice** iff each non-empty finite subset of $L$ has a lub
- A **bounded lattice** iff $L$ is a lattice and has $\top$ and $\bot$ elements
A Bounded Lattice Need Not be Complete (1)

• Let $A$ be all finite subsets of $\mathbb{Z}$
  Then, $A$ is an infinite set

• The poset $L = (A \cup \{\mathbb{Z}\}, \subseteq)$ is a bounded lattice with $\top = \mathbb{Z}$ and $\bot = \emptyset$
  The join $\sqcup$ of this lattice is $\bigcup$

• To see why, consider a set $S$ containing those subsets of $L$ that do not contain the number 1
  There are two possibilities:
  
  ▶ $S$ contains only a finite number of sets that not contain 1 (say $S_f$)
  $\Rightarrow S_f$ is a finite set

  ▶ $S$ contains all finite sets that do not contain 1 (say $S_\infty$)
  $\Rightarrow S_\infty$ is a infinite set

Aug 2017 IIT Bombay
A Bounded Lattice Need Not be Complete (2)

- $S_f$ contains only a finite number of sets each of which does not contain 1
  - The union of all its member sets is a finite set excluding 1
  - Thus $S_f$ has a lub in $L$
A Bounded Lattice Need Not be Complete (2)

- $S_f$ contains only a finite number of sets each of which does not contain 1
  - The union of all its member sets is a finite set excluding 1
  - Thus $S_f$ has a lub in $L$

- $S_\infty$ contains all finite sets that do not contain 1
  - Since the number of such sets is infinite, their union is an infinite set
  - $\mathbb{Z} - \{1\}$ is not contained in $L$ (the only infinite set in $L$ is $\mathbb{Z}$)
  - $S_\infty$ does not have a lub in $L$
A Bounded Lattice Need Not be Complete (2)

- $S_f$ contains only a finite number of sets each of which does not contain 1
  - The union of all its member sets is a finite set excluding 1
  - Thus $S_f$ has a lub in $L$

- $S_\infty$ contains all finite sets that do not contain 1
  - Since the number of such sets is infinite, their union is an infinite set
  - $\mathbb{Z} - \{1\}$ is not contained in $L$ (the only infinite set in $L$ is $\mathbb{Z}$)
  - $S_\infty$ does not have a lub in $L$

Hence $L$ is not complete
A Bounded Lattice Need Not be Complete (2)

- $S_f$ contains only a finite number of sets each of which does not contain 1
  - The union of all its member sets is a finite set excluding 1
  - Thus $S_f$ has a lub in $L$

- $S_\infty$ contains all finite sets that do not contain 1
  - Since the number of such sets is infinite, their union is an infinite set
  - $\mathbb{Z} - \{1\}$ is not contained in $L$ (the only infinite set in $L$ is $\mathbb{Z}$)
  - $S_\infty$ does not have a lub in $L$
  - Hence $L$ is not complete

- It may be tempting to assume that $\mathbb{Z}$ is the lub of $S_\infty$ because it is an upper bound of $S_\infty$ and no other upper bound of $S_\infty$ in the lattice is weaker $\mathbb{Z}$

- However, the join operation $\cup$ of $L$ does not compute $\mathbb{Z}$ as the lub of $S_\infty$ (because it must exclude 1)

- The join operation $\cup$ is inconsistent with the partial order $\supseteq$ of $L$. Hence we say that join does not exist for $S_\infty$
A Bounded Lattice Need Not be Complete (2)

- A bounded lattice \( L \) has a glb and lub of \( L \) in \( L \)
- A complete lattice \( L \) should have glb and lub of all subsets of \( L \)
- A lattice \( L \) should have glb and lub of all finite non-empty subsets of \( L \)
Ascending and Descending Chains

- Strictly ascending chain $x \sqsubset y \sqsubset \cdots \sqsubset z$
- Strictly descending chain $x \sqsupset y \sqsupset \cdots \sqsupset z$
- DCC: Descending Chain Condition
  All strictly descending chains are finite
- ACC: Ascending Chain Condition
  All strictly ascending chains are finite
Complete Lattice and Ascending and Descending Chains

- If $L$ satisfies acc and dcc, then
  - $L$ has finite height, and
  - $L$ is complete

- A complete lattice need not have finite height (i.e. strict chains may not be finite)

Example:
Lattice of integers under $\leq$ relation with $\infty$ as $\top$ and $-\infty$ as $\bot$
Variants of Lattices

Meet Semilattices
Variants of Lattices

Meet Semilattices

Meet Semilattices with $\bot$ element
Variants of Lattices

- Meet Semilattices
- Meet Semilattices satisfying dcc
- Meet Semilattices with ⊥ element

- dcc: descending chain condition
Variants of Lattices

- Meet Semilattices
- Meet Semilattices satisfying dcc
- Meet Semilattices with ⊥ element
- Join Semilattices

- dcc: descending chain condition
Variants of Lattices

- Meet Semilattices
- Meet Semilattices satisfying dcc
- Meet Semilattices with $\bot$ element

- dcc: descending chain condition

Lattices

Join Semilattices
Variants of Lattices

- Meet Semilattices
- Meet Semilattices satisfying dcc
- Meet Semilattices with ⊥ element

- Join Semilattices
- Join Semilattices with ⊤ element

- dcc: descending chain condition
Variants of Lattices

- Lattices
- Bounded lattices
- Meet Semilattices
- Join Semilattices
- Meet Semilattices with \( \perp \) element
- Meet Semilattices satisfying dcc
- Join Semilattices with \( \top \) element

- dcc: descending chain condition
Variants of Lattices

- Lattices
- Meet Semilattices
- Meet Semilattices satisfying dcc
- Meet Semilattices with $\perp$ element
- Bounded lattices
- Join Semilattices
- Join Semilattices satisfying acc
- Join Semilattices with $\top$ element

- dcc: descending chain condition
- acc: ascending chain condition

Aug 2017 IIT Bombay
Variants of Lattices

- Meet Semilattices
  - Meet Semilattices with $\bot$ element
- Meet Semilattices satisfying dcc
- Complete lattices
- Bounded lattices
- Join Semilattices
  - Join Semilattices satisfying acc
- Join Semilattices with $\top$ element

- dcc: descending chain condition
- acc: ascending chain condition
Variants of Lattices

- Lattices
- Meet Semilattices
  - with ⊥ element
  - satisfying dcc
- Join Semilattices
  - with ⊤ element
  - satisfying acc
- Complete lattices
- Complete lattices with dcc and acc

dcc: descending chain condition
acc: ascending chain condition
An Example of Lattices: Maintaining Like Counts on Cloud

Maintain $n$ servers and divide the traffic
- Each server maintains an $n$-tuple for each page
- Updates the counters for its own slot

<table>
<thead>
<tr>
<th></th>
<th>Server Blue</th>
<th>Server Red</th>
<th>Server Green</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 1</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>Page 2</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>Page 3</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
</tbody>
</table>
An Example of Lattices: Maintaining Like Counts on Cloud

Like for Page 1

Server Blue

Page 1: 1
Page 2: 0
Page 3: 0

Server Red

Page 1: 0
Page 2: 0
Page 3: 0

Server Green

Page 1: 0
Page 2: 0
Page 3: 0
An Example of Lattices: Maintaining Like Counts on Cloud

```
Like for Page 3
```

<table>
<thead>
<tr>
<th>Page 1</th>
<th>Page 2</th>
<th>Page 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server Blue</td>
<td>1 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>Server Red</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>Server Green</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
</tbody>
</table>

Aug 2017
An Example of Lattices: Maintaining Like Counts on Cloud

Like for Page 1

Server Blue  |  Server Red  |  Server Green
---|---|---
Page 1  |  Page 2  |  Page 3
1 0 0 0  |  0 0 0 0  |  0 0 0 0
0 1 0 0  |  0 0 0 0  |  0 0 0 0
0 0 0 0  |  0 0 0 0  |  0 0 0 1

Aug 2017
An Example of Lattices: Maintaining Like Counts on Cloud

Like for Page 3

Server Blue
Page 1: 1 0 0 0
Page 2: 0 0 0 0
Page 3: 1 0 0 0

Server Red
Page 1: 0 1 0 0
Page 2: 0 0 0 0
Page 3: 0 0 0 0

Server Green
Page 1: 0 0 0 0
Page 2: 0 0 0 0
Page 3: 0 0 0 1
An Example of Lattices: Maintaining Like Counts on Cloud

Like for Page 3

Server Blue
Page 1: 1 0 0 0
Page 2: 0 0 0 0
Page 3: 1 0 0 0

Server Red
Page 1: 0 1 0 0
Page 2: 0 0 0 0
Page 3: 0 1 0 0

Server Green
Page 1: 0 0 0 0
Page 2: 0 0 0 0
Page 3: 0 0 1 0
An Example of Lattices: Maintaining Like Counts on Cloud

Like for Page 3

Server Blue

Page 1: 1 0 0 0  
Page 2: 0 0 0 0  
Page 3: 2 0 0 0

Server Red

Page 1: 0 1 0 0  
Page 2: 0 0 0 0  
Page 3: 0 1 0 0

Server Green

Page 1: 0 0 0 0  
Page 2: 0 0 0 0  
Page 3: 0 0 1 0

Aug 2017 IIT Bombay
An Example of Lattices: Maintaining Like Counts on Cloud

Synchronize:
- Send the data to other servers
- Update the counters using point-wise max

| Page 1 | Server Blue | 1 | 0 | 0 |
|        | Server Red  | 0 | 1 | 0 |
|        | Server Green| 0 | 0 | 0 |
| Page 2 | Server Blue | 0 | 0 | 0 |
|        | Server Red  | 0 | 0 | 0 |
|        | Server Green| 0 | 0 | 0 |
| Page 3 | Server Blue | 2 | 0 | 0 |
|        | Server Red  | 0 | 1 | 0 |
|        | Server Green| 0 | 0 | 1 |
An Example of Lattices: Maintaining Like Counts on Cloud

Synchronize:
- Send the data to other servers
- Update the counters using point-wise max

Lattice of \( n \)-tuples using point-wise \( \geq \) as the partial order

\[
\langle x_1, x_2, \ldots, x_n \rangle \sqsubseteq \langle y_1, y_2, \ldots, y_n \rangle = (x_1 \geq y_1) \land (x_2 \geq y_2) \ldots \land (x_n \geq y_n)
\]

Tuples merged with max operation

\[
\langle x_1, x_2, \ldots, x_n \rangle \sqcap \langle y_1, y_2, \ldots, y_n \rangle = \langle \max(x_1, y_1), \max(x_2, y_2), \ldots, \max(x_n, y_n) \rangle
\]
### An Example of Lattices: Maintaining Like Counts on Cloud

Synchronize:
- Send the data to other servers
- Update the counters using point-wise max

<table>
<thead>
<tr>
<th></th>
<th>Server Blue</th>
<th>Server Red</th>
<th>Server Green</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Page 1</strong></td>
<td>1 0 0 0</td>
<td>0 1 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td><strong>Page 2</strong></td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td><strong>Page 3</strong></td>
<td>2 0 0 0</td>
<td>0 1 0 0</td>
<td>0 0 1 0</td>
</tr>
</tbody>
</table>
An Example of Lattices: Maintaining Like Counts on Cloud

Synchronize:
- Send the data to other servers
- Update the counters using point-wise max

<table>
<thead>
<tr>
<th></th>
<th>Server Blue</th>
<th>Server Red</th>
<th>Server Green</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 1</td>
<td>1 0 0</td>
<td>1 1 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>Page 2</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>Page 3</td>
<td>2 0 0</td>
<td>2 1 0</td>
<td>0 0 1</td>
</tr>
</tbody>
</table>
An Example of Lattices: Maintaining Like Counts on Cloud

Synchronize:
- Send the data to other servers
- Update the counters using point-wise max

<table>
<thead>
<tr>
<th></th>
<th>Server Blue</th>
<th>Server Red</th>
<th>Server Green</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 1</td>
<td>1 0 0 0</td>
<td>1 1 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>Page 2</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>Page 3</td>
<td>2 0 0 0</td>
<td>2 1 0 0</td>
<td>0 0 1 0</td>
</tr>
</tbody>
</table>
An Example of Lattices: Maintaining Like Counts on Cloud

Synchronize:
- Send the data to other servers
- Update the counters using point-wise max

<table>
<thead>
<tr>
<th></th>
<th>Server Blue</th>
<th>Server Red</th>
<th>Server Green</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 1</td>
<td>1 0 0 0</td>
<td>1 1 1 0</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>Page 2</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>Page 3</td>
<td>2 0 0 0</td>
<td>2 1 1 0</td>
<td>2 0 1 0</td>
</tr>
</tbody>
</table>

Aug 2017
An Example of Lattices: Maintaining Like Counts on Cloud

Synchronize:
- Send the data to other servers
- Update the counters using point-wise max

<table>
<thead>
<tr>
<th>Server Blue</th>
<th>Server Red</th>
<th>Server Green</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 1</td>
<td>1 0 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>Page 2</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>Page 3</td>
<td>2 0 0</td>
<td>2 1 0</td>
</tr>
</tbody>
</table>
An Example of Lattices: Maintaining Like Counts on Cloud

Synchronize:
- Send the data to other servers
- Update the counters using point-wise max

```
<table>
<thead>
<tr>
<th>Server Blue</th>
<th>Server Red</th>
<th>Server Green</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 (1)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Page 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Page 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1 (1)</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Aug 2017
An Example of Lattices: Maintaining Like Counts on Cloud

Synchronize:
- Send the data to other servers
- Update the counters using point-wise max

<table>
<thead>
<tr>
<th></th>
<th>Server Blue</th>
<th>Server Red</th>
<th>Server Green</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 1</td>
<td>1 1 0</td>
<td>1 1 0</td>
<td>1 0 0</td>
</tr>
<tr>
<td>Page 2</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>Page 3</td>
<td>2 1 0</td>
<td>2 1 0</td>
<td>2 0 1</td>
</tr>
</tbody>
</table>
An Example of Lattices: Maintaining Like Counts on Cloud

Synchronize:
- Send the data to other servers
- Update the counters using point-wise max

<table>
<thead>
<tr>
<th>Page 1</th>
<th>Server Blue</th>
<th>Server Red</th>
<th>Server Green</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 1 0</td>
<td>1 1 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>Page 2</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>Page 3</td>
<td>2 1 0</td>
<td>2 1 0</td>
<td>2 1 1</td>
</tr>
</tbody>
</table>
An Example of Lattices: Maintaining Like Counts on Cloud

Synchronize:
- Send the data to other servers
- Update the counters using point-wise max

<table>
<thead>
<tr>
<th></th>
<th>Server Blue</th>
<th>Server Red</th>
<th>Server Green</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Page 1</strong></td>
<td>1 1 0</td>
<td>1 1 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td><strong>Page 2</strong></td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td><strong>Page 3</strong></td>
<td>2 1 0</td>
<td>2 1 0</td>
<td>2 1 1</td>
</tr>
</tbody>
</table>
An Example of Lattices: Maintaining Like Counts on Cloud

Synchronize:
- Send the data to other servers
- Update the counters using point-wise max

<table>
<thead>
<tr>
<th>Page 1</th>
<th>Server Blue</th>
<th>Server Red</th>
<th>Server Green</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 1 0</td>
<td>1 1 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>Page 2</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>Page 3</td>
<td>2 1 1</td>
<td>2 1 0</td>
<td>2 1 1</td>
</tr>
</tbody>
</table>
Synchronize:
- Send the data to other servers
- Update the counters using point-wise max

<table>
<thead>
<tr>
<th>Page</th>
<th>Server Blue</th>
<th>Server Red</th>
<th>Server Green</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 1</td>
<td>1 1 0</td>
<td>1 1 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>Page 2</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>Page 3</td>
<td>2 1 1</td>
<td>2 1 0</td>
<td>2 1 1</td>
</tr>
</tbody>
</table>
Synchronize:
- Send the data to other servers
- Update the counters using point-wise max
An Example of Lattices: Maintaining Like Counts on Cloud

After synchronization, all servers have the same data Count for a page:
- Take sum of all counts at any server for the page

<table>
<thead>
<tr>
<th></th>
<th>Server Blue</th>
<th></th>
<th>Server Red</th>
<th></th>
<th>Server Green</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 1</td>
<td>1 1 0</td>
<td></td>
<td>1 1 0</td>
<td></td>
<td>1 1 0</td>
</tr>
<tr>
<td>Page 2</td>
<td>0 0 0</td>
<td></td>
<td>0 0 0</td>
<td></td>
<td>0 0 0</td>
</tr>
<tr>
<td>Page 3</td>
<td>2 1 1</td>
<td></td>
<td>2 1 1</td>
<td></td>
<td>2 1 1</td>
</tr>
</tbody>
</table>

Aug 2017
Constructing Lattices

- Powerset construction with subset or superset relation
- Products of lattices
  - Cartesian product
  - Lexicographic product
  - Interval product
  - Set of mappings
- Lattices on sequences using prefix or suffix as partial orders
Cartesian Product of Lattice

\[ \langle L_N, \sqsubseteq_N, \sqcap_N, \sqcup_N \rangle \times \langle L_A, \sqsubseteq_A, \sqcap_A, \sqcup_A \rangle = \langle L_{NA}, \sqsubseteq_{NA}, \sqcap_{NA}, \sqcup_{NA} \rangle \]
Cartesian Product of Lattice

\[ \langle L_N, \sqsubseteq_N, \sqcap_N, \sqcup_N \rangle \times \langle L_A, \sqsubseteq_A, \sqcap_A, \sqcup_A \rangle = \langle 1, a \rangle \approx \langle 2, a \rangle \approx \langle 3, a \rangle \approx \langle 4, a \rangle \]

Aug 2017
Cartesian Product of Lattice

\[ \langle L_N, \subseteq_N, \cap_N, \cup_N \rangle \times \langle L_A, \subseteq_A, \cap_A, \cup_A \rangle = \langle 1, a \rangle \langle 2, a \rangle \langle 3, a \rangle \langle 4, a \rangle = \langle 1, b \rangle \langle 2, b \rangle \langle 3, b \rangle \langle 4, b \rangle \]
Cartesian Product of Lattice

\[ \langle L_N, \sqsubseteq_N, \sqcap_N, \sqcup_N \rangle \times \langle L_A, \sqsubseteq_A, \sqcap_A, \sqcup_A \rangle = \langle 1, a \rangle \langle 2, a \rangle \langle 3, a \rangle \langle 4, a \rangle \langle 1, b \rangle \langle 2, b \rangle \langle 3, b \rangle \langle 4, b \rangle \]
Cartesian Product of Lattice

\[ \langle L_N, \sqsubseteq_N, \sqcap_N, \sqcup_N \rangle \times \langle L_A, \sqsubseteq_A, \sqcap_A, \sqcup_A \rangle = \langle \quad \rangle \]

Aug 2017 IIT Bombay
Cartesian Product of Lattice

\[
\langle L_N, \sqsubseteq_N, \sqcap_N, \sqcup_N \rangle \times \langle L_A, \sqsubseteq_A, \sqcap_A, \sqcup_A \rangle = \langle \begin{array}{c} 1, a \\langle 2, a \rangle \\langle 3, a \rangle \\langle 4, a \rangle \\langle 1, b \rangle \\langle 2, b \rangle \\langle 3, b \rangle \\langle 4, b \rangle \end{array} \rangle
\]
Cartesian Product of Lattice

\[ \langle L_N, \sqsubseteq_N, \sqcap_N, \sqcup_N \rangle \times \langle L_A, \sqsubseteq_A, \sqcap_A, \sqcup_A \rangle = \langle 1, a \rangle \langle 2, a \rangle \langle 3, a \rangle \langle 4, a \rangle \langle 1, b \rangle \langle 2, b \rangle \langle 3, b \rangle \langle 4, b \rangle \]
Cartesian Product of Lattice

\[ \langle L_N, \sqsubseteq_N, \sqcap_N, \sqcup_N \rangle \times \langle L_A, \sqsubseteq_A, \sqcap_A, \sqcup_A \rangle = \langle L_C, \sqsubseteq_C, \sqcap_C, \sqcup_C \rangle \]
Cartesian Product of Lattice

\[ (L_N, \sqsubseteq_N, \sqcap_N, \sqcup_N) \times (L_A, \sqsubseteq_A, \sqcap_A, \sqcup_A) = (L_C, \sqsubseteq_C, \sqcap_C, \sqcup_C) \]

\[ (x_1, y_1) \sqsubseteq_C (x_2, y_2) \iff x_1 \sqsubseteq_N x_2 \land y_1 \sqsubseteq_A y_2 \]

\[ (x_1, y_1) \sqcap_C (x_2, y_2) = (x_1 \sqcap_N x_2, y_1 \sqcap_A y_2) \]

\[ (x_1, y_1) \sqcup_C (x_2, y_2) = (x_1 \sqcup_N x_2, y_1 \sqcup_A y_2) \]
Example of Cartesian Product: Concept Lattices

- **Context of concepts.** A collection of objects and their attributes
- **Concepts.** Sets of attributes as exhibited by specific objects
  - A concept $C$ is a pair $(O, A)$ where
    - $O$ is a set of objects exhibiting attributes in the set $A$
    - Every object in $O$ has every attribute in $A$
- **Partial order.** $(O_2, A_2) \sqsubseteq (O_1, A_1) \iff O_2 \subseteq O_1$
  - Very few objects have all properties
  - Since $A$ is the set of attributes common to all objects in $O$,
    \[ O_2 \subseteq O_1 \Rightarrow A_2 \supseteq A_1 \]
As the number of chosen objects decreases, the number of common attributes increases
Example of Concept Lattice (1)

From *Introduction to Lattices and Order* by Davey and Priestley [2002]

<table>
<thead>
<tr>
<th></th>
<th>Size</th>
<th>Distance from Sun</th>
<th>Moon?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Small (ss)</td>
<td>Medium (sm)</td>
<td>Large (sl)</td>
</tr>
<tr>
<td>Mercury Me</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Venus V</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Earth E</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mars Ma</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jupiter J</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Saturn S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uranus U</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Neptune N</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Pluto P</td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
Example of Concept Lattice (2)

We write \((O, A)\) as \(\frac{O}{A}\)

\[
\begin{align*}
\{ r, v, e, m, j, s, u, n, p \} \\
\{ \} 
\end{align*}
\]
Example of Concept Lattice (2)

We write \((O, A)\) as \(\frac{O}{A}\)

\[
\begin{array}{c}
\{Me, V, E, Ma, J, S, U, N, P\} \\
\{\}
\end{array}
\]

\[
\begin{array}{c}
\{Me, V, E, Ma, P\} \\
\{ss\}
\end{array}
\]

\[
\begin{array}{c}
\{E, Ma, J, S, U, N, P\} \\
\{my\}
\end{array}
\]
We write \((O, A)\) as \(\frac{O}{A}\)

\[
\begin{align*}
\{\text{Me, V, E, Ma, J, S, U, N, P}\} \\
\{\text{Me, V, E, Ma, P}\} \\
\{\text{Me, V, E, Ma}\} \\
\{\text{Me, V, E}\} \\
\{\text{Me}\}
\end{align*}
\]

\[
\begin{align*}
\{\text{E, Ma, J, S, U, N, P}\} \\
\{\text{E, Ma}\} \\
\{\text{E}\} \\
\{\text{V}\} \\
\{\text{Me}\}
\end{align*}
\]

\[
\begin{align*}
\{\text{J, S, U, N, P}\} \\
\{\text{J, S}\} \\
\{\text{S}\} \\
\{\text{U}\} \\
\{\text{N}\} \\
\{\text{P}\}
\end{align*}
\]
Example of Concept Lattice (2)

We write \((O, A)\) as \(\frac{O}{A}\)

\[
\begin{array}{c}
\{Me, V, E, Ma, J, S, U, N, P\} \\
\{ss\} \\
\{Me, V, E, Ma\} \\
\{ss, dn\} \\
\{Me, V\} \\
\{ss, dn, mn\} \\
\{E, Ma\} \\
\{ss, my\} \\
\{E, Ma\} \\
\{ss, my\} \\
\{P\} \\
\{ss, df, my\} \\
\{J, S\} \\
\{sl, df, my\} \\
\{U, N\} \\
\{sm, df, my\} \\
\end{array}
\]
Example of Concept Lattice (2)

We write \((O, A)\) as \(\frac{O}{A}\)

\[
\begin{align*}
\{Me, V, E, Ma, J, S, U, N, P\} & \quad \{\} \\
\{Me, V, E, Ma, P\} & \quad \{ss\} \\
\{Me, V, E, Ma\} & \quad \{ss, dn\} \\
\{Me, V\} & \quad \{ss, dn, mn\} \\
\{E, Ma\} & \quad \{ss, my\} \\
\{E, Ma\} & \quad \{ss, dn, my\} \\
\{E, Ma\} & \quad \{ss, df, my\} \\
\{P\} & \quad \{ss, df, my\} \\
\{J, S\} & \quad \{sl, df, my\} \\
\{U, N\} & \quad \{sm, df, my\} \\
\{ss, sm, sl, dn, df, my, mn\} & \\
\end{align*}
\]
Variants of Products

In each case $L \subseteq L_1 \times L_2$

- **Cartesian Product**
  
  $$(x_1, x_2) \sqsubseteq (y_1, y_2) \text{ iff } x_1 \sqsubseteq_1 y_1 \land x_2 \sqsubseteq_2 y_2$$

- **Interval Product**

- **Lexicographic Product**

- **Set of mappings $L_1 \to L_2$**
Variants of Products

In each case $L \subseteq L_1 \times L_2$

- **Cartesian Product**

  $$(x_1, x_2) \sqsubseteq (y_1, y_2) \text{ iff } x_1 \sqsubseteq_1 y_1 \land x_2 \sqsubseteq_2 y_2$$

- **Interval Product**

  $$(x_1, x_2) \sqsubseteq (y_1, y_2) \text{ iff } x_1 \sqsubseteq_1 y_1 \land y_2 \sqsubseteq_2 x_2$$

- **Lexicographic Product**

- **Set of mappings $L_1 \rightarrow L_2$**
Variants of Products

In each case $L \subseteq L_1 \times L_2$

- **Cartesian Product**
  
  $(x_1, x_2) \sqsubseteq (y_1, y_2)$ iff $x_1 \sqsubseteq_1 y_1 \land x_2 \sqsubseteq_2 y_2$

- **Interval Product**
  
  $(x_1, x_2) \sqsubseteq (y_1, y_2)$ iff $x_1 \sqsubseteq_1 y_1 \land y_2 \sqsubseteq_2 x_2$

- **Lexicographic Product**
  
  $(x_1, x_2) \sqsubseteq (y_1, y_2)$ iff $(x_1 \sqsubseteq_1 y_1) \lor (x_1 = y_1 \land x_2 \sqsubseteq_2 y_2)$

- **Set of mappings** $L_1 \to L_2$
Variants of Products

In each case $L \subseteq L_1 \times L_2$

- **Cartesian Product**
  
  $$(x_1, x_2) \subseteq (y_1, y_2) \text{ iff } x_1 \sqsubseteq_1 y_1 \land x_2 \sqsubseteq_2 y_2$$

- **Interval Product**
  
  $$(x_1, x_2) \subseteq (y_1, y_2) \text{ iff } x_1 \sqsubseteq_1 y_1 \land y_2 \sqsubseteq_2 x_2$$

- **Lexicographic Product**
  
  $$(x_1, x_2) \subseteq (y_1, y_2) \text{ iff } (x_1 \sqsubseteq_1 y_1) \lor (x_1 = y_1 \land x_2 \sqsubseteq_2 y_2)$$

- **Set of mappings $L_1 \rightarrow L_2$**
  
  $$(x_1, x_2) \subseteq (y_1, y_2) \text{ iff } x_1 = y_1 \land x_2 \sqsubseteq_2 y_2$$
Part 5

Data Flow Values: Details
The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition
The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- Requirement: glb must exist for all non-empty finite subsets
The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- **Requirement:** glb must exist for all non-empty finite subsets
- **Corollary:** \( \bot \) must exist

What guarantees the presence of \( \bot \)?
The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- **Requirement**: glb must exist for all non-empty finite subsets
- **Corollary**: $\bot$ must exist
  
  What guarantees the presence of $\bot$?

- $\top$ may not exist. Can be added artificially
The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- **Requirement**: glb must exist for all non-empty finite subsets
- **Corollary**: \( \bot \) must exist
  
  What guarantees the presence of \( \bot \)?
  
  - Assume that two maximal descending chains terminate at two incomparable elements \( x_1 \) and \( x_2 \)

- \( \top \) may not exist. Can be added artificially
The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- **Requirement:** glb must exist for all non-empty finite subsets
- **Corollary:** \( \bot \) must exist

What guarantees the presence of \( \bot \)?

- Assume that two maximal descending chains terminate at two incomparable elements \( x_1 \) and \( x_2 \)
- Since this is a meet semilattice, glb of \( \{x_1, x_2\} \) must exist (say \( z \))

- \( \top \) may not exist. Can be added artificially
The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- **Requirement:** glb must exist for all non-empty finite subsets
- **Corollary:** \( \bot \) must exist

What guarantees the presence of \( \bot \)?

- Assume that two maximal descending chains terminate at two incomparable elements \( x_1 \) and \( x_2 \)
- Since this is a meet semilattice, glb of \( \{x_1, x_2\} \) must exist (say \( z \))
  \( \Rightarrow \) Neither of the chains is maximal
  Both of them can be extended to include \( z \)

- \( \top \) may not exist. Can be added artificially
The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- **Requirement:** glb must exist for all non-empty finite subsets
- **Corollary:** \( \bot \) must exist

What guarantees the presence of \( \bot \)?

- Assume that two maximal descending chains terminate at two incomparable elements \( x_1 \) and \( x_2 \)
- Since this is a meet semilattice, glb of \( \{ x_1, x_2 \} \) must exist (say \( z \))
  \( \Rightarrow \) Neither of the chains is maximal
  Both of them can be extended to include \( z \)
- Extending this argument to all strictly descending chains, it is easy to see that \( \bot \) must exist

- \( \top \) may not exist. Can be added artificially
The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- **Requirement:** glb must exist for all non-empty finite subsets
- **Corollary:** \( \bot \) must exist

What guarantees the presence of \( \bot \)?

- Assume that two maximal descending chains terminate at two incomparable elements \( x_1 \) and \( x_2 \)
- Since this is a meet semilattice, glb of \( \{x_1, x_2\} \) must exist (say \( z \))
  \(\Rightarrow\) Neither of the chains is maximal
  Both of them can be extended to include \( z \)
- Extending this argument to all strictly descending chains, it is easy to see that \( \bot \) must exist

- \( \top \) may not exist. Can be added artificially
  - lub of arbitrary elements may not exist
The Set of Data Flow Values For Available Expressions Analysis

- The powerset of the universal set of expressions
- Partial order is the subset relation

\[
\begin{align*}
\{e_1, e_2, e_3\} & \quad \{e_1, e_2\} \quad \{e_1, e_3\} \quad \{e_2, e_3\} \\
\{e_1\} & \quad \{e_2\} \quad \{e_3\} \\
\emptyset &
\end{align*}
\]

Set View of the Lattice
The Set of Data Flow Values For Available Expressions Analysis

- The powerset of the universal set of expressions
- Partial order is the subset relation

Set View of the Lattice
The Set of Data Flow Values For Available Expressions Analysis

- The powerset of the universal set of expressions
- Partial order is the subset relation

Set View of the Lattice

Bit Vector View
The Concept of Approximation

- $x$ approximates $y$ \textit{iff}
  
  $x$ can be used in place of $y$ without causing any problems

- Validity of approximation is context specific
  
  $x$ may be approximated by $y$ in one context and by $z$ in another
The Concept of Approximation

- $x$ approximates $y$ iff $x$ can be used in place of $y$ without causing any problems.

- Validity of approximation is context specific. $x$ may be approximated by $y$ in one context and by $z$ in another.

  - Approximating Money
    - Earnings: Rs. 1050 can be safely approximated by Rs. 1000
    - Expenses: Rs. 1050 can be safely approximated by Rs. 1100
The Concept of Approximation

• $x$ approximates $y$ iff

  $x$ can be used in place of $y$ without causing any problems

• Validity of approximation is context specific

  $x$ may be approximated by $y$ in one context and by $z$ in another

  ▶ Approximating Money

    Earnings: Rs. 1050 can be safely approximated by Rs. 1000
    Expenses: Rs. 1050 can be safely approximated by Rs. 1100

  ▶ Approximating Time

    Travel time: 2 hours required can be safely approximated by 3 hours
    Study time: 3 available days can be safely assumed to be only 2 days
Two Important Objectives in Data Flow Analysis

- The discovered data flow information should be
  - **Exhaustive.** No optimization opportunity should be missed
  - **Safe.** Optimizations which do not preserve semantics should not be enabled
Two Important Objectives in Data Flow Analysis

- The discovered data flow information should be
  - **Exhaustive.** No optimization opportunity should be missed
  - **Safe.** Optimizations which do not preserve semantics should not be enabled
- Conservative approximations of these objectives are allowed
Two Important Objectives in Data Flow Analysis

- The discovered data flow information should be
  - *Exhaustive*. No optimization opportunity should be missed
  - *Safe*. Optimizations which do not preserve semantics should not be enabled
- Conservative approximations of these objectives are allowed
- The intended use of data flow information (≡ context) determines validity of approximations
## Context Determines the Validity of Approximations

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Application</th>
<th>Safe Approximation</th>
<th>Exhaustive Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Will not do incorrect optimization</td>
<td>Will not miss any correct optimization</td>
<td>May prohibit correct optimization</td>
<td>May enable incorrect optimization</td>
</tr>
</tbody>
</table>
Context Determines the Validity of Approximations

Will not do incorrect optimization
May prohibit correct optimization

Will not miss any correct optimization
May enable incorrect optimization

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Application</th>
<th>Safe Approximation</th>
<th>Exhaustive Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Live variables</td>
<td>Dead code elimination</td>
<td>A dead variable is considered live</td>
<td>A live variable is considered dead</td>
</tr>
</tbody>
</table>
Context Determines the Validity of Approximations

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Application</th>
<th>Safe Approximation</th>
<th>Exhaustive Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Live variables</td>
<td>Dead code elimination</td>
<td>A dead variable is considered live</td>
<td>A live variable is considered dead</td>
</tr>
<tr>
<td>Available expressions</td>
<td>Common subexpression elimination</td>
<td>An available expression is considered non-available</td>
<td>A non-available expression is considered available</td>
</tr>
</tbody>
</table>
## Context Determines the Validity of Approximations

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Application</th>
<th>Safe Approximation</th>
<th>Exhaustive Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Live variables</td>
<td>Dead code elimination</td>
<td>A dead variable is considered live</td>
<td>A live variable is considered dead</td>
</tr>
<tr>
<td>Available expressions</td>
<td>Common subexpression elimination</td>
<td>An available expression is considered non-available</td>
<td>A non-available expression is considered available</td>
</tr>
</tbody>
</table>

- **Spurious Inclusion**
- **Spurious Exclusion**
Partial Order Captures Approximation

- \( \sqsubseteq \) captures valid approximations for safety

\[ x \sqsubseteq y \Rightarrow x \text{ is weaker than } y \]

- The data flow information represented by \( x \) can be safely used in place of the data flow information represented by \( y \)
- It may be imprecise, though
Partial Order Captures Approximation

- $\sqsubseteq$ captures valid approximations for **safety**
  - $x \sqsubseteq y \implies x$ is *weaker than* $y$
    - The data flow information represented by $x$ can be safely used in place of the data flow information represented by $y$
    - It may be imprecise, though

- $\sqsupseteq$ captures valid approximations for **exhaustiveness**
  - $x \sqsupseteq y \implies x$ is *stronger than* $y$
    - The data flow information represented by $x$ contains every value contained in the data flow information represented by $y$
    - $x \sqcap y$ will not compute a value weaker than $y$
    - It may be unsafe, though
Partial Order Captures Approximation

- \( \sqsubseteq \) captures valid approximations for safety

\[ x \sqsubseteq y \Rightarrow x \text{ is weaker than } y \]

- The data flow information represented by \( x \) can be safely used in place of the data flow information represented by \( y \)
- It may be imprecise, though

- \( \sqsupseteq \) captures valid approximations for exhaustiveness

\[ x \sqsupseteq y \Rightarrow x \text{ is stronger than } y \]

- The data flow information represented by \( x \) contains every value contained in the data flow information represented by \( y \)
- \( x \sqcap y \) will not compute a value weaker than \( y \)
- It may be unsafe, though

We want most exhaustive information which is also safe
Most Approximate Values in a Complete Lattice

- **Top.** $\forall x \in L, \ x \sqsubseteq \top$ Exhaustive approximation of all values

- **Bottom.** $\forall x \in L, \bot \sqsubseteq x$ Safe approximation of all values
Most Approximate Values in a Complete Lattice

- **Top.** $\forall x \in L, x \sqsubseteq \top$ Exhaustive approximation of all values
  - Using $\top$ in place of any data flow value will never miss out (or rule out) any possible value

- **Bottom.** $\forall x \in L, \bot \sqsubseteq x$ Safe approximation of all values
Most Approximate Values in a Complete Lattice

- **Top.** \( \forall x \in L, \ x \sqsubseteq \top \)** Exhaustive approximation of all values
  - Using \( \top \) in place of any data flow value will never miss out (or rule out) any possible value
  - The consequences may be semantically *unsafe*, or *incorrect*

- **Bottom.** \( \forall x \in L, \ \bot \sqsubseteq x \)** Safe approximation of all values
Most Approximate Values in a Complete Lattice

- **Top.** \( \forall x \in L, \ x \sqsubseteq \top \)** Exhaustive approximation of all values
  
  ▶ Using \( \top \) in place of any data flow value will never miss out (or rule out) any possible value
  
  ▶ The consequences may be semantically *unsafe*, or *incorrect*

- **Bottom.** \( \forall x \in L, \ \bot \sqsubseteq x \)** Safe approximation of all values
  
  ▶ Using \( \bot \) in place of any data flow value will never be *unsafe*, or *incorrect*
Most Approximate Values in a Complete Lattice

- **Top.** $\forall x \in L, \ x \sqsubseteq \top$ Exhaustive approximation of all values
  - Using $\top$ in place of any data flow value will never miss out (or rule out) any possible value
  - The consequences may be semantically *unsafe*, or *incorrect*

- **Bottom.** $\forall x \in L, \bot \sqsubseteq x$ Safe approximation of all values
  - Using $\bot$ in place of any data flow value will never be *unsafe*, or *incorrect*
  - The consequences may be *undefined* or *useless* because this replacement might miss out valid values
Most Approximate Values in a Complete Lattice

- **Top.** $\forall x \in L, \ x \sqsubseteq \top$ Exhaustive approximation of all values
  - Using $\top$ in place of any data flow value will never miss out (or rule out) any possible value
  - The consequences may be semantically *unsafe*, or *incorrect*

- **Bottom.** $\forall x \in L, \bot \sqsubseteq x$ Safe approximation of all values
  - Using $\bot$ in place of any data flow value will never be *unsafe*, or *incorrect*
  - The consequences may be *undefined* or *useless* because this replacement might miss out valid values

*Appropriate orientation chosen by design*
Setting Up Lattices

Available Expressions Analysis

Live Variables Analysis

\[ \{ e_1, e_2, e_3 \} \]
\[ \{ e_1, e_2 \} \]
\[ \{ e_1, e_3 \} \]
\[ \{ e_2, e_3 \} \]
\[ \{ e_1 \} \]
\[ \{ e_2 \} \]
\[ \{ e_3 \} \]

\[ \{ v_1 \} \]
\[ \{ v_2 \} \]
\[ \{ v_3 \} \]
\[ \{ v_1, v_2 \} \]
\[ \{ v_1, v_3 \} \]
\[ \{ v_2, v_3 \} \]
\[ \{ v_1, v_2, v_3 \} \]

\( \sqsubseteq \; is \; \subseteq \)

\( \sqcap \; is \; \cap \)

\( \sqcup \; is \; \cup \)
Partial Order Relation

- Reflexive: \( x \sqsubseteq x \)
- Transitive: \( x \sqsubseteq y, y \sqsubseteq z \Rightarrow x \sqsubseteq z \)
- Antisymmetric: \( x \sqsubseteq y, y \sqsubseteq x \Leftrightarrow x = y \)
## Partial Order Relation

<table>
<thead>
<tr>
<th>Type</th>
<th>Condition</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reflexive</td>
<td>$x \sqsubseteq x$</td>
<td>$x$ can be safely used in place of $x$</td>
</tr>
<tr>
<td>Transitive</td>
<td>$x \sqsubseteq y$, $y \sqsubseteq z \implies x \sqsubseteq z$</td>
<td>If $x$ can be safely used in place of $y$ and $y$ can be safely used in place of $z$, then $x$ can be safely used in place of $z$</td>
</tr>
<tr>
<td>Antisymmetric</td>
<td>$x \sqsubseteq y$, $y \sqsubseteq x \iff x = y$</td>
<td>If $x$ can be safely used in place of $y$ and $y$ can be safely used in place of $x$, then $x$ must be same as $y$</td>
</tr>
</tbody>
</table>
Merging Information

- \( x \sqcap y \) computes the *greatest lower bound* of \( x \) and \( y \) i.e. largest \( z \) such that \( z \sqsubseteq x \) and \( z \sqsubseteq y \)

The largest safe approximation of combining data flow information \( x \) and \( y \)
Merging Information

- \( x \sqcap y \) computes the *greatest lower bound* of \( x \) and \( y \) i.e. largest \( z \) such that \( z \sqsubseteq x \) and \( z \sqsubseteq y \)

The largest safe approximation of combining data flow information \( x \) and \( y \)

- Commutative  \( x \sqcap y = y \sqcap x \)

- Associative  \( x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z \)

- Idempotent  \( x \sqcap x = x \)
Merging Information

- $x \sqcap y$ computes the *greatest lower bound* of $x$ and $y$ i.e. the largest $z$ such that $z \subseteq x$ and $z \subseteq y$

The largest safe approximation of combining data flow information $x$ and $y$

- **Commutative** $x \sqcap y = y \sqcap x$  
  The order in which the data flow information is merged, does not matter

- **Associative** $x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z$  
  Allow $n$-ary merging without any restriction on the order

- **Idempotent** $x \sqcap x = x$  
  No loss of information if $x$ is merged with itself

Aug 2017
Merging Information

• $x \sqcap y$ computes the greatest lower bound of $x$ and $y$ i.e. largest $z$ such that $z \sqsubseteq x$ and $z \sqsubseteq y$

The largest safe approximation of combining data flow information $x$ and $y$

• Commutative $x \sqcap y = y \sqcap x$ The order in which the data flow information is merged, does not matter

• Associative $x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z$ Allow n-ary merging without any restriction on the order

• Idempotent $x \sqcap x = x$ No loss of information if $x$ is merged with itself

• $\top$ is the identity of $\sqcap$

  ▶ Presence of loops $\Rightarrow$ self dependence of data flow information
  ▶ Using $\top$ as the initial value ensure exhaustiveness
More on Lattices in Data Flow Analysis

\[ L = \text{Lattice for all expressions} \]

\[ \hat{L} = \text{Lattice for a single expression} \]

(111)

(110) \rightarrow (101) \rightarrow (011)

(100) \rightarrow (010) \rightarrow (001)

(000)

(Expressions \( e \) is available)

1 or \{e\}

0 or \( \emptyset \)

(Expressions \( e \) is not available)
More on Lattices in Data Flow Analysis

\[ L = \text{Lattice for all expressions} \]

\[ \widehat{L} = \text{Lattice for a single expression} \]

Cartesian products if sets are used, vectors (or tuples) if bit are used

- \( L = \widehat{L} \times \widehat{L} \times \widehat{L} \) and \( x = (\widehat{x}_1, \widehat{x}_2, \widehat{x}_3) \in L \) where \( \widehat{x}_i \in \widehat{L} \)

- \( \sqsubseteq = \widehat{\sqsubseteq} \times \widehat{\sqsubseteq} \times \widehat{\sqsubseteq} \) and \( \sqcap = \widehat{\sqcap} \times \widehat{\sqcap} \times \widehat{\sqcap} \)

- \( \top = \widehat{\top} \times \widehat{\top} \times \widehat{\top} \) and \( \bot = \widehat{\bot} \times \widehat{\bot} \times \widehat{\bot} \)

(Expression e is available)

- 1 or \{e\}

(Expression e is not available)

- 0 or \(\emptyset\)
Component Lattice for Data Flow Information Represented By Bit Vectors

\[ \begin{align*}
(\hat{\top}) & \quad (\hat{\top}) \\
1 & \quad 0 \\
\quad & \quad \\
0 & \quad 1 \\
(\hat{\bot}) & \quad (\hat{\bot})
\end{align*} \]

\( \sqcap \) is \( \cap \) or Boolean AND \\
\( \sqcup \) is \( \cup \) or Boolean OR
Component Lattice for Integer Constant Propagation

- Overall lattice $L$ is the set of mappings from variables to $\hat{L}$
- $\sqsubseteq$ and $\sqsupseteq$ get defined by $\sqsubseteq$ and $\sqsupseteq$

<table>
<thead>
<tr>
<th>$\hat{\sqsubseteq}$</th>
<th>$\langle a, ud \rangle$</th>
<th>$\langle a, nc \rangle$</th>
<th>$\langle a, c_1 \rangle$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle a, ud \rangle$</td>
<td>$\langle a, ud \rangle$</td>
<td>$\langle a, nc \rangle$</td>
<td>$\langle a, c_1 \rangle$</td>
</tr>
<tr>
<td>$\langle a, nc \rangle$</td>
<td>$\langle a, nc \rangle$</td>
<td>$\langle a, nc \rangle$</td>
<td>$\langle a, nc \rangle$</td>
</tr>
<tr>
<td>$\langle a, c_2 \rangle$</td>
<td>$\langle a, c_2 \rangle$</td>
<td>$\langle a, nc \rangle$</td>
<td>If $c_1 = c_2$ then $\langle a, c_1 \rangle$ else $\langle a, nc \rangle$</td>
</tr>
</tbody>
</table>
Component Lattice for May Points-To Analysis

- Relation between pointer variables and locations in the memory
Component Lattice for May Points-To Analysis

- Relation between pointer variables and locations in the memory
- Assuming three locations $l_1$, $l_2$, and $l_3$, the component lattice for pointer $p$ is

```
(⊤)
∅

{p → l_1}  {p → l_2}  {p → l_3}

{p → l_1, p → l_2}  {p → l_1, p → l_3}  {p → l_2, p → l_3}

{p → l_1, p → l_2, p → l_3}
```

(⊥)
Component Lattice for May Points-To Analysis

- Relation between pointer variables and locations in the memory
- Assuming three locations $l_1$, $l_2$, and $l_3$, the component lattice for pointer $p$ is

$$
\begin{align*}
\emptyset & & & \emptyset \\
\{p \rightarrow l_1\} & & \{p \rightarrow l_2\} & \{p \rightarrow l_3\} \\
\{p \rightarrow l_1, p \rightarrow l_2\} & & \{p \rightarrow l_1, p \rightarrow l_3\} & \{p \rightarrow l_2, p \rightarrow l_3\} \\
\{p \rightarrow l_1, p \rightarrow l_2, p \rightarrow l_2\} & & & (\hat{\perp})
\end{align*}
$$

Alternatively,

$$
\begin{align*}
\emptyset & & & \emptyset \\
\{l_1\} & & \{l_2\} & \{l_3\} \\
\{l_1, l_2\} & & \{l_1, l_3\} & \{l_2, l_3\} \\
\{l_1, l_2, l_2\} & & & (\hat{\perp})
\end{align*}
$$
Component Lattice for Must Points-To Analysis

- A pointer can point to at most one location

```
(⊤)
/
\ undef
/   |
|   |
|   |
|   |
\ p→l₁  p→l₂  p→l₃
|
|
none
|
| (⊥)

Alternatively,

(⊤)
/
\ undef
/   |
|   |
|   |
\ l₁  l₂  l₃
|
|
none
|
| (⊥)
```
Combined Total and Partial Availability Analysis

- Two bits per expression rather than one. Can be implemented using AND (as below) or using OR (reversed lattice)

```
unknown
(Bits 11)
```

```
must-be-available
(is-not-available
(Bits 10)
(Bits 01)

may-be-available
(Bits 00)
```

Can also be implemented as a product of 1-0 and 0-1 lattice with AND for the first bit and OR for the second bit

- What approximation of safety does this lattice capture?
  Uncertain information (= no optimization) is guaranteed to be safe
**General Lattice for May-Must Analysis**

Interpreting data flow values

- *Unknown*. Nothing is known as yet
- *No*. Information does not hold along any path
- *Must*. Information must hold along all paths
- *May*. Information may hold along some path

Possible Applications

- Pointer Analysis: No need of separate of *May* and *Must* analyses
  eg. \((p \rightarrow l, \text{May})\), \((p \rightarrow l, \text{Must})\), \((p \rightarrow l, \text{No})\), or \((p \rightarrow l, \text{Unknown})\)
- Type Inferencing for Dynamically Checked Languages
Part 6

Flow Functions
Flow Functions: An Outline of Our Discussion

- Defining flow functions
- Properties of flow functions
  (Some properties discussed in the context of solutions of data flow analysis)
The Set of Flow Functions

- $F$ is the set of functions $f : L \rightarrow L$ such that
  - $F$ contains an identity function
    To model “empty” statements, i.e. statements which do not influence the data flow information
  - $F$ is closed under composition
    Cumulative effect of statements should generate data flow information from the same set
  - For every $x \in L$, there must be a finite set of flow functions $\{f_1, f_2, \ldots f_m\} \subseteq F$ such that
    $$x = \prod_{1 \leq i \leq m} f_i(BI)$$

- Properties of $f$
  - Monotonicity and Distributivity
  - Loop Closure Boundedness and Separability
Flow Functions in Bit Vector Data Flow Frameworks

- Bit Vector Frameworks: Available Expressions Analysis, Reaching Definitions Analysis Live variable Analysis, Anticipable Expressions Analysis, Partial Redundancy Elimination etc
  - All functions can be defined in terms of constant Gen and Kill

\[ f(x) = \text{Gen} \cup (x - \text{Kill}) \]

- Lattices are powersets with partial orders as \( \subseteq \) or \( \supseteq \) relations
- Information is merged using \( \cap \) or \( \cup \)
Flow Functions in Bit Vector Data Flow Frameworks

- Bit Vector Frameworks: Available Expressions Analysis, Reaching Definitions Analysis Live variable Analysis, Anticipable Expressions Analysis, Partial Redundancy Elimination etc
  - All functions can be defined in terms of constant Gen and Kill
    \[ f(x) = \text{Gen} \cup (x - \text{Kill}) \]
  - Lattices are powersets with partial orders as $\subseteq$ or $\supseteq$ relations
  - Information is merged using $\cap$ or $\cup$
- Flow functions in Strong Liveness Analysis, Pointer Analyses, Constant Propagation, Possibly Uninitialized Variables cannot be expressed using constant Gen and Kill
  - Local context alone is not sufficient to describe the effect of statements fully
Monotonicity of Flow Functions

- Partial order is preserved: If $x$ can be safely used in place of $y$ then $f(x)$ can be safely used in place of $f(y)$

\[ f(x) \leq f(y) \]
Monotonicity of Flow Functions

- Partial order is preserved: If $x$ can be safely used in place of $y$ then $f(x)$ can be safely used in place of $f(y)$.
Monotonicity of Flow Functions

- Partial order is preserved: If $x$ can be safely used in place of $y$ then $f(x)$ can be safely used in place of $f(y)$

$$\forall x, y \in L, x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$$
Monotonicity of Flow Functions

- Partial order is preserved: If $x$ can be safely used in place of $y$ then $f(x)$ can be safely used in place of $f(y)$

\[ \forall x, y \in L, x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y) \]

- Alternative definition

\[ \forall x, y \in L, f(x \sqcap y) \sqsubseteq f(x) \sqcap f(y) \]
Monotonicity of Flow Functions

- Partial order is preserved: If $x$ can be safely used in place of $y$ then $f(x)$ can be safely used in place of $f(y)$

\[ \forall x, y \in L, x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y) \]

- Alternative definition

\[ \forall x, y \in L, f(x \sqcap y) \sqsubseteq f(x) \sqcap f(y) \]

- Merging at intermediate points in shared segments of paths is safe (However, it may lead to imprecision)
Distributivity of Flow Functions

- Merging distributes over function application

\[ f(x) \sqcap f(y) \]
Distributivity of Flow Functions

- Merging distributes over function application

\[ f(x \sqcap y) = f(x) \sqcap f(y) \]
Distributivity of Flow Functions

- Merging distributes over function application

\[ \forall x, y \in L, f(x \cap y) = f(x) \cap f(y) \]
Distributivity of Flow Functions

- Merging distributes over function application

\[ \forall x, y \in L, f(x \sqcap y) = f(x) \sqcap f(y) \]

- Merging at intermediate points in shared segments of paths does not lead to imprecision
Monotonicity and Distributivity
Monotonicity and Distributivity

\[
\top \\
L \\
\bot
\]
Monotonicity and Distributivity
Monotonicity and Distributivity
Monotonicity and Distributivity
Monotonicity and Distributivity

Distributive and hence monotonic

\[ L \]
Monotonicity and Distributivity

Monotonic but not distributive

$L$ $L$
Distributivity of Bit Vector Frameworks

\[ f(x) = \text{Gen} \cup (x - \text{Kill}) \]
\[ f(y) = \text{Gen} \cup (y - \text{Kill}) \]

\[ f(x \cup y) = \text{Gen} \cup ((x \cup y) - \text{Kill}) \]
\[ = \text{Gen} \cup ((x - \text{Kill}) \cup (y - \text{Kill})) \]
\[ = (\text{Gen} \cup (x - \text{Kill}) \cup \text{Gen} \cup (y - \text{Kill})) \]
\[ = f(x) \cup f(y) \]

\[ f(x \cap y) = \text{Gen} \cup ((x \cap y) - \text{Kill}) \]
\[ = \text{Gen} \cup ((x - \text{Kill}) \cap (y - \text{Kill})) \]
\[ = (\text{Gen} \cup (x - \text{Kill}) \cap \text{Gen} \cup (y - \text{Kill})) \]
\[ = f(x) \cap f(y) \]
Non-Distributivity of Constant Propagation

\[
\begin{align*}
\text{n}_1 & : a = 1 \\
& b = 2 \\
& c = a + b \\
\text{n}_2 & : c = a + b \\
& d = a \times b \\
\text{n}_3 & : d = c - 1 \\
& a = 2 \\
& b = 1 \\
& c = a + b
\end{align*}
\]
Non-Distributivity of Constant Propagation

- $x = \langle 1, 2, 3, ud \rangle$ (Along $Out_{n_1} \to In_{n_2}$)

Diagram:

- $n_1$
  - $a = 1$
  - $b = 2$
  - $c = a + b$
  - $a = 1, b = 2$

- $n_2$
  - $c = a + b$
  - $d = a \times b$

- $n_3$
  - $d = c - 1$
  - $a = 2$
  - $b = 1$
  - $c = a + b$
Non-Distributivity of Constant Propagation

- $x = \langle 1, 2, 3, ud \rangle$ (Along $Out_{n_1} \rightarrow Ln_{n_2}$)
- $y = \langle 2, 1, 3, 2 \rangle$ (Along $Out_{n_3} \rightarrow Ln_{n_2}$)
Non-Distributivity of Constant Propagation

- $x = \langle 1, 2, 3, ud \rangle$ (Along $Out_{n_1} \rightarrow In_{n_2}$)
- $y = \langle 2, 1, 3, 2 \rangle$ (Along $Out_{n_3} \rightarrow In_{n_2}$)
- Function application for block $n_2$ before merging

\[
f(x) \sqcap f(y) = f(\langle 1, 2, 3, ud \rangle) \sqcap f(\langle 2, 1, 3, 2 \rangle)
= \langle 1, 2, 3, 2 \rangle \sqcap \langle 2, 1, 3, 2 \rangle
= \langle \perp, \perp, 3, 2 \rangle
\]
Non-Distributivity of Constant Propagation

- \( x = \langle 1, 2, 3, ud \rangle \) (Along \( Out_{n_1} \rightarrow Ln_{n_2} \))
- \( y = \langle 2, 1, 3, 2 \rangle \) (Along \( Out_{n_3} \rightarrow Ln_{n_2} \))
- Function application for block \( n_2 \) before merging
  \[
  f(x) \sqcap f(y) = f(\langle 1, 2, 3, ud \rangle) \sqcap f(\langle 2, 1, 3, 2 \rangle) \\
  = \langle 1, 2, 3, 2 \rangle \sqcap \langle 2, 1, 3, 2 \rangle \\
  = \langle \bot, \bot, 3, 2 \rangle
  \]
- Function application for block \( n_2 \) after merging
  \[
  f(x \sqcap y) = f(\langle 1, 2, 3, ud \rangle \sqcap \langle 2, 1, 3, 2 \rangle) \\
  = f(\langle \bot, \bot, 3, 2 \rangle) \\
  = \langle \bot, \bot, \bot, \bot \rangle
  \]
Non-Distributivity of Constant Propagation

- $x = \langle 1, 2, 3, ud \rangle$ (Along $Out_{n_1} \rightarrow In_{n_2}$)
- $y = \langle 2, 1, 3, 2 \rangle$ (Along $Out_{n_3} \rightarrow In_{n_2}$)
- Function application for block $n_2$ before merging
  
  $$f(x) \sqcap f(y) = f(\langle 1, 2, 3, ud \rangle) \sqcap f(\langle 2, 1, 3, 2 \rangle)$$
  $$= \langle 1, 2, 3, 2 \rangle \sqcap \langle 2, 1, 3, 2 \rangle$$
  $$= \langle \top, \top, 3, 2 \rangle$$

- Function application for block $n_2$ after merging
  
  $$f(x \sqcap y) = f(\langle 1, 2, 3, ud \rangle \sqcap \langle 2, 1, 3, 2 \rangle)$$
  $$= f(\langle \top, \top, 3, 2 \rangle)$$
  $$= \langle \top, \top, \top, \top \rangle$$

- $f(x \sqcap y) \sqsubseteq f(x) \sqcap f(y)$
Why is Constant Propagation Non-Distributive?

\[
a = 1 \\
b = 2
\]

\[
a = 2 \\
b = 1
\]

\[
c = a + b
\]
Why is Constant Propagation Non-Distributive?

Possible combinations due to merging

\[
\begin{align*}
  a &= 1 & a &= 2 & b &= 1 & b &= 2 \\
  b &= 2 & b &= 1 \\
  c &= a + b \\
\end{align*}
\]
Why is Constant Propagation Non-Distributive?

Possible combinations due to merging

\[ a = 1 \quad a = 2 \quad b = 1 \quad b = 2 \]

\[ c = a + b = 3 \]

- Correct combination
Why is Constant Propagation Non-Distributive?

Possible combinations due to merging

- $c = a + b = 3$

- Correct combination

$a = 1$

$b = 2$

$a = 2$

$b = 1$

$c = a + b = 3$
Why is Constant Propagation Non-Distributive?

Possible combinations due to merging

\[
\begin{align*}
a &= 1 \\
b &= 2 \\
c &= a + b = 2
\end{align*}
\]

\[
\begin{align*}
a &= 2 \\
b &= 1 \\
c &= a + b = 2
\end{align*}
\]

- Wrong combination
- Mutually exclusive information
- No execution path along which this information holds
Why is Constant Propagation Non-Distributive?

Possible combinations due to merging:

- Wrong combination
- Mutually exclusive information
- No execution path along which this information holds
Part 7

Solutions of Data Flow Analysis
Solutions of Data Flow Analysis: An Outline of Our Discussion

- MoP and MFP assignments and their relationship
- Existence of MoP assignment
  - Boundedness of flow functions
- Existence and Computability of MFP assignment
  - Flow functions Vs. function computed by data flow equations
- Safety of MFP solution
**Solutions of Data Flow Analysis**

- An assignment $A$ associates data flow values with program points $A \sqsubseteq B$ if for all program points $p$, $A(p) \sqsubseteq B(p)$

- Performing data flow analysis

  Given

  - A set of flow functions, a lattice, and merge operation
  - A program flow graph with a mapping from nodes to flow functions

  Find out

  - An assignment $A$ which is as exhaustive as possible and is safe
An Example For Available Expressions Analysis

Program

<table>
<thead>
<tr>
<th>Some Assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td><strong>In</strong>&lt;sub&gt;1&lt;/sub&gt;</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td><strong>Out</strong>&lt;sub&gt;1&lt;/sub&gt;</td>
</tr>
<tr>
<td><strong>In</strong>&lt;sub&gt;2&lt;/sub&gt;</td>
</tr>
<tr>
<td><strong>Out</strong>&lt;sub&gt;2&lt;/sub&gt;</td>
</tr>
</tbody>
</table>
An Example For Available Expressions Analysis

Program

```
a * b
b * c
```

Some Assignments

<table>
<thead>
<tr>
<th></th>
<th>( A_0 )</th>
<th>( A_1 )</th>
<th>( A_2 )</th>
<th>( A_3 )</th>
<th>( A_4 )</th>
<th>( A_5 )</th>
<th>( A_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( In_1 )</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>( Out_1 )</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>( In_2 )</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>( Out_2 )</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>10</td>
</tr>
</tbody>
</table>

Lattice \( L \) of data flow values at a node

```
11

10 01

00
```
An Example For Available Expressions Analysis

Program

1

\[
\begin{align*}
a \times b \\
b \times c
\end{align*}
\]

Lattice \( L \) of data flow values at a node

\[
\begin{align*}
11 \\
10 & 01 \\
00
\end{align*}
\]

Some Assignments

<table>
<thead>
<tr>
<th></th>
<th>( A_0 )</th>
<th>( A_1 )</th>
<th>( A_2 )</th>
<th>( A_3 )</th>
<th>( A_4 )</th>
<th>( A_5 )</th>
<th>( A_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{In}_1 )</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>( \text{Out}_1 )</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>( \text{In}_2 )</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>( \text{Out}_2 )</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>10</td>
</tr>
</tbody>
</table>

Lattice \( L \times L \times L \times L \) for data flow values at all nodes
Meet Over Paths (MoP) Assignment

- The largest safe approximation of the information reaching a program point along all information flow paths

\[
\text{MoP}(p) = \bigcap_{\rho \in \text{Paths}(p)} f_{\rho}(BI)
\]

- \(f_{\rho}\) represents the compositions of flow functions along \(\rho\)
- \(BI\) refers to the relevant information from the calling context
- All execution paths are considered potentially executable by ignoring the results of conditionals
Meet Over Paths (MoP) Assignment

- The largest safe approximation of the information reaching a program point along all information flow paths

\[
\text{MoP}(p) = \bigcap_{\rho \in \text{Paths}(p)} f_\rho(BI)
\]

- \( f_\rho \) represents the compositions of flow functions along \( \rho \)
- \( BI \) refers to the relevant information from the calling context
- All execution paths are considered potentially executable by ignoring the results of conditionals

- Any \( \text{Info}(p) \subseteq \text{MoP}(p) \) is safe
Maximum Fixed Point (MFP) Assignment

- Difficulties in computing MoP assignment
Maximum Fixed Point (MFP) Assignment

- Difficulties in computing MoP assignment
  - In the presence of cycles there are infinite paths
    - If all paths need to be traversed $\Rightarrow$ Undecidability
Maximum Fixed Point (MFP) Assignment

- Difficulties in computing MoP assignment
  - In the presence of cycles there are infinite paths
    - If all paths need to be traversed $\Rightarrow$ Undecidability
  - Even if a program is acyclic, every conditional multiplies the number of paths by two
    - If all paths need to be traversed $\Rightarrow$ Intractability
Maximum Fixed Point (MFP) Assignment

- Difficulties in computing MoP assignment
  - In the presence of cycles there are infinite paths
    - If all paths need to be traversed ⇒ Undecidability
  - Even if a program is acyclic, every conditional multiplies the number of paths by two
    - If all paths need to be traversed ⇒ Intractability
- Why not merge information at intermediate points?
  - Merging is safe but may lead to imprecision
  - Computes fixed point solutions of data flow equations
Maximum Fixed Point (MFP) Assignment

• Difficulties in computing MoP assignment
  ▶ In the presence of cycles there are infinite paths
    If all paths need to be traversed ⇒ Undecidability
  ▶ Even if a program is acyclic, every conditional multiplies the number of paths by two
    If all paths need to be traversed ⇒ Intractability

• Why not merge information at intermediate points?
  ▶ Merging is safe but may lead to imprecision
  ▶ Computes fixed point solutions of data flow equations
Computing MFP Vs. Computing MoP

Expression Tree for MFP

Program

Expression Tree for MoP

\[ f_1 \]

\[ f_2 \]

\[ f_3 \]

\[ f_4 \]

\[ f_5 \]
Computing MFP Vs. Computing MoP

Expression Tree for MFP

Expression Tree for MoP

Program
Computing MFP Vs. Computing MoP

Expression Tree for MFP

Program

Expression Tree for MoP

Aug 2017 IIT Bombay
Assignments for Constant Propagation Example

\[ a = 1 \]
\[ b = 2 \]
\[ c = a + b \]

\[ c = a + b \]
\[ d = a \times b \]

\[ d = c - 1 \]
\[ a = 2 \]
\[ b = 1 \]
\[ c = a + b \]
Assignments for Constant Propagation Example

MoP

\[ \langle \top, \top, \top, \top \rangle \]
\[ \langle 1, 2, 3, \top \rangle \]
\[ \langle \bot, \bot, 3, 2 \rangle \]
\[ \langle \bot, \bot, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
Assignments for Constant Propagation Example

\[ a = 1 \]
\[ b = 2 \]
\[ c = a + b \]

\[ n_1 \]

\[ c = a + b \]
\[ d = a \times b \]

\[ n_2 \]

\[ d = c - 1 \]
\[ a = 2 \]
\[ b = 1 \]
\[ c = a + b \]

\[ n_3 \]

\[ \text{MoP} \]
\[ \langle \top, \top, \top, \top \rangle \]
\[ \langle 1, 2, 3, \top \rangle \]
\[ \langle \bot, \bot, 3, 2 \rangle \]
\[ \langle \bot, \bot, 3, \top \rangle \]

\[ \text{MFP} \]
\[ \langle \top, \top, \top, \top \rangle \]
\[ \langle 1, 2, 3, \top \rangle \]
\[ \langle \bot, \bot, 3, \bot \rangle \]
\[ \langle \bot, \bot, 3, \bot \rangle \]
\[ \langle \bot, \bot, \bot, \bot \rangle \]
Possible Assignments as Solutions of Data Flow Analyses

All possible assignments
Possible Assignments as Solutions of Data Flow Analyses

All possible assignments

All safe assignments
Possible Assignments as Solutions of Data Flow Analyses

- All possible assignments
- All safe assignments
- All fixed point solutions
Possible Assignments as Solutions of Data Flow Analyses

All possible assignments

All safe assignments

All fixed point solutions

∀i, Inᵢ = Outᵢ = ⊤
Possible Assignments as Solutions of Data Flow Analyses

- All possible assignments
- All safe assignments
- All fixed point solutions

\( \forall i, \text{In}_i = \text{Out}_i = \top \)

\( \forall i, \text{In}_i = \text{Out}_i = \bot \)
Possible Assignments as Solutions of Data Flow Analyses

- All possible assignments: $\forall i, In_i = Out_i = \top$
- All safe assignments
- All fixed point solutions: $\forall i, In_i = Out_i = \bot$

Meet Over Paths Assignment
Possible Assignments as Solutions of Data Flow Analyses

- All possible assignments
- All safe assignments
- All fixed point solutions

\( \forall i, \text{ln}_i = \text{out}_i = \top \)

Meet Over Paths Assignment

\( \forall i, \text{ln}_i = \text{out}_i = \bot \)

Maximum Fixed Point

Aug 2017
Possible Assignments as Solutions of Data Flow Analyses

- All possible assignments
- All safe assignments
- All fixed point solutions

∀i, Inᵢ = Outᵢ = ⊤

Meet Over Paths Assignment

Maximum Fixed Point

Least Fixed Point

∀i, Inᵢ = Outᵢ = ⊥
An Instance of Available Expressions Analysis

Lattice

<table>
<thead>
<tr>
<th>Constant Functions</th>
<th>Dependent Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f )</td>
<td>( f(x) )</td>
</tr>
<tr>
<td>( f_T )</td>
<td>( {a \ast b, b \ast c} )</td>
</tr>
<tr>
<td>( f_\perp )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( f_a )</td>
<td>( {a \ast b} )</td>
</tr>
<tr>
<td>( f_b )</td>
<td>( {b \ast c} )</td>
</tr>
<tr>
<td>( f_f )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
An Instance of Available Expressions Analysis

Lattice

\[ \{a\ast b, b\ast c\} \]
\[ \{a\ast b\} \quad \{b\ast c\} \]
\[ \emptyset \]

<table>
<thead>
<tr>
<th>Constant Functions</th>
<th>Dependent Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f)</td>
<td>(f(x))</td>
</tr>
<tr>
<td>(f_\top)</td>
<td>({a\ast b, b\ast c})</td>
</tr>
<tr>
<td>(f_\bot)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>(f_a)</td>
<td>({a\ast b})</td>
</tr>
<tr>
<td>(f_b)</td>
<td>({b\ast c})</td>
</tr>
<tr>
<td>(f_f)</td>
<td>(x - {b\ast c})</td>
</tr>
</tbody>
</table>

- Is the lattice a meet semilattice?
An Instance of Available Expressions Analysis

- Is the lattice a meet semilattice?
- What is the meet operation that computes $\text{glb}$?

### Lattice

- $\{a \times b, b \times c\}$
- $\{a \times b\}$
- $\{b \times c\}$
- $\emptyset$

### Tables

<table>
<thead>
<tr>
<th>Constant Functions</th>
<th>Dependent Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f$</td>
<td>$f(x)$</td>
</tr>
<tr>
<td>$f_{\top}$</td>
<td>${a \times b, b \times c}$</td>
</tr>
<tr>
<td>$f_{\bot}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$f_a$</td>
<td>${a \times b}$</td>
</tr>
<tr>
<td>$f_b$</td>
<td>${b \times c}$</td>
</tr>
<tr>
<td>$f_f$</td>
<td>$x - {b \times c}$</td>
</tr>
</tbody>
</table>
### An Instance of Available Expressions Analysis

#### Lattice

\[
\begin{align*}
\{a*b, b*c\} & \quad \{a*b\} \quad \{b*c\} \\
\{a*b\} & \quad \{b*c\} \\
\emptyset &
\end{align*}
\]

<table>
<thead>
<tr>
<th>Constant Functions</th>
<th>Dependent Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f )</td>
<td>( f(x) )</td>
</tr>
<tr>
<td>( f_\top )</td>
<td>( {a<em>b, b</em>c} )</td>
</tr>
<tr>
<td>( f_\perp )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( f_a )</td>
<td>( {a*b} )</td>
</tr>
<tr>
<td>( f_b )</td>
<td>( {b*c} )</td>
</tr>
<tr>
<td>( f_f )</td>
<td>( x - {b*c} )</td>
</tr>
</tbody>
</table>

- Is the lattice a meet semilattice?
- What is the meet operation that computes glb?
- Are all strictly descending chains finite?
An Instance of Available Expressions Analysis

Lattice

\{a*b, b*c\}

\{a*b\} \quad \{b*c\}

\emptyset

<table>
<thead>
<tr>
<th>Constant Functions</th>
<th>Dependent Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f)</td>
<td>(f(x))</td>
</tr>
<tr>
<td>(f_\top)</td>
<td>{a<em>b, b</em>c}</td>
</tr>
<tr>
<td>(f_\bot)</td>
<td>\emptyset</td>
</tr>
<tr>
<td>(f_a)</td>
<td>{a*b}</td>
</tr>
<tr>
<td>(f_b)</td>
<td>{b*c}</td>
</tr>
<tr>
<td>(f_c)</td>
<td>(x \cup {a*b})</td>
</tr>
<tr>
<td>(f_d)</td>
<td>(x \cup {b*c})</td>
</tr>
<tr>
<td>(f_e)</td>
<td>(x - {a*b})</td>
</tr>
<tr>
<td>(f_f)</td>
<td>(x - {b*c})</td>
</tr>
</tbody>
</table>

- Is the lattice a meet semilattice?
- What is the meet operation that computes glb?
- Are all strictly descending chains finite?
- Does the function space have an identity function?
An Instance of Available Expressions Analysis

Lattice

\{a \ast b, b \ast c\}
\{a \ast b\} \quad \{b \ast c\}
\emptyset

<table>
<thead>
<tr>
<th></th>
<th>Constant Functions</th>
<th>Dependent Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f$</td>
<td>$f(x)$</td>
<td>$f$</td>
</tr>
<tr>
<td>$f_\top$</td>
<td>${a \ast b, b \ast c}$</td>
<td>$f_{id}$</td>
</tr>
<tr>
<td>$f_\bot$</td>
<td>$\emptyset$</td>
<td>$f_c$</td>
</tr>
<tr>
<td>$f_a$</td>
<td>${a \ast b}$</td>
<td>$f_d$</td>
</tr>
<tr>
<td>$f_b$</td>
<td>${b \ast c}$</td>
<td>$f_e$</td>
</tr>
<tr>
<td>$f_f$</td>
<td></td>
<td>$f_f$</td>
</tr>
</tbody>
</table>

- Is the lattice a meet semilattice?
- What is the meet operation that computes glb?
- Are all strictly descending chains finite?
- Does the function space have an identity function?
- Are all values in the lattice computable from a finite merge of flow functions?
An Instance of Available Expressions Analysis

Lattice

\{a \ast b, b \ast c\} \\
\{a \ast b\} \quad \{b \ast c\} \\
\emptyset

<table>
<thead>
<tr>
<th>Constant Functions</th>
<th>Dependent Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f )</td>
<td>( f(x) )</td>
</tr>
<tr>
<td>( f_\top )</td>
<td>{a \ast b, b \ast c}</td>
</tr>
<tr>
<td>( f_\bot )</td>
<td>\emptyset</td>
</tr>
<tr>
<td>( f_a )</td>
<td>{a \ast b}</td>
</tr>
<tr>
<td>( f_b )</td>
<td>{b \ast c}</td>
</tr>
<tr>
<td>( f_f )</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>

- Is the lattice a meet semilattice?
- What is the meet operation that computes glb?
- Are all strictly descending chains finite?
- Does the function space have an identity function?
- Are all values in the lattice computable from a finite merge of flow functions?
- Is the function space closed under composition?
An Instance of Available Expressions Analysis

Lattice

\[
\{a \ast b, b \ast c\} \\
\{a \ast b\} \\
\emptyset \\
\{b \ast c\}
\]

<table>
<thead>
<tr>
<th></th>
<th>Constant Functions</th>
<th>Dependent Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f )</td>
<td>( f(x) )</td>
<td>( f )</td>
</tr>
<tr>
<td>( f_\top )</td>
<td>( {a \ast b, b \ast c} )</td>
<td>( f_{id} )</td>
</tr>
<tr>
<td>( f_\perp )</td>
<td>( \emptyset )</td>
<td>( f_c )</td>
</tr>
<tr>
<td>( f_a )</td>
<td>( {a \ast b} )</td>
<td>( f_d )</td>
</tr>
<tr>
<td>( f_b )</td>
<td>( {b \ast c} )</td>
<td>( f_e )</td>
</tr>
<tr>
<td>( f_f )</td>
<td>( \emptyset )</td>
<td>( f_f )</td>
</tr>
</tbody>
</table>
An Instance of Available Expressions Analysis

Lattice

\{a \ast b, b \ast c\}
\{a \ast b\}
\{b \ast c\}
\emptyset

<table>
<thead>
<tr>
<th>Constant Functions</th>
<th>Dependent Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f )</td>
<td>( f(x) )</td>
</tr>
<tr>
<td>( f_T )</td>
<td>{a \ast b, b \ast c}</td>
</tr>
<tr>
<td>( f_{\perp} )</td>
<td>\emptyset</td>
</tr>
<tr>
<td>( f_a )</td>
<td>{a \ast b}</td>
</tr>
<tr>
<td>( f_b )</td>
<td>{b \ast c}</td>
</tr>
<tr>
<td>( f_c )</td>
<td>( x \cup {a \ast b} )</td>
</tr>
<tr>
<td>( f_d )</td>
<td>( x \cup {b \ast c} )</td>
</tr>
<tr>
<td>( f_e )</td>
<td>( x - {a \ast b} )</td>
</tr>
<tr>
<td>( f_f )</td>
<td>( x - {b \ast c} )</td>
</tr>
</tbody>
</table>

Program

1
\( a \ast b \)
\( b \ast c \)

2
An Instance of Available Expressions Analysis

Lattice

\{a \ast b, b \ast c\}

\{a \ast b\} \quad \{b \ast c\}

\emptyset

<table>
<thead>
<tr>
<th>Constant Functions</th>
<th>Dependent Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f )</td>
<td>( f(x) )</td>
</tr>
<tr>
<td>( f_{\top} )</td>
<td>{a \ast b, b \ast c}</td>
</tr>
<tr>
<td>( f_{\bot} )</td>
<td>\emptyset</td>
</tr>
<tr>
<td>( f_a )</td>
<td>{a \ast b}</td>
</tr>
<tr>
<td>( f_b )</td>
<td>{b \ast c}</td>
</tr>
<tr>
<td>( f_c )</td>
<td>\emptyset</td>
</tr>
<tr>
<td>( f_d )</td>
<td>\emptyset</td>
</tr>
<tr>
<td>( f_e )</td>
<td>\emptyset</td>
</tr>
<tr>
<td>( f_f )</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>

Program

1

2

\( a \ast b \)

\( b \ast c \)

Flow Functions

<table>
<thead>
<tr>
<th>Node</th>
<th>Flow Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( f_{\top} )</td>
</tr>
<tr>
<td>2</td>
<td>( f_{id} )</td>
</tr>
</tbody>
</table>
An Instance of Available Expressions Analysis

Lattice

\{a*b, b*c\}
\{a*b\} \{b*c\}
\emptyset

<table>
<thead>
<tr>
<th>Constant Functions</th>
<th>Dependent Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f )</td>
<td>( f(x) )</td>
</tr>
<tr>
<td>( f_T )</td>
<td>{a<em>b, b</em>c}</td>
</tr>
<tr>
<td>( f_\perp )</td>
<td>\emptyset</td>
</tr>
<tr>
<td>( f_a )</td>
<td>{a*b}</td>
</tr>
<tr>
<td>( f_b )</td>
<td>{b*c}</td>
</tr>
<tr>
<td>( f_f )</td>
<td>\emptyset</td>
</tr>
<tr>
<td>( f_{id} )</td>
<td>( x )</td>
</tr>
<tr>
<td>( f_c )</td>
<td>( x \cup {a*b} )</td>
</tr>
<tr>
<td>( f_d )</td>
<td>( x \cup {b*c} )</td>
</tr>
<tr>
<td>( f_e )</td>
<td>( x - {a*b} )</td>
</tr>
<tr>
<td>( f_f )</td>
<td>( x - {b*c} )</td>
</tr>
</tbody>
</table>

Program

1
\( a*b \)
\( b*c \)

2

Flow Functions

<table>
<thead>
<tr>
<th>Node</th>
<th>Flow Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( f_T )</td>
</tr>
<tr>
<td>2</td>
<td>( f_{id} )</td>
</tr>
</tbody>
</table>

Some Possible Assignments

<table>
<thead>
<tr>
<th>In(_1)</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
<th>A6</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Out(_1)</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
<th>A6</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>In(_2)</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
<th>A6</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>11</td>
<td>10</td>
<td>01</td>
<td>01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Out(_2)</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
<th>A6</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>11</td>
<td>10</td>
<td>01</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An Instance of Available Expressions Analysis

Lattice

\{a \ast b, b \ast c\}

\{a \ast b\}

- Maximum fixed point assignment
- Initialization for round robin iterative method: 11
- Safe assignment

Program

Flow Functions

<table>
<thead>
<tr>
<th>Node</th>
<th>Flow Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>f_T</td>
</tr>
<tr>
<td>2</td>
<td>f_{id}</td>
</tr>
</tbody>
</table>

Some Possible Assignments

<table>
<thead>
<tr>
<th>In_1</th>
<th>A_1</th>
<th>A_2</th>
<th>A_3</th>
<th>A_4</th>
<th>A_5</th>
<th>A_6</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Out_1</th>
<th>A_1</th>
<th>A_2</th>
<th>A_3</th>
<th>A_4</th>
<th>A_5</th>
<th>A_6</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>In_2</th>
<th>A_1</th>
<th>A_2</th>
<th>A_3</th>
<th>A_4</th>
<th>A_5</th>
<th>A_6</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>01</td>
<td>01</td>
<td>01</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Out_2</th>
<th>A_1</th>
<th>A_2</th>
<th>A_3</th>
<th>A_4</th>
<th>A_5</th>
<th>A_6</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>01</td>
<td>01</td>
<td>10</td>
</tr>
</tbody>
</table>
An Instance of Available Expressions Analysis

Lattice

\{a*b, b*c\} → \{a*b\} → \emptyset

- Not a fixed point assignment
- Safe assignment

Program

1
\(a*b\)
\(b*c\)

2

<table>
<thead>
<tr>
<th>Node</th>
<th>Flow Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(f_T)</td>
</tr>
<tr>
<td>2</td>
<td>(f_{id})</td>
</tr>
</tbody>
</table>

Flow Functions

<table>
<thead>
<tr>
<th>Node</th>
<th>Flow Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(f_T)</td>
</tr>
<tr>
<td>2</td>
<td>(f_{id})</td>
</tr>
</tbody>
</table>

Some Possible Assignments

<table>
<thead>
<tr>
<th>In(n_1)</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
<th>A6</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Out(n_1)</th>
<th>11</th>
<th>00</th>
<th>11</th>
<th>11</th>
<th>11</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>In(n_2)</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>Out(n_2)</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>10</td>
</tr>
</tbody>
</table>

Aug 2017

IIT Bombay
An Instance of Available Expressions Analysis

Lattice

\{a*b, b*c\} → \{a*b\} → \{b*c\} → ∅

- Minimum fixed point assignment
- Initialization for round robin iterative method: 00
- Safe assignment

Program

\(a*b\)

\(b*c\)

Flow Functions

<table>
<thead>
<tr>
<th>Node</th>
<th>Flow Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(f_\top)</td>
</tr>
<tr>
<td>2</td>
<td>(f_{id})</td>
</tr>
</tbody>
</table>

Some Possible Assignments

<table>
<thead>
<tr>
<th>In1</th>
<th>00</th>
<th>00</th>
<th>00</th>
<th>00</th>
<th>00</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out1</td>
<td>11</td>
<td>00</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>In2</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>Out2</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>10</td>
</tr>
</tbody>
</table>

Constant Functions

<table>
<thead>
<tr>
<th>(f)</th>
<th>(f(x))</th>
</tr>
</thead>
</table>

Dependent Functions

<table>
<thead>
<tr>
<th>(f)</th>
<th>(f(x))</th>
</tr>
</thead>
</table>

- \(f_\top\) → \(x\)
- \(f_{id}\) → \(x\) ∪ \(\{a*b\}\)
- \(f_{\bot}\) → \(x\) ∪ \(\{b*c\}\)
- \(f_a\) → \(x\) − \(\{a*b\}\)
- \(f_b\) → \(x\) − \(\{b*c\}\)
An Instance of Available Expressions Analysis

Lattice

- Fixed point assignment which is neither maximum nor minimum
- Initialization for round robin iterative method: 10
- Safe assignment

Program

Flow Functions

<table>
<thead>
<tr>
<th>Node</th>
<th>Flow Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( f_\top )</td>
</tr>
<tr>
<td>2</td>
<td>( f_{id} )</td>
</tr>
</tbody>
</table>

Some Possible Assignments

<table>
<thead>
<tr>
<th></th>
<th>( A_1 )</th>
<th>( A_2 )</th>
<th>( A_3 )</th>
<th>( A_4 )</th>
<th>( A_5 )</th>
<th>( A_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( In_1 )</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>( Out_1 )</td>
<td>11</td>
<td>00</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>( In_2 )</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>( Out_2 )</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>10</td>
</tr>
</tbody>
</table>
An Instance of Available Expressions Analysis

Lattice

\{a*b, b*c\}

- Fixed point assignment which is neither maximum nor minimum
- Initialization for round robin iterative method: 01
- Safe assignment

Program

1
- \(a*b\)
- \(b*c\)

2

Flow Functions

<table>
<thead>
<tr>
<th>Node</th>
<th>Flow Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(f_T)</td>
</tr>
<tr>
<td>2</td>
<td>(f_id)</td>
</tr>
</tbody>
</table>

Constant Functions

<table>
<thead>
<tr>
<th>(f)</th>
<th>(f(x))</th>
</tr>
</thead>
</table>

Dependent Functions

<table>
<thead>
<tr>
<th>(f)</th>
<th>(f(x))</th>
</tr>
</thead>
</table>

Some Possible Assignments

<table>
<thead>
<tr>
<th>(A_1)</th>
<th>(A_2)</th>
<th>(A_3)</th>
<th>(A_4)</th>
<th>(A_5)</th>
<th>(A_6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(In_1)</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>(Out_1)</td>
<td>11</td>
<td>00</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>(In_2)</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>(Out_2)</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
</tr>
</tbody>
</table>

Aug 2017 IIT Bombay
An Instance of Available Expressions Analysis

Lattice

\{a*b, b*c\}
\{a*b\}
\emptyset

- Not a fixed point assignment
- Safe assignment

Program

a*b
b*c

Flow Functions

<table>
<thead>
<tr>
<th>Node</th>
<th>Flow Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>f_T</td>
</tr>
<tr>
<td>2</td>
<td>f_id</td>
</tr>
</tbody>
</table>

Some Possible Assignments

<table>
<thead>
<tr>
<th></th>
<th>A_1</th>
<th>A_2</th>
<th>A_3</th>
<th>A_4</th>
<th>A_5</th>
<th>A_6</th>
</tr>
</thead>
<tbody>
<tr>
<td>ln₁</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>out₁</td>
<td>11</td>
<td>00</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>ln₂</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>out₂</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>10</td>
</tr>
</tbody>
</table>
Lattice of Assignments for Available Expressions Analysis

Program

\[
\begin{align*}
1 &: \quad a \times b \\
2 &: 
\end{align*}
\]

Some Assignments

<table>
<thead>
<tr>
<th></th>
<th>(A_0)</th>
<th>(A_1)</th>
<th>(A_2)</th>
<th>(A_3)</th>
<th>(A_4)</th>
<th>(A_5)</th>
<th>(A_6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(In_1)</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>(Out_1)</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>(In_2)</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>(Out_2)</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>10</td>
</tr>
</tbody>
</table>
Lattice of Assignments for Available Expressions Analysis

Program

```
1
  a*b
  b*c

2
```

<table>
<thead>
<tr>
<th></th>
<th>A0</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
<th>A6</th>
</tr>
</thead>
<tbody>
<tr>
<td>In₁</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>Out₁</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>In₂</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>Out₂</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>10</td>
</tr>
</tbody>
</table>

Lattice $L \times L \times L \times L$ for all assignments
(many assignments omitted, e.g. node 1 could have data flow values 10 and 01)
### Lattice of Assignments for Available Expressions Analysis

#### Program

1. \(a \times b\)
2. \(b \times c\)

#### Some Assignments

<table>
<thead>
<tr>
<th></th>
<th>(A_0)</th>
<th>(A_1)</th>
<th>(A_2)</th>
<th>(A_3)</th>
<th>(A_4)</th>
<th>(A_5)</th>
<th>(A_6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(In_1)</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>(Out_1)</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>(In_2)</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>(Out_2)</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>10</td>
</tr>
</tbody>
</table>

#### Lattice

Lattice \(L \times L \times L \times L\) for all assignments (many assignments omitted, e.g. node 1 could have data flow values 10 and 01)

#### Safe Assignments

- \(A_0\)
- \(A_1\)
- \(A_4\)
- \(A_5\)
- \(A_6\)
- \(A_3\)
- \(A_2\)
Lattice of Assignments for Available Expressions Analysis

Program

\[
\begin{array}{c}
\text{In}_1 \\
\text{Out}_1 \\
\text{In}_2 \\
\text{Out}_2
\end{array}
\]

Some Assignments

<table>
<thead>
<tr>
<th></th>
<th>(A_0)</th>
<th>(A_1)</th>
<th>(A_2)</th>
<th>(A_3)</th>
<th>(A_4)</th>
<th>(A_5)</th>
<th>(A_6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{In}_1)</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>(\text{Out}_1)</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>(\text{In}_2)</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>(\text{Out}_2)</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>10</td>
</tr>
</tbody>
</table>

Lattice \(L \times L \times L \times L\) for all assignments (many assignments omitted, e.g. node 1 could have data flow values 10 and 01)

Safe assignments

Fixed point assignments
Existence of an MoP Assignment (1)

\[ MoP(p) = \bigcap_{\rho \in \text{Paths}(p)} f_{\rho}(BL) \]

- If a finite number of paths reach \( p \), then existence of solution trivially follows
  - Function space is closed under composition
  - \( \text{glb} \) exists for all non-empty finite subsets of the lattice
    (Assuming that the data flow values form a meet semilattice)
Existence of an MoP Assignment (2)

\[ MoP(p) = \bigcap_{\rho \in \text{Paths}(p)} f_\rho(BI) \]

- If an infinite number of paths reach \( p \) then,

\[ MoP(p) = f_{\rho_1}(BI) \cap f_{\rho_2}(BI) \cap f_{\rho_3}(BI) \cap \ldots \]
Existence of an MoP Assignment (2)

\[ MoP(p) = \bigcap_{\rho \in \text{Paths}(p)} f_\rho(BI) \]

- If an infinite number of paths reach \( p \) then,

\[ MoP(p) = f_{\rho_1}(BI) \cap f_{\rho_2}(BI) \cap f_{\rho_3}(BI) \cap \ldots \]

\[ X_1 \]
Existence of an MoP Assignment (2)

\[ \text{MoP}(p) = \bigsqcap_{\rho \in \text{Paths}(p)} f_{\rho}(B I) \]

- If an infinite number of paths reach \( p \) then,

\[ \text{MoP}(p) = f_{\rho_1}(B I) \sqcap f_{\rho_2}(B I) \sqcap f_{\rho_3}(B I) \sqcap \ldots \]

\[ X_1 \]

\[ X_2 \]

- Every meet results in a weaker value
Existence of an MoP Assignment (2)

\[ MoP(p) = \bigsqcap_{\rho \in \text{Paths}(p)} f_\rho(BI) \]

- If an infinite number of paths reach \( p \) then,

\[ MoP(p) = f_{\rho_1}(BI) \sqcap f_{\rho_2}(BI) \sqcap f_{\rho_3}(BI) \sqcap \ldots \]

\[ X_1 \]

\[ X_2 \]

\[ X_3 \]

- Every meet results in a weaker value
Existence of an MoP Assignment (2)

\[ \text{MoP}(p) = \bigsqcap_{\rho \in \text{Paths}(p)} f_\rho(BI) \]

- If an infinite number of paths reach \( p \) then,

\[ \text{MoP}(p) = f_{\rho_1}(BI) \sqcap f_{\rho_2}(BI) \sqcap f_{\rho_3}(BI) \sqcap \ldots \]

\[ \begin{array}{c}
\hline
X_1 \\
\hline
X_2 \\
\hline
X_3 \\
\end{array} \]

- Every meet results in a weaker value
- The sequence \( X_1, X_2, X_3, \ldots \) follows a descending chain
- Since all strictly descending chains are finite, MoP exists

(Assuming that our meet semilattice satisfies DCC)
Does existence of MoP imply it is computable?

\[
\begin{align*}
\text{MoP}(p_2) &= x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap f^4(x) \sqcap \ldots
\end{align*}
\]

<table>
<thead>
<tr>
<th>Paths reaching the entry of ( p_2 )</th>
<th>Data Flow Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_1, p_2 )</td>
<td>( x )</td>
</tr>
<tr>
<td>( p_1, p_2, p_3, p_2 )</td>
<td>( f(x) )</td>
</tr>
<tr>
<td>( p_1, p_2, p_3, p_2, p_3, p_2 )</td>
<td>( f(f(x)) = f^2(x) )</td>
</tr>
<tr>
<td>( p_1, p_2, p_3, p_2, p_3, p_3, p_2 )</td>
<td>( f(f(f(x))) = f^3(x) )</td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
</tr>
</tbody>
</table>
MoP Computation is Undecidable

There does not exist any algorithm that can compute MoP assignment for every possible instance of every possible monotone data flow framework

- Reducing MPCP (Modified Post’s Correspondence Problem) to constant propagation
- MPCP is known to be undecidable
- If an algorithm exists for detecting all constants
  \[ \Rightarrow \] MPCP would be decidable
- Since MPCP is undecidable
  \[ \Rightarrow \] There does not exist an algorithm for detecting all constants
  \[ \Rightarrow \] Static analysis is undecidable
Post’s Correspondence Problem (PCP)

- Given strings \( u_i, v_i \in \Sigma^+ \) for some alphabet \( \Sigma \), and two \( k \)-tuples,

\[
U = (u_1, u_2, \ldots, u_k)
\]
\[
V = (v_1, v_2, \ldots, v_k)
\]

Is there a sequence \( i_1, i_2, \ldots, i_m \) of one or more integers such that

\[
u_{i_1} u_{i_2} \ldots u_{i_m} = v_{i_1} v_{i_2} \ldots v_{i_m}
\]
Post’s Correspondence Problem (PCP)

- Given strings $u_i, v_i \in \Sigma^+$ for some alphabet $\Sigma$, and two $k$-tuples,

\[
U = (u_1, u_2, \ldots, u_k) \\
V = (v_1, v_2, \ldots, v_k)
\]

Is there a sequence $i_1, i_2, \ldots, i_m$ of one or more integers such that

\[
u_{i_1} u_{i_2} \ldots u_{i_m} = v_{i_1} v_{i_2} \ldots v_{i_m}
\]

- For $U = (101, 11, 100)$ and $V = (01, 1, 11001)$ the solution is 2, 3, 2

\[
u_2 u_3 u_2 = 1110011 \\
v_2 v_3 v_2 = 1110011
\]
Post’s Correspondence Problem (PCP)

- Given strings $u_i, v_i \in \Sigma^+$ for some alphabet $\Sigma$, and two $k$-tuples,

$$
U = (u_1, u_2, \ldots, u_k) \\
V = (v_1, v_2, \ldots, v_k)
$$

Is there a sequence $i_1, i_2, \ldots, i_m$ of one or more integers such that

$$
u_{i_1} u_{i_2} \ldots u_{i_m} = v_{i_1} v_{i_2} \ldots v_{i_m}
$$

- For $U = (101, 11, 100)$ and $V = (01, 1, 11001)$ the solution is 2, 3, 2

$$
u_2 u_3 u_2 = 1110011 \\
v_2 v_3 v_2 = 1110011
$$

- For $U = (1, 10111, 10), V = (111, 10, 0)$, the solution is 2, 1, 1, 3
Post’s Correspondence Problem (PCP)

- Given strings $u_i, v_i \in \Sigma^+$ for some alphabet $\Sigma$, and two $k$-tuples,

\[
U = (u_1, u_2, \ldots, u_k) \\
V = (v_1, v_2, \ldots, v_k)
\]

Is there a sequence $i_1, i_2, \ldots, i_m$ of one or more integers such that

\[
u_{i_1} u_{i_2} \ldots u_{i_m} = v_{i_1} v_{i_2} \ldots v_{i_m}
\]

- For $U = (101, 11, 100)$ and $V = (01, 1, 11001)$ the solution is $2, 3, 2$

\[
u_2 u_3 u_2 = 1110011 \\
v_2 v_3 v_2 = 1110011
\]

- For $U = (1, 10111, 10), V = (111, 10, 0)$, the solution is $2, 1, 1, 3$

- For $U = (01, 110), V = (00, 11)$, there is no solution
Post’s Correspondence Problem (PCP)

• Given strings $u_i, v_i \in \Sigma^+$ for some alphabet $\Sigma$, and two $k$-tuples,

\[
U = (u_1, u_2, \ldots, u_k) \\
V = (v_1, v_2, \ldots, v_k)
\]

Is there a sequence $i_1, i_2, \ldots, i_m$ of one or more integers such that

\[
u_{i_1} u_{i_2} \ldots u_{i_m} = v_{i_1} v_{i_2} \ldots v_{i_m}
\]

• Sets $U$ and $V$ are finite and contain the same number of strings

• The strings in $U$ and $V$ are finite and are of varying lengths

• For constructing the new strings using the strings in $U$ and $V$
  
  ▶ The strings at the same the index of must be used
  
  ▶ There is no limit on the length of the new string

Indices could repeat without any bound
Modified Post’s Correspondence Problem (MPCP)

- The first string in the correspondence relation should be the first string from the $k$-tuple

$$u_1 u_{i_1} u_{i_2} \ldots u_{i_m} = v_1 v_{i_1} v_{i_2} \ldots v_{i_m}$$
Modified Post’s Correspondence Problem (MPCP)

- The first string in the correspondence relation should be the first string from the $k$-tuple

  $$u_1 u_{i_1} u_{i_2} \ldots u_{i_m} = v_1 v_{i_1} v_{i_2} \ldots v_{i_m}$$

- For $U = (11, 1, 0111, 10)$, $V = (1, 111, 10, 0)$, the solution is 3, 2, 2, 4

  $$u_1 u_3 u_2 u_2 u_4 = 1101111110$$

  $$v_1 v_3 v_2 v_2 v_4 = 1101111110$$
Hecht’s Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with \( \Sigma = \{0, 1\} \)

\[
\begin{align*}
x &= u_1; y = v_1 \\
x &= x \oplus u_2 \\
y &= y \oplus v_2 \\
x &= x \oplus u_3 \\
y &= y \oplus v_3 \\
x &= x \oplus u_k \\
y &= y \oplus v_k \\
i &= \text{atoi}(x); j = \text{atoi}(y) \\
r &= \frac{1}{((i - j)^2 + 1)}
\end{align*}
\]
Hecht’s Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$

$$x = u_1; y = v_1$$

Random branching for random selection of index

Each block in the loop corresponds to a particular index

$$x = x@u_2; \quad y = y@v_2$$

$$x = x@u_3; \quad y = y@v_3$$

$$x = x@u_k; \quad y = y@v_k$$

$$i = \text{atoi}(x); \quad j = \text{atoi}(y)$$

$$r = 1/((i - j)^2 + 1)$$
Hecht’s Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$

$\begin{align*}
x &= u_1; y = v_1 \\
x &= x \oplus u_2; y = y \oplus v_2 \\
x &= x \oplus u_3; y = y \oplus v_3 \\
x &= x \oplus u_k; y = y \oplus v_k \\
i &= \text{atoi}(x); j = \text{atoi}(y) \\
r &= \frac{1}{((i - j)^2 + 1)}
\end{align*}$
Hecht’s Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$

$x = u_1; y = v_1$

$x = x @ u_2$
$y = y @ v_2$

$x = x @ u_3$
$y = y @ v_3$

$u_i, v_i$

$x = x @ u_k$
$y = y @ v_k$

$i = \text{atoi}(x); j = \text{atoi}(y)$

$r = 1/((i - j)^2 + 1)$

Each block in the loop corresponds to a particular index

Random branching for random selection of index

String append

String to integer conversion
Hecht’s Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with \( \Sigma = \{0, 1\} \)

\[
\begin{align*}
  x &= u_1; y = v_1 \\
  x &= x \oplus u_2; \\
  y &= y \oplus v_2 \\
  x &= x \oplus u_3; \\
  y &= y \oplus v_3 \\
  x &= x \oplus u_k; \\
  y &= y \oplus v_k \\
  i &= \text{atoi}(x); j = \text{atoi}(y) \\
  r &= 1/((i - j)^2 + 1)
\end{align*}
\]

Each block in the loop corresponds to a particular index.

Random branching for random selection of index.

String append.

String to integer conversion.

Integer division.

MoP computation. No merge at intermediate points. Merge only at the point of interest.
Hecht’s Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$

$x = u_1; y = v_1$

$x = x@u_2$
$y = y@v_2$

$x = x@u_3$
$y = y@v_3$

$x = x@u_k$
$y = y@v_k$

\[ i = \text{atoi}(x); j = \text{atoi}(y) \]
\[ r = 1/((i - j)^2 + 1) \]
Hecht’s Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$

- $i = j \Rightarrow r = 1$
- $i \neq j \Rightarrow r = 0$

\[ x = u_1; y = v_1 \]

\[ x = x \oplus u_2; y = y \oplus v_2 \]

\[ x = x \oplus u_3; y = y \oplus v_3 \]

\[ x = x \oplus u_k; y = y \oplus v_k \]

\[ i = \text{atoi}(x); j = \text{atoi}(y) \]

\[ r = 1/((i - j)^2 + 1) \]
Hecht’s Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$

\[
\begin{align*}
  x &= u_1; y = v_1 \\
  x &= x \oplus u_2; y = y \oplus v_2 \\
  x &= x \oplus u_3; y = y \oplus v_3 \\
  x &= x \oplus u_k; y = y \oplus v_k \\
  i &= \text{atoi}(x); j = \text{atoi}(y) \\
  r &= 1/((i-j)^2 + 1)
\end{align*}
\]

- $i = j \Rightarrow r = 1$
- $i \neq j \Rightarrow r = 0$

\textbf{If} there exists an algorithm which can determine that

\[
\begin{align*}
\{ & \}
\end{align*}
\]
Hecht’s Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$

- $i = j \implies r = 1$
- $i \neq j \implies r = 0$

- If there exists an algorithm which can determine that
  
  $\{ \triangleright r = 0 \text{ along every path} \}$
  
  ($x$ is never equal to $y$, MPCP instance does not have a solution)

$r \mapsto 0 \in \text{MoP}$
Hecht’s Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$

$\begin{align*}
x &= u_1; y = v_1 \\
x &= x @ u_2; y &= y @ v_2 \\
x &= x @ u_3; y &= y @ v_3 \\
x &= x @ u_k; y &= y @ v_k \\
i &= \text{atoi}(x); j &= \text{atoi}(y) \\
r &= 1/((i - j)^2 + 1)
\end{align*}$

- $i = j \Rightarrow r = 1$
- $i \neq j \Rightarrow r = 0$
- **If** there exists an algorithm which can determine that
  
  \[
  \begin{cases}
  r = 0 \text{ along every path} & (x \text{ is never equal to } y, \text{ MPCP instance does not have a solution}) \\
  r = 1 \text{ along some path} & (some x \text{ is equal to } y, \text{ MPCP instance has a solution})
  \end{cases}
  \]

*Then* MPCP is decidable
Hecht’s Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$

\[
x = u_1; y = v_1
\]

\[
x = x \oplus u_2; y = y \oplus v_2
\]

\[
x = x \oplus u_3; y = y \oplus v_3
\]

\[
x = x \oplus u_k; y = y \oplus v_k
\]

\[
i = \text{atoi}(x); j = \text{atoi}(y)
\]

\[
r = 1/((i - j)^2 + 1)
\]

\[
r \mapsto 0 \in \text{MoP}
\]

\[
r \mapsto 1 \in \text{MoP}
\]

\[
r \mapsto \bot \in \text{MoP}
\]

\[
The tricky part!!
\]

- $i = j \implies r = 1$
- $i \neq j \implies r = 0$

- If there exists an algorithm which can determine that
  \[
  \{ \begin{align*}
  & r = 0 \text{ along every path} \\
  & (x \text{ is never equal to } y, \text{ MPCP instance does not have a solution}) \\
  & r = 1 \text{ along some path} \\
  & (some x \text{ is equal to } y, \text{ MPCP instance has a solution})
  \end{align*}
  \]
  Then MPCP is decidable.
Hecht’s Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$

- Asserting that no $x$ is equal to $y$ requires us to examine infinitely many $(x, y)$ pairs
- If we keep finding $x$ and $y$ that are unequal, how long do we wait to decide that there is no $x$ that is equal to $y$?
- In a lucky case we may find an $x$ that is equal to $y$, but there is no guarantee

\[ i = j \Rightarrow r = 1 \]
\[ i \neq j \Rightarrow r = 0 \]

If there exists an algorithm which can determine that
\[ r = 0 \text{ along every path} \]
\[ (x \text{ is never equal to } y, \text{ MPCP instance does not have a solution}) \]

- $r = 1$ along some path
\[ (\text{some } x \text{ is equal to } y, \text{ MPCP instance has a solution}) \]

Then MPCP is decidable
Hecht’s Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$

- Asserting that no $x$ is equal to $y$ requires us to examine infinitely many $(x, y)$ pairs
- If we keep finding $x$ and $y$ that are unequal, how long do we wait to decide that there is no $x$ that is equal to $y$?
- In a lucky case we may find an $x$ that is equal to $y$, but there is no guarantee

**MPCP is not decidable**

$\Rightarrow$ **Constant Propagation is not decidable**

The tricky part!!

\[
i = j \Rightarrow r = 1 \\
i \neq j \Rightarrow r = 0
\]

If there exists an algorithm which can determine that
\[
\{ \\
\quad r = 0 \text{ along every path} \\
\quad (x \text{ is never equal to } y, \text{ MPCP instance does not have a solution}) \\
\}
\]

Then MPCP is decidable
Hecht’s Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$

- Asserting that no $x$ is equal to $y$ requires us to examine infinitely many $(x, y)$ pairs
- If we keep finding $x$ and $y$ that are unequal, how long do we wait to decide that there is no $x$ that is equal to $y$?
- In a lucky case we may find an $x$ that is equal to $y$, but there is no guarantee

**MPCP is not decidable**

$\Rightarrow$ **Constant Propagation is not decidable**

- Descending chains consist of sets of pairs $(x, y)$ with $\top$ as $\emptyset$
  
  Since there is no bound on the length of $x$ and $y$, the number of these sets is infinite
  $\Rightarrow$ DCC is violated

If there exists an algorithm which can determine that

\[
\begin{align*}
  i = j & \Rightarrow r = 1 \\
  i \neq j & \Rightarrow r = 0
\end{align*}
\]

\[
\{ r = 0 \text{ along every path} \} \\
(x \text{ is never equal to } y, \text{ MPCP instance does not have a solution})
\]

$\Rightarrow$ *r = 1 along some path*  
(some $x$ is equal to $y$, MPCP instance has a solution)

Then MPCP is decidable
Is MFP Always Computable?

MFP assignment may not be computable

- if the flow functions are non-monotonic, or
- if some strictly descending chain is not finite
Computability of MFP

- If $f$ is not monotonic, the computation may not converge
Computability of MFP

- If $f$ is not monotonic, the computation may not converge
Computability of MFP

- If $f$ is not monotonic, the computation may not converge

<table>
<thead>
<tr>
<th></th>
<th>$f(x)$</th>
<th>$f^2(x)$</th>
<th>$f^3(x)$</th>
<th>$f^4(x)$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Computability of MFP

- If $f$ is not monotonic, the computation may not converge

\[
\begin{array}{cccccc}
\text{x} & f(x) & f^2(x) & f^3(x) & f^4(x) & \ldots \\
1 & 0 & 1 & 0 & 1 & \ldots \\
\end{array}
\]

\[\text{MoP} = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots = 0\]
Computability of MFP

• If $f$ is not monotonic, the computation may not converge

\[
\begin{array}{cccccc}
 x & f(x) & f^2(x) & f^3(x) & f^4(x) & \ldots \\
1 & 0 & 1 & 0 & 1 & \ldots \\
\end{array}
\]

\[MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots = 0\]

• Computing MFP iteratively
**Computability of MFP**

- If \( f \) is not monotonic, the computation may not converge

\[
\begin{array}{c|c|c|c|c|c}
  & x & f(x) & f^2(x) & f^3(x) & f^4(x) \\
\hline
1 & 1 & 0 & 1 & 0 & 1 \\
\end{array}
\]

\( MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots = 0 \)

- Computing MFP iteratively
Computability of MFP

• If $f$ is not monotonic, the computation may not converge

\[
\begin{array}{cccccc}
  & x & f(x) & f^2(x) & f^3(x) & f^4(x) & \ldots \\
1 & 0 & 1 & 0 & 1 & \ldots \\
\end{array}
\]

\[
MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots = 0
\]

• Computing MFP iteratively
Computability of MFP

- If $f$ is not monotonic, the computation may not converge

\[
\begin{array}{ccccc}
  x & f(x) & f^2(x) & f^3(x) & f^4(x) \\
  1 & 0 & 1 & 0 & 1 \\
\end{array}
\]

\[\text{MoP} = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots = 0\]

- Computing MFP iteratively
Computability of MFP

- If \( f \) is not monotonic, the computation may not converge

\[
\begin{array}{cccccc}
  x & f(x) & f^2(x) & f^3(x) & f^4(x) & \ldots \\
  1 & 0 & 1 & 0 & 1 & \ldots \\
\end{array}
\]

\[\text{MoP} = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots = 0\]

- Computing MFP iteratively
Computability of MFP

• If \( f \) is not monotonic, the computation may not converge

\[
\begin{array}{cccccc}
   x & f(x) & f^2(x) & f^3(x) & f^4(x) & \ldots \\
   1 & 0 & 1 & 0 & 1 & \ldots \\
\end{array}
\]

\[MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots = 0\]

• Computing MFP iteratively
Computability of MFP

- If $f$ is not monotonic, the computation may not converge

$$
\begin{array}{cccccc}
  x & f(x) & f^2(x) & f^3(x) & f^4(x) & \ldots \\
  1 & 0 & 1 & 0 & 1 & \ldots \\
\end{array}
$$

$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots = 0$$

- Computing MFP iteratively
Computability of MFP

- If $f$ is not monotonic, the computation may not converge

\[ \begin{array}{cccccc}
  x & f(x) & f^2(x) & f^3(x) & f^4(x) & \ldots \\
  1 & 0 & 1 & 0 & 1 & \ldots \\
\end{array} \]

\[ \text{MoP} = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots = 0 \]

- Computing MFP iteratively
## Computability of MFP

- If $f$ is not monotonic, the computation may not converge.

\[
\begin{array}{cccccc}
  x & f(x) & f^2(x) & f^3(x) & f^4(x) & \ldots \\
  1 & 0 & 1 & 0 & 1 & \ldots \\
\end{array}
\]

\[MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots = 0\]

- Computing MFP iteratively:

  MFP does not exist and is not computable.

---

Aug 2017

IIT Bombay
Computability of MFP

- If $f$ is not monotonic, the computation may not converge

\[
\begin{array}{ccccccc}
\hline
x & f(x) & f^2(x) & f^3(x) & f^4(x) & \ldots \\
1 & 0 & 1 & 0 & 1 & \ldots \\
\hline
\end{array}
\]

\[MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots = 0\]

- Computing MFP iteratively

MFP does not exist and is not computable
Computability of MFP

- If $f$ is not monotonic, the computation may not converge

\[
\begin{array}{cccccc}
  x & f(x) & f^2(x) & f^3(x) & f^4(x) & \ldots \\
  1 & 0 & 1 & 0 & 1 & \ldots \\
\end{array}
\]

\[MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots = 0\]

- Computing MFP iteratively

MFP does not exist and is not computable
Computability of MFP

- If $f$ is not monotonic, the computation may not converge

\[
\begin{array}{c|ccccc}
  x & f(x) & f^2(x) & f^3(x) & f^4(x) & \ldots \\
  \hline
  1 & 0 & 1 & 0 & 1 & \ldots \\
\end{array}
\]

\[\text{MoP} = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots = 0\]

- Computing MFP iteratively

MFP does not exist and is not computable
Computability of MFP

• If $f$ is not monotonic, the computation may not converge

\[
\begin{array}{cccccc}
  x & f(x) & f^2(x) & f^3(x) & f^4(x) & \ldots \\
  1 & 0 & 1 & 0 & 1 & \ldots \\
\end{array}
\]

\[MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots = 0\]

• Computing MFP iteratively

MFP does not exist and is not computable
Computability of MFP

• If $f$ is not monotonic, the computation may not converge

\[
\begin{array}{lcccc}
  x & f(x) & f^2(x) & f^3(x) & f^4(x) & \ldots \\
  1 & 0 & 1 & 0 & 1 & \ldots \\
\end{array}
\]

\[
MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots = 0
\]

• Computing MFP iteratively

MFP does not exist and is not computable
Computability of MFP

- If $f$ is not monotonic, the computation may not converge

\[
\begin{array}{cccccc}
  x & f(x) & f^2(x) & f^3(x) & f^4(x) & \ldots \\
  1 & 0 & 1 & 0 & 1 & \ldots \\
\end{array}
\]

$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots = 0$$

- Computing MFP iteratively

MFP does not exist and is not computable

MFP exist and is computable
Computability of MFP

<table>
<thead>
<tr>
<th>⊑</th>
<th>≤</th>
<th>≤</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊓</td>
<td>min</td>
<td>min</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>-3</td>
<td>-3</td>
<td>-3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
</tr>
</tbody>
</table>

Hasse diagram

MFP exists?

MFP computable?

MoP exists?
### Computability of MFP

<table>
<thead>
<tr>
<th>$\subseteq$</th>
<th>$\leq$</th>
<th>$\leq$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\cap$</td>
<td>min</td>
<td>min</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-2</td>
<td>-3</td>
<td>-3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$-\infty$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Hasse diagram**

- MFP exists? No
- MFP computable? No
- MoP exists? No
### Computability of MFP

<table>
<thead>
<tr>
<th>$\subseteq$</th>
<th>$\leq$</th>
<th>$\leq$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\cap$</td>
<td>$\min$</td>
<td>$\min$</td>
</tr>
<tr>
<td>$x = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x = x - 1$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Hasse diagram

- $0$  $\rightarrow$  $-1$  $\rightarrow$  $-2$  $\rightarrow$  $-3$  $\rightarrow$  $\ldots$  $\rightarrow$  $-\infty$

#### Table

<table>
<thead>
<tr>
<th></th>
<th>MFP exists?</th>
<th>MFP computable?</th>
<th>MoP exists?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Point of interest
### Computability of MFP

<table>
<thead>
<tr>
<th>⊑</th>
<th>≤</th>
<th>≤</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊓</td>
<td>min</td>
<td>min</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td></td>
</tr>
</tbody>
</table>

- Flow functions are monotonic
- Strictly descending chains are not finite

<table>
<thead>
<tr>
<th>MFP exists?</th>
<th>No</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFP computable?</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>MoP exists?</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Existence and Computation of the Maximum Fixed Point

If $L$ is a meet semilattice satisfying DCC, $f : L \rightarrow L$ is monotonic, then

$MFP(f) = f^{k+1}(\top) = f^k(\top)$ such that $f^{j+1}(\top) \neq f^j(\top)$, $j < k$
Existence and Computation of the Maximum Fixed Point

If \( L \) is a meet semilattice satisfying DCC, \( f : L \rightarrow L \) is monotonic, then
\[
MFP(f) = f^{k+1}(\top) = f^k(\top)
\]
such that \( f^{j+1}(\top) \neq f^j(\top), \ j < k \)

Claims being made:

- \( \exists k \ s.t. \ f^{k+1}(\top) = f^k(\top) \)
- Since \( k \) is finite, \( f^k(\top) \) exists and is computable
- \( f^k(\top) \) is a fixed point
- \( f^k(\top) \) is a the maximum fixed point
Existence and Computation of the Maximum Fixed Point

If $L$ is a meet semilattice satisfying DCC, $f : L \rightarrow L$ is monotonic, then $MFP(f) = f^{k+1}() = f^k()$ such that $f^{j+1}(\top) \neq f^j(\top), j < k$

Claims being made:

- $\exists k$ s.t. $f^{k+1}(\top) = f^k(\top)$
- Since $k$ is finite, $f^k(\top)$ exists and is computable
- $f^k(\top)$ is a fixed point
- $f^k(\top)$ is a the maximum fixed point

The proof depends on:

- The existence of glb for every pair of values in $L$
- Finiteness of strictly descending chains
- Monotonicity of $f$
Existence and Computation of the Maximum Fixed Point

If $L$ is a meet semilattice satisfying DCC, $f : L \to L$ is monotonic, then $MFP(f) = f^{k+1}(\top) = f^k(\top)$ such that $f^{j+1}(\top) \neq f^j(\top)$, $j < k$. 

- Diagram: A diagram with a circle and a point labeled $\top$.
Existence and Computation of the Maximum Fixed Point

If $L$ is a meet semilattice satisfying DCC, $f : L \rightarrow L$ is monotonic, then $\text{MFP}(f) = f^{k+1}(\top) = f^k(\top)$ such that $f^{j+1}(\top) \neq f^j(\top)$, $j < k$
Existence and Computation of the Maximum Fixed Point

If \( L \) is a meet semilattice satisfying DCC, \( f : L \to L \) is monotonic, then

\[ MFP(f) = f^{k+1}(\top) = f^k(\top) \]

such that \( f^{j+1}(\top) \neq f^j(\top) \), \( j < k \)

- \( \top \sqsupseteq f(\top) \sqsupseteq f^2(\top) \sqsupseteq f^3(\top) \sqsupseteq f^4(\top) \sqsupseteq \ldots \)
Existence and Computation of the Maximum Fixed Point

If \( L \) is a meet semilattice satisfying DCC, \( f : L \rightarrow L \) is monotonic, then
\[
MFP(f) = f^{k+1}(\top) = f^k(\top)
\]
such that \( f^{j+1}(\top) \neq f^j(\top), \ j < k \)

- \( \top \supseteq f(\top) \supseteq f^2(\top) \supseteq f^3(\top) \supseteq f^4(\top) \supseteq \ldots \)

- Since strictly descending chains are finite, there must exist \( f^k(\top) \) such that
\( f^{k+1}(\top) = f^k(\top) \) and \( f^{j+1}(\top) \neq f^j(\top), \ j < k \)
**Existence and Computation of the Maximum Fixed Point**

If $L$ is a meet semilattice satisfying DCC, $f : L \rightarrow L$ is monotonic, then $MFP(f) = f^{k+1}(\top) = f^k(\top)$ such that $f^{j+1}(\top) \neq f^j(\top)$, $j < k$

- $\top \supseteq f(\top) \supseteq f^2(\top) \supseteq f^3(\top) \supseteq f^4(\top) \supseteq \ldots$
- Since strictly descending chains are finite, there must exist $f^k(\top)$ such that $f^{k+1}(\top) = f^k(\top)$ and $f^{j+1}(\top) \neq f^j(\top)$, $j < k$
- If $p$ is a fixed point of $f$ then $p \subseteq f^k(\top)$

Proof strategy: Induction on $i$ for $f^i(\top)$
Existence and Computation of the Maximum Fixed Point

If $L$ is a meet semilattice satisfying DCC, $f : L \rightarrow L$ is monotonic, then $MFP(f) = f^{k+1}(\top) = f^k(\top)$ such that $f^{j+1}(\top) \neq f^j(\top)$, $j < k$

- $\top \supseteq f(\top) \supseteq f^2(\top) \supseteq f^3(\top) \supseteq f^4(\top) \supseteq \ldots$
- Since strictly descending chains are finite, there must exist $f^k(\top)$ such that $f^{k+1}(\top) = f^k(\top)$ and $f^{j+1}(\top) \neq f^j(\top)$, $j < k$
- If $p$ is a fixed point of $f$ then $p \subseteq f^k(\top)$

Proof strategy: Induction on $i$ for $f^i(\top)$

- Basis ($i = 0$): $p \subseteq f^0(\top) = \top$
- Inductive Hypothesis: Assume that $p \subseteq f^i(\top)$
Existence and Computation of the Maximum Fixed Point

If $L$ is a meet semilattice satisfying DCC, $f : L \to L$ is monotonic, then $MFP(f) = f^{k+1}(\top) = f^k(\top)$ such that $f^{j+1}(\top) \neq f^j(\top)$, $j < k$

- $\top \supseteq f(\top) \supseteq f^2(\top) \supseteq f^3(\top) \supseteq f^4(\top) \supseteq \ldots$
- Since strictly descending chains are finite, there must exist $f^k(\top)$ such that $f^{k+1}(\top) = f^k(\top)$ and $f^{j+1}(\top) \neq f^j(\top)$, $j < k$
- If $p$ is a fixed point of $f$ then $p \sqsubseteq f^k(\top)$

Proof strategy: Induction on $i$ for $f^i(\top)$

- Basis ($i = 0$): $p \sqsubseteq f^0(\top) = \top$
- Inductive Hypothesis: Assume that $p \sqsubseteq f^i(\top)$
- Proof: $f(p) \sqsubseteq f(f^i(\top))$ ($f$ is monotonic)
  \[ \Rightarrow p \sqsubseteq f(f^i(\top)) \quad (f(p) = p) \]
  \[ \Rightarrow p \sqsubseteq f^{i+1}(\top) \]
Existence and Computation of the Maximum Fixed Point

If \( L \) is a meet semilattice satisfying DCC, \( f : L \to L \) is monotonic, then
\[
MFP(f) = f^{k+1}(\top) = f^k(\top)
\]
such that \( f^{j+1}(\top) \neq f^j(\top), j < k \)

- \( \top \supseteq f(\top) \supseteq f^2(\top) \supseteq f^3(\top) \supseteq f^4(\top) \supseteq \ldots \)
- Since strictly descending chains are finite, there must exist \( f^k(\top) \) such that \( f^{k+1}(\top) = f^k(\top) \) and \( f^{j+1}(\top) \neq f^j(\top), j < k \)
- If \( p \) is a fixed point of \( f \) then \( p \sqsubseteq f^k(\top) \)

Proof strategy: Induction on \( i \) for \( f^i(\top) \)

- Basis \((i = 0)\): \( p \sqsubseteq f^0(\top) = \top \)
- Inductive Hypothesis: Assume that \( p \sqsubseteq f^i(\top) \)
- Proof: \( f(p) \sqsubseteq f(f^i(\top)) \) (\( f \) is monotonic)
  \[
  \Rightarrow p \sqsubseteq f(f^i(\top)) \quad (f(p) = p)
  \]
\[
  \Rightarrow p \sqsubseteq f^{i+1}(\top)
  \]
- Since this holds for every \( p \) that is a fixed point, \( f^{k+1}(\top) \) must be the Maximum Fixed Point
• Recall that

\[ MFP(f) = f^{k+1}(\top) = f^k(\top) \text{ such that } f^{j+1}(\top) \neq f^j(\top), \ j < k. \]
• Recall that

\[
MFP(f) = f^{k+1}(\top) = f^k(\top) \text{ such that } f^{j+1}(\top) \neq f^j(\top), \ j < k.
\]

▷ What is \( f \) in the above?
Fixed Points Computation: Flow Functions Vs. Equations

• Recall that

\[ MFP(f) = f^{k+1}(\top) = f^k(\top) \text{ such that } f^{j+1}(\top) \neq f^j(\top), j < k. \]

▷ What is \( f \) in the above?
▷ Flow function of a block? Which block?
Recall that

\[ MFP(f) = f^{k+1}(\top) = f^k(\top) \text{ such that } f^{j+1}(\top) \neq f^j(\top), \quad j < k. \]

- What is \( f \) in the above?
- Flow function of a block? Which block?

- Our method computes the maximum fixed point of data flow equations!
Fixed Points Computation: Flow Functions Vs. Equations

- Recall that

\[ MFP(f) = f^{k+1}(\top) = f^k(\top) \text{ such that } f^{j+1}(\top) \neq f^j(\top), \ j < k. \]

- What is \( f \) in the above?
- Flow function of a block? Which block?

- Our method computes the maximum fixed point of data flow equations!

- What is the relation between the maximum fixed point of data flow equations and the MFP defined above?
Fixed Points Computation: Flow Functions Vs. Equations

• Data flow equations for a CFG with $N$ nodes can be written as

\[
\begin{align*}
In_1 &= Bl \\
Out_1 &= f_1(In_1) \\
In_2 &= Out_1 \sqcup \ldots \\
Out_2 &= f_2(In_2) \\
&\vdots \\
In_N &= Out_{N-1} \sqcup \ldots \\
Out_N &= f_N(In_N)
\end{align*}
\]
Fixed Points Computation: Flow Functions Vs. Equations

- Data flow equations for a CFG with N nodes can be written as

\[
\begin{align*}
    \text{In}_1 &= f_{\text{In}_1}(\langle \text{In}_1, \text{Out}_1, \ldots, \text{In}_N, \text{Out}_N \rangle) \\
    \text{Out}_1 &= f_{\text{Out}_1}(\langle \text{In}_1, \text{Out}_1, \ldots, \text{In}_N, \text{Out}_N \rangle) \\
    \text{In}_2 &= f_{\text{In}_2}(\langle \text{In}_1, \text{Out}_1, \ldots, \text{In}_N, \text{Out}_N \rangle) \\
    \text{Out}_2 &= f_{\text{Out}_2}(\langle \text{In}_1, \text{Out}_1, \ldots, \text{In}_N, \text{Out}_N \rangle) \\
    \vdots
    \text{In}_N &= f_{\text{In}_N}(\langle \text{In}_1, \text{Out}_1, \ldots, \text{In}_N, \text{Out}_N \rangle) \\
    \text{Out}_N &= f_{\text{Out}_N}(\langle \text{In}_1, \text{Out}_1, \ldots, \text{In}_N, \text{Out}_N \rangle)
\end{align*}
\]

where each flow function is of the form \(L \times L \times \ldots \times L \rightarrow L\)
Fixed Points Computation: Flow Functions Vs. Equations

- Data flow equations for a CFG with \( N \) nodes can be written as

\[
\langle \text{In}_1, \text{Out}_1, \ldots, \text{In}_N, \text{Out}_N \rangle = \langle f_{\text{In}_1}(\langle \text{In}_1, \text{Out}_1, \ldots, \text{In}_N, \text{Out}_N \rangle), f_{\text{Out}_1}(\langle \text{In}_1, \text{Out}_1, \ldots, \text{In}_N, \text{Out}_N \rangle), \ldots, f_{\text{In}_N}(\langle \text{In}_1, \text{Out}_1, \ldots, \text{In}_N, \text{Out}_N \rangle), f_{\text{Out}_N}(\langle \text{In}_1, \text{Out}_1, \ldots, \text{In}_N, \text{Out}_N \rangle) \rangle
\]

where each flow function is of the form \( L \times L \times \ldots \times L \to L \)
Fixed Points Computation: Flow Functions Vs. Equations

• Data flow equations for a CFG with N nodes can be written as

\[ \mathcal{X} = \langle f_{\text{In}_1}(\mathcal{X}), f_{\text{Out}_1}(\mathcal{X}), \ldots, f_{\text{In}_N}(\mathcal{X}), f_{\text{Out}_N}(\mathcal{X}) \rangle \]

where \( \mathcal{X} = \langle \text{In}_1, \text{Out}_1, \ldots, \text{In}_N, \text{Out}_N \rangle \)
Fixed Points Computation: Flow Functions Vs. Equations

- Data flow equations for a CFG with N nodes can be written as

\[ \mathcal{X} = \mathcal{F}(\mathcal{X}) \]

where

\[ \mathcal{X} = \langle \text{In}_1, \text{Out}_1, \ldots, \text{In}_N, \text{Out}_N \rangle \]

\[ \mathcal{F}(\mathcal{X}) = \langle f_{\text{In}_1}(\mathcal{X}), f_{\text{Out}_1}(\mathcal{X}), \ldots, f_{\text{In}_N}(\mathcal{X}), f_{\text{Out}_N}(\mathcal{X}) \rangle \]
Fixed Points Computation: Flow Functions Vs. Equations

- Data flow equations for a CFG with N nodes can be written as

\[ X = F(X) \]

where

\[ X = \langle \text{In}_1, \text{Out}_1, \ldots, \text{In}_N, \text{Out}_N \rangle \]

\[ F(X) = \langle f_{\text{in}_1}(X), f_{\text{out}_1}(X), \ldots, f_{\text{in}_N}(X), f_{\text{out}_N}(X) \rangle \]

We compute the fixed points of function \( F \) defined above.
An Instance of Available Expressions Analysis

- Conventional data flow equations

\[ \text{In}_1 = 00 \]
\[ \text{Out}_1 = 11 \]
An Instance of Available Expressions Analysis

- Conventional data flow equations

\[ \text{In}_1 = 00 \]
\[ \text{Out}_1 = 11 \]
\[ \text{In}_2 = \text{Out}_1 \cap \text{Out}_2 \]
\[ \text{Out}_2 = \text{In}_2 \]
An Instance of Available Expressions Analysis

- Conventional data flow equations

\[
\begin{align*}
\text{In}_1 &= 00 \\
\text{In}_2 &= \text{Out}_1 \cap \text{Out}_2 \\
\text{Out}_1 &= 11 \\
\text{Out}_2 &= \text{In}_2
\end{align*}
\]

- Data Flow Equation \( \mathcal{X} = \mathcal{F}(\mathcal{X}) \) is

\[
\mathcal{F}(⟨\text{In}_1, \text{Out}_1, \text{In}_2, \text{Out}_2⟩) = ⟨00, 11, \text{Out}_1 \cap \text{Out}_2, \text{In}_2⟩
\]
An Instance of Available Expressions Analysis

- Conventional data flow equations

\[
\begin{align*}
In_1 &= 00 & In_2 &= Out_1 \cap Out_2 \\
Out_1 &= 11 & Out_2 &= In_2
\end{align*}
\]

- Data Flow Equation \( \mathcal{X} = \mathcal{F}(\mathcal{X}) \) is

\[
\mathcal{F}(\langle In_1, Out_1, In_2, Out_2 \rangle) = \langle 00, 11, Out_1 \cap Out_2, In_2 \rangle
\]

- The maximum fixed point assignment is

\[
\mathcal{F}(\langle 11, 11, 11, 11 \rangle) = \langle 00, 11, 11, 11 \rangle
\]
An Instance of Available Expressions Analysis

- Conventional data flow equations

\[
\begin{align*}
In_1 &= 00 & In_2 &= Out_1 \cap Out_2 \\
Out_1 &= 11 & Out_2 &= In_2
\end{align*}
\]

- Data Flow Equation \( \mathcal{X} = \mathcal{F}(\mathcal{X}) \) is

\[
\mathcal{F}(\langle In_1, Out_1, In_2, Out_2 \rangle) = \langle 00, 11, Out_1 \cap Out_2, In_2 \rangle
\]

- The maximum fixed point assignment is

\[
\mathcal{F}(\langle 11, 11, 11, 11 \rangle) = \langle 00, 11, 11, 11 \rangle
\]

- The minimum fixed point assignment is

\[
\mathcal{F}(\langle 00, 00, 00, 00 \rangle) = \langle 00, 11, 00, 00 \rangle
\]
Safety of FP Assignment: \( FP \subseteq MoP \)
Safety of FP Assignment: $FP \subseteq MoP$

- $MoP(v) = \bigcap_{\rho \in Paths(v)} f_{\rho}(BI)$
Safety of FP Assignment: $FP \subseteq MoP$

- $MoP(v) = \bigcap_{\rho \in Paths(v)} f_{\rho}(BI)$

- Proof Obligation: $\forall \rho_v FP(v) \subseteq f_{\rho_v}(BI)$
Safety of FP Assignment: \( FP \subseteq MoP \)

- \( MoP(v) = \bigwedge_{\rho \in \text{Paths}(v)} f_\rho(BI) \)
- Proof Obligation: \( \forall \rho_v \ FP(v) \subseteq f_\rho_v(BI) \)
- Claim 1: \( \forall u \rightarrow v, \ FP(v) \subseteq f_{u \rightarrow v}(FP(u)) \)
Safety of FP Assignment: \( FP \subseteq MoP \)

- \( MoP(v) = \bigcap_{\rho \in Paths(v)} f_{\rho}(BI) \)
- Proof Obligation: \( \forall \rho_v \ FP(v) \subseteq f_{\rho_v}(BI) \)
- Claim 1: \( \forall u \rightarrow v, FP(v) \subseteq f_{u \rightarrow v}(FP(u)) \)
- Proof Outline: Induction on path length
  - Base case: Path of length 0
    \[ FP(Entry) = MoP(Entry) = BI \]
  - Inductive hypothesis: Assume it holds for paths consisting of \( k \) edges (say at \( u \))
    \[ FP(u) \subseteq f_{\rho_u}(BI) \] (Inductive hypothesis)
    \[ FP(v) \subseteq f_{u \rightarrow v}(FP(u)) \] (Claim 1)
    \[ \Rightarrow FP(v) \subseteq f_{u \rightarrow v}(f_{\rho_u}(BI)) \]
    \[ \Rightarrow FP(v) \subseteq f_{\rho_v}(BI) \]
  - This holds for every \( FP \) and hence for \( MFP \) also
Part 8

Theoretical Abstractions: A Summary
Theoretical Abstractions: A Summary

Necessary and sufficient conditions for designing a data flow framework
Theoretical Abstractions: A Summary

Necessary and sufficient conditions for designing a data flow framework

- A meet semilattice satisfying dcc
Theoretical Abstractions: A Summary

Necessary and sufficient conditions for designing a data flow framework

- A meet semilattice satisfying dcc

- A function space
  - Monotonic functions
Necessary and sufficient conditions for designing a data flow framework

- A meet semilattice satisfying dcc
  - Meet: commutative, associative, and idempotent
  - Partial order: reflexive, transitive, and antisymmetric
  - Existence of $\perp$

- A function space
  - Monotonic functions
Theoretical Abstractions: A Summary

Necessary and sufficient conditions for designing a data flow framework

- A meet semilattice satisfying dcc
  - Meet: commutative, associative, and idempotent
  - Partial order: reflexive, transitive, and antisymmetric
  - Existence of $\bot$

- A function space
  - Existence of the identity function
  - Closure under composition
  - Monotonic functions
Part 9

Performing Data Flow Analysis
Performing Data Flow Analysis

- Algorithms for computing MFP solution
- Complexity of data flow analysis
- Factor affecting the complexity of data flow analysis
Iterative Methods of Performing Data Flow Analysis

Successive recomputation after conservative initialization (⊤)

- **Round Robin.** Repeated traversals over nodes in a fixed order

  **Termination:** After values stabilise
  
  + Simplest to understand and implement
  
  – May perform unnecessary computations
Iterative Methods of Performing Data Flow Analysis

Successive recomputation after conservative initialization ($\top$)

- **Round Robin.** Repeated traversals over nodes in a fixed order

  Termination: After values stabilise
  
  + Simplest to understand and implement
  
  - May perform unnecessary computations

Our examples use this method.
Iterative Methods of Performing Data Flow Analysis

Successive recomputation after conservative initialization ($\top$)

- **Round Robin.** Repeated traversals over nodes in a fixed order
  
  Termination: After values stabilise
  
  - Simplest to understand and implement
  - May perform unnecessary computations

- **Work List.** Dynamic list of nodes which need recomputation
  
  Termination: When the list becomes empty
  
  - Demand driven. Avoid unnecessary computations
  - Overheads of maintaining work list

Our examples use this method
Elimination Methods of Performing Data Flow Analysis

Delayed computations of dependent data flow values of dependent nodes

Find suitable single-entry regions

- *Interval Based Analysis*. Uses graph partitioning
- \( T_1, T_2 \) *Based Analysis*. Uses graph parsing
Classification of Edges in a Graph

Graph $G$

1 – 2 – 3 – 4 – 5 – 6 – 7 – 8
Classification of Edges in a Graph

Graph $G$

A depth first spanning tree of $G$
Classification of Edges in a Graph

Graph $G$

A depth first spanning tree of $G$

Back edges
Forward edges
Tree edges
Cross edges
Classification of Edges in a Graph

For data flow analysis, we club tree, forward, and cross edges into forward edges. Thus we have just forward or back edges in a control flow graph.
Reverse Post Order Traversal

- A reverse post order (rpo) is a topological sort of the graph obtained after removing back edges.

- Some possible RPOs for $G$ are: $(1, 2, 3, 4, 5, 6, 7, 8)$, $(1, 6, 7, 2, 3, 4, 5, 8)$, $(1, 6, 2, 7, 4, 3, 5, 8)$, and $(1, 2, 6, 7, 3, 4, 5, 8)$.
Round Robin Iterative Algorithm

\begin{algorithm}
\begin{algorithmic}
\STATE $l_{n_0} = B l$
\FORALL {$j \neq 0$}
\STATE $l_{n_j} = \top$
\STATE $\text{change} = \text{true}$
\WHILE {$\text{change}$}
\STATE $\text{change} = \text{false}$
\FOR {$j = 1$ to $N - 1$}
\STATE $\text{temp} = \bigcap_{p \in \text{pred}(j)} f_p(l_{n_p})$
\IF {$\text{temp} \neq l_{n_j}$}
\STATE $l_{n_j} = \text{temp}$
\STATE $\text{change} = \text{true}$
\ENDIF
\ENDFOR
\ENDWHILE
\end{algorithmic}
\end{algorithm}

Round Robin Iterative Algorithm

1. \( I_{n_0} = B I \)
2. \textbf{for all } \( j \neq 0 \) \textbf{do}
3. \hspace{1em} \( I_{n_j} = \top \)
4. \hspace{1em} \textit{change} = \textit{true}
5. \hspace{1em} \textbf{while } \textit{change} \textbf{ do}
6. \hspace{2em} \{ \hspace{1em} \textit{change} = \textit{false}
7. \hspace{3em} \textbf{for } j = 1 \text{ to } N - 1 \textbf{ do}
8. \hspace{4em} \{ \hspace{1em} \textit{temp} = \prod_{p \in \text{pred}(j)} f_p(I_{n_p})
9. \hspace{5em} \textbf{if } \textit{temp} \neq I_{n_j} \textbf{ then}
10. \hspace{6em} \{ \hspace{1em} I_{n_j} = \textit{temp}
11. \hspace{7em} \textit{change} = \textit{true}
12. \hspace{6em} \}
13. \hspace{4em} \}
14. \}

- Computation of \( O_{u_j} \) has been left implicit

Works fine for unidirectional frameworks
Round Robin Iterative Algorithm

1. \( I_{n_0} = BI \)
2. for all \( j \neq 0 \) do
3. \( I_{n_j} = \top \)
4. \( \text{change} = \text{true} \)
5. while \( \text{change} \) do
6. \{ \( \text{change} = \text{false} \)
7. for \( j = 1 \) to \( N - 1 \) do
8. \{ \( \text{temp} = \bigcap_{p \in \text{pred}(j)} f_p(I_{n_p}) \)
9. if \( \text{temp} \neq I_{n_j} \) then
10. \{ \( I_{n_j} = \text{temp} \)
11. \( \text{change} = \text{true} \)
12. \}
13. \}
14. \}

- Computation of \( Out_j \) has been left implicit
  Works fine for unidirectional frameworks
- \( \top \) is the identity of \( \sqcap \)
  (line 3)
Round Robin Iterative Algorithm

\begin{verbatim}
1 \text{In}_0 = BI
2 \text{for all } j \neq 0 \text{ do}
3 \quad \text{In}_j = \top
4 \quad \text{change} = \text{true}
5 \text{while change do}
6 \quad \{ \text{change} = \text{false}
7 \quad \text{for } j = 1 \text{ to } N - 1 \text{ do}
8 \quad \{ \text{temp} = \bigwedge_{p \in \text{pred}(j)} f_p(\text{In}_p)
9 \quad \text{if temp} \neq \text{In}_j \text{ then}
10 \quad \{ \text{In}_j = \text{temp}
11 \quad \text{change} = \text{true}
12 \quad \}
13 \}
14 \}
\end{verbatim}

- Computation of $Out_j$ has been left implicit
  - Works fine for unidirectional frameworks
- $\top$ is the identity of $\sqcap$
  - (line 3)
- Reverse postorder (rpo) traversal for efficiency
  - (line 7)
Round Robin Iterative Algorithm

1. $I_{n_0} = BI$
2. **for** all $j \neq 0$ **do**
   3. $I_{n_j} = \top$
   4. $change = true$
5. **while** change **do**
   6. { change = false
   7.     **for** $j = 1$ to $N - 1$ **do**
       8.         { temp = $\bigcap_{p \in \text{pred}(j)} f_p(I_{n_p})$
           9.             **if** temp $\neq I_{n_j}$ **then**
               10.                 { $I_{n_j} = \text{temp}$
                   11.                   change = true
                   12.                 }
           13.         }
        14. }

- Computation of $Out_j$ has been left implicit
  Works fine for unidirectional frameworks
- $\top$ is the identity of $\sqcap$ (line 3)
- Reverse postorder (rpo) traversal for efficiency (line 7)
- rpo traversal AND no loops $\Rightarrow$ no need of initialization
Complexity of Round Robin Iterative Algorithm

- Unidirectional bit vector frameworks
  - Construct a spanning tree $T$ of $G$ to identify postorder traversal
  - Traverse $G$ in reverse postorder for forward problems and Traverse $G$ in postorder for backward problems
  - Depth $d(G, T)$: Maximum number of back edges in any acyclic path

<table>
<thead>
<tr>
<th>Task</th>
<th>Number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>First computation of $In$ and $Out$</td>
<td>1</td>
</tr>
<tr>
<td>Convergence</td>
<td>$d(G, T)$</td>
</tr>
<tr>
<td>(until change remains true)</td>
<td></td>
</tr>
<tr>
<td>Verifying convergence</td>
<td>1</td>
</tr>
<tr>
<td>(change becomes false)</td>
<td></td>
</tr>
</tbody>
</table>
Complexity of Round Robin Iterative Algorithm

- Unidirectional bit vector frameworks
  - Construct a spanning tree $T$ of $G$ to identify postorder traversal
  - Traverse $G$ in reverse postorder for forward problems and Traverse $G$ in postorder for backward problems
  - Depth $d(G, T)$: Maximum number of back edges in any acyclic path

<table>
<thead>
<tr>
<th>Task</th>
<th>Number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>First computation of $In$ and $Out$</td>
<td>1</td>
</tr>
<tr>
<td>Convergence (until $change$ remains true)</td>
<td>$d(G, T)$</td>
</tr>
<tr>
<td>Verifying convergence ($change$ becomes false)</td>
<td>1</td>
</tr>
</tbody>
</table>

- What about bidirectional bit vector frameworks?
Complexity of Round Robin Iterative Algorithm

- Unidirectional bit vector frameworks
  - Construct a spanning tree $T$ of $G$ to identify postorder traversal
  - Traverse $G$ in reverse postorder for forward problems and Traverse $G$ in postorder for backward problems
  - Depth $d(G, T)$: Maximum number of back edges in any acyclic path

<table>
<thead>
<tr>
<th>Task</th>
<th>Number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>First computation of $In$ and $Out$</td>
<td>1</td>
</tr>
<tr>
<td>Convergence (until $change$ remains true)</td>
<td>$d(G, T)$</td>
</tr>
<tr>
<td>Verifying convergence ($change$ becomes false)</td>
<td>1</td>
</tr>
</tbody>
</table>

- What about bidirectional bit vector frameworks?
- What about other frameworks?
Example C Program with $d(G,T) = 2$

```c
void fun(int m, int n)
{
    int i,j,a,b,c;
    c=a+b;
    i=0;
    while(i<m)
    {
        j=0;
        while(j<n)
        {
            a=i+j;
            j=j+1;
        }
        i=i+1;
    }
}
```
Example C Program with \( d(G,T) = 2 \)

```c
void fun(int m, int n)
{
    int i, j, a, b, c;
    c = a + b;
    i = 0;
    while (i < m)
    {
        j = 0;
        while (j < n)
        {
            a = i + j;
            j = j + 1;
        }
        i = i + 1;
    }
}
```
Example C Program with $d(G,T) = 2$

```c
void fun(int m, int n)
{
    int i,j,a,b,c;
    c=a+b;
    i=0;
    while(i<m)
    {
        j=0;
        while(j<n)
        {
            a=i+j;
            j=j+1;
        }
        i=i+1;
    }
}
```

Availability of $a+b$ in iteration #1
Example C Program with $d(G,T) = 2$

1. void fun(int m, int n)
2. {
3.     int i,j,a,b,c;
4.     c=a+b;
5.     i=0;
6.     while(i<m)
7.     {
8.         j=0;
9.         while(j<n)
10.        {
11.             a=i+j;
12.             j=j+1;
13.         }
14.         i=i+1;
15.     }
16. }

Availability of $a+b$ in iteration #2
Example C Program with \( d(G,T) = 2 \)

```c
1  void fun(int m, int n)  
2  {
3      int i,j,a,b,c;
4      c=a+b;
5      i=0;
6      while(i<m)
7         {
8             j=0;
9             while(j<n)
10                {
11                    a=i+j;
12                    j=j+1;
13                }
14             i=i+1;
15         }
16    }
```

Availability of \( a+b \) in iteration #3
Example C Program with $d(G,T) = 2$

```c
1 void fun(int m, int n)
2 {
3     int i, j, a, b, c;
4     c = a + b;
5     i = 0;
6     while (i < m)
7         {
8             j = 0;
9             while (j < n)
10                {
11                 a = i + j;
12                 j = j + 1;
13             }
14             i = i + 1;
15         }
16 }
```

Availability of $a + b$ in iteration #4

3 + 1 iterations for available expressions analysis
Example C Program with $d(G,T) = 2$

```c
void fun(int m, int n) {
    int i, j, a, b, c;
    c = a + b;
    i = 0;
    while (i < m) {
        j = 0;
        while (j < n) {
            a = i + j;
            j = j + 1;
        }
        i = i + 1;
    }
}
```

Diagram:
- `c = a + b` (node $n_1$)
- `i = 0` (node $n_2$)
- `if (i < m)` (node $n_3$)
- `j = 0` (node $n_4$)
- `if (j < n)` (node $n_5$)
- `a = i + j` (node $n_6$)
- `j = j + 1` (node $n_7$)
- `i = i + 1` (node $n_8$)
```c
void fun(int m, int n) {
    int i,j,a,b,c;
    c=a+b;
    i=0;
    while(i<m) {
        j=0;
        while(j<n) {
            a=i+j;
            j=j+1;
        }
        i=i+1;
    }
}
```
Example C Program with $d(G,T) = 2$

```c
void fun(int m, int n)
{
    int i, j, a, b, c;
    c = a + b;
    i = 0;
    while (i < m)
    {
        j = 0;
        while (j < n)
        {
            a = i + j;
            j = j + 1;
        }
        i = i + 1;
    }
}
```
Example: Consider the following CFG for PRE

```
1
  ^
 /  \
2    6
  |  /
  | \
3    7
  |  /
  | \
4    8
  |  /
  | \
5    9
  |  /
  | \
  | 10
  |  /
  | \
  | 11
  |  /
  | \
  | 12
```
Example: Consider the following CFG for PRE

- Node numbers are in reverse post order
Example: Consider the following CFG for PRE

- Node numbers are in reverse post order
- Back edges in the graph are $n_5 \rightarrow n_2$ and $n_{10} \rightarrow n_9$
Complexity of Bidirectional Bit Vector Frameworks

Example: Consider the following CFG for PRE

- Node numbers are in reverse post order
- Back edges in the graph are $n_5 \rightarrow n_2$ and $n_{10} \rightarrow n_9$
- $d(G, T) = 1$
Example: Consider the following CFG for PRE

![CFG Diagram]

- Node numbers are in reverse post order
- Back edges in the graph are $n_5 \rightarrow n_2$ and $n_{10} \rightarrow n_9$
- $d(G, T) = 1$
- Actual iterations: 5
Complexity of Bidirectional Bit Vector Frameworks

![Graph Diagram]

<table>
<thead>
<tr>
<th>Initialization</th>
<th>Changes in Iterations</th>
<th>Final values &amp; transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>O,I</td>
<td>#1</td>
<td>O,I</td>
</tr>
<tr>
<td>12</td>
<td>0,1</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1,1</td>
<td></td>
</tr>
</tbody>
</table>
Complexity of Bidirectional Bit Vector Frameworks

Pairs of Out, In Values

<table>
<thead>
<tr>
<th>Initialization</th>
<th>Changes in Iterations</th>
<th>Final values &amp; transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#1</td>
<td>#2</td>
</tr>
<tr>
<td>O,1</td>
<td>O,1</td>
<td>O,1</td>
</tr>
<tr>
<td>12</td>
<td>0,1</td>
<td>0,0</td>
</tr>
<tr>
<td>11</td>
<td>1,1</td>
<td>0,1</td>
</tr>
<tr>
<td>10</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1,1</td>
<td>1,0</td>
</tr>
<tr>
<td>5</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1,1</td>
<td>0,0</td>
</tr>
</tbody>
</table>
Complexity of Bidirectional Bit Vector Frameworks

Pairs of $Out, In$ Values

<table>
<thead>
<tr>
<th>Initialization</th>
<th>Changes in Iterations</th>
<th>Final values &amp; transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#1</td>
<td>#2</td>
</tr>
<tr>
<td>12</td>
<td>0,1</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1,1</td>
<td>1,0</td>
</tr>
<tr>
<td>5</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1,1</td>
<td>1,0</td>
</tr>
<tr>
<td>1</td>
<td>1,1</td>
<td>0,0</td>
</tr>
</tbody>
</table>

Diagram:

- Node 1
- Node 6: $b \times c$
- Node 2
- Node 7
- Node 3
- Node 4
- Node 5
- Node 8
- Node 9
- Node 10: $b \times c$
- Node 11
- Node 12
Complexity of Bidirectional Bit Vector Frameworks

Pairs of $Out, In$ Values

<table>
<thead>
<tr>
<th>Initialization</th>
<th>Changes in Iterations</th>
<th>Final values &amp; transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#1</td>
<td>#2</td>
</tr>
<tr>
<td>O,1</td>
<td>O,1</td>
<td>O,1</td>
</tr>
<tr>
<td>12</td>
<td>0,1</td>
<td>0,0</td>
</tr>
<tr>
<td>11</td>
<td>1,1</td>
<td>0,1</td>
</tr>
<tr>
<td>10</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1,1</td>
<td>1,0</td>
</tr>
<tr>
<td>5</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1,1</td>
<td>1,0</td>
</tr>
<tr>
<td>1</td>
<td>1,1</td>
<td></td>
</tr>
</tbody>
</table>
Complexity of Bidirectional Bit Vector Frameworks

Pairs of $Out, In$ Values

<table>
<thead>
<tr>
<th>Initialization</th>
<th>Changes in Iterations</th>
<th>Final values &amp; transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#1</td>
<td>#2</td>
</tr>
<tr>
<td>0,1</td>
<td>0,1</td>
<td>0,1</td>
</tr>
<tr>
<td>12</td>
<td>0,1</td>
<td>0,0</td>
</tr>
<tr>
<td>11</td>
<td>1,1</td>
<td>0,1</td>
</tr>
<tr>
<td>10</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1,1</td>
<td>1,0</td>
</tr>
<tr>
<td>5</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1,1</td>
<td>1,0</td>
</tr>
<tr>
<td>1</td>
<td>1,1</td>
<td>0,0</td>
</tr>
</tbody>
</table>
Complexity of Bidirectional Bit Vector Frameworks

Pairs of $Out, In$ Values

<table>
<thead>
<tr>
<th>Initialization</th>
<th>Changes in Iterations</th>
<th>Final values &amp; transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#1</td>
<td>#2</td>
</tr>
<tr>
<td></td>
<td>$O,1$</td>
<td>$O,1$</td>
</tr>
<tr>
<td>12</td>
<td>$0,1$</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>$1,1$</td>
<td>$0,1$</td>
</tr>
<tr>
<td>10</td>
<td>$1,1$</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>$1,1$</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>$1,1$</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>$1,1$</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>$1,1$</td>
<td>$1,0$</td>
</tr>
<tr>
<td>5</td>
<td>$1,1$</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$1,1$</td>
<td>$0,1$</td>
</tr>
<tr>
<td>3</td>
<td>$1,1$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$1,1$</td>
<td>$1,0$</td>
</tr>
<tr>
<td>1</td>
<td>$1,1$</td>
<td>$0,0$</td>
</tr>
</tbody>
</table>
Complexity of Bidirectional Bit Vector Frameworks

Pairs of $Out, In$ Values

<table>
<thead>
<tr>
<th>Initialization</th>
<th>Changes in Iterations #1</th>
<th>Changes in Iterations #2</th>
<th>Changes in Iterations #3</th>
<th>Changes in Iterations #4</th>
<th>Changes in Iterations #5</th>
<th>Final values &amp; transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0,1$</td>
<td>$0,1$</td>
<td>$0,1$</td>
<td>$0,1$</td>
<td>$0,1$</td>
<td>$0,1$</td>
<td>$0,1$</td>
</tr>
<tr>
<td>$0,1$</td>
<td>$0,0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$0,0$</td>
</tr>
<tr>
<td>$1,1$</td>
<td>$0,1$</td>
<td>$0,0$</td>
<td></td>
<td></td>
<td></td>
<td>$0,0$</td>
</tr>
<tr>
<td>$1,1$</td>
<td>$0,1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$0,1$</td>
</tr>
<tr>
<td>$1,1$</td>
<td></td>
<td></td>
<td>$1,0$</td>
<td></td>
<td></td>
<td>$1,0$</td>
</tr>
<tr>
<td>$1,1$</td>
<td></td>
<td></td>
<td></td>
<td>$1,0$</td>
<td>$1,0$</td>
<td>$1,0$</td>
</tr>
<tr>
<td>$1,1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$0,0$</td>
<td>$0,0$</td>
</tr>
<tr>
<td>$1,1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$0,0$</td>
</tr>
<tr>
<td>$1,1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$0,0$</td>
</tr>
<tr>
<td>$1,1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$0,0$</td>
</tr>
<tr>
<td>$1,1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$0,0$</td>
</tr>
<tr>
<td>$1,1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$0,0$</td>
</tr>
<tr>
<td>$1,1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$0,0$</td>
</tr>
</tbody>
</table>
Complexity of Bidirectional Bit Vector Frameworks

Pairs of Out, In Values

<table>
<thead>
<tr>
<th>Initialization</th>
<th>Changes in Iterations</th>
<th>Final values &amp; transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>#2</td>
<td>#3</td>
</tr>
<tr>
<td>O,1</td>
<td>O,1</td>
<td>O,1</td>
</tr>
<tr>
<td>0,1</td>
<td>0,0</td>
<td>0,0</td>
</tr>
<tr>
<td>1,1</td>
<td>0,1</td>
<td>0,0</td>
</tr>
<tr>
<td>1,1</td>
<td>1,0</td>
<td>1,0</td>
</tr>
<tr>
<td>1,1</td>
<td>0,0</td>
<td>0,0</td>
</tr>
<tr>
<td>1,1</td>
<td>0,0</td>
<td>0,0</td>
</tr>
<tr>
<td>1,1</td>
<td>1,0</td>
<td>0,0</td>
</tr>
<tr>
<td>1,1</td>
<td>0,0</td>
<td>0,0</td>
</tr>
<tr>
<td>1,1</td>
<td>0,0</td>
<td>0,0</td>
</tr>
</tbody>
</table>

Aug 2017 IIT Bombay
An Example of Information Flow in Our PRE Analysis

- $P_{avln_6}$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)
An Example of Information Flow in Our PRE Analysis

- $PavlIn_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)
An Example of Information Flow in Our PRE Analysis

- $Pavl_{n_6}$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)
An Example of Information Flow in Our PRE Analysis

- $PavIn_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)
An Example of Information Flow in Our PRE Analysis

- $PavIn_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)
An Example of Information Flow in Our PRE Analysis

- \( \text{PavIn}_6 \) becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)
An Example of Information Flow in Our PRE Analysis

- $Pavl_{ln_6}$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)
An Example of Information Flow in Our PRE Analysis

- $P_{avl\ln_i}$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)
An Example of Information Flow in Our PRE Analysis

- $PavIn_6$ becomes 0 in the first iteration
- This causes many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)
Information Flow and Information Flow Paths

- Default value at each program point: ⊤
- *Information flow path*
Information Flow and Information Flow Paths

- Default value at each program point: $\top$
- Information flow path
  
  Sequence of adjacent program points
Information Flow and Information Flow Paths

- Default value at each program point: $\top$

- *Information flow path*

  Sequence of adjacent program points along which data flow values change
Information Flow and Information Flow Paths

- Default value at each program point: $\top$

- Information flow path
  
  Sequence of adjacent program points along which data flow values change

- A change in the data flow at a program point could be
  
  - Generation of information
    Change from $\top$ to a non-$\top$ due to local effect (i.e. $f(\top) \neq \top$)
  
  - Propagation of information
    Change from $x$ to $y$ such that $y \sqsubseteq x$ due to global effect
Information Flow and Information Flow Paths

- Default value at each program point: $\top$
- *Information flow path*
  
  Sequence of adjacent program points along which data flow values change
- A change in the data flow at a program point could be
  
  - *Generation of information*
    Change from $\top$ to a non-$\top$ due to local effect (i.e. $f(\top) \neq \top$)
  
  - *Propagation of information*
    Change from $x$ to $y$ such that $y \sqsupseteq x$ due to global effect
- Information flow path (ifp) need not be a graph theoretic path
**Edge and Node Flow Functions**

\[ \text{In}_n \rightarrow f_n \rightarrow \text{Out}_n \]

\[ \text{In}_m \rightarrow f_m \rightarrow \text{Out}_m \]
Edge and Node Flow Functions

Forward Node Flow Function

\[ f^f_n \quad f^f_{n \rightarrow m} \quad f^f_m \]

\[ \text{In}_n \quad \text{Out}_n \quad \text{In}_m \quad \text{Out}_m \]
Edge and Node Flow Functions

Forward Node Flow Function

Forward Edge Flow Function

In_n \rightarrow f^f_n \rightarrow Out_n

In_m \rightarrow f^f_{n \rightarrow m} \rightarrow Out_m

In_n \rightarrow f^b_n \rightarrow Out_n

In_m \rightarrow f^b_{n \rightarrow m} \rightarrow Out_m
Edge and Node Flow Functions

Forward Node Flow Function

\[ f_n^f \]

\[ f_{n\rightarrow m}^f \]

\[ f_m^f \]

\[ f_{n\rightarrow m}^b \]

\[ f_n^b \]

\[ f_m^b \]

\[ f_{n\rightarrow m}^b \]

Backward Node Flow Function

Aug 2017
Edge and Node Flow Functions

Forward Node Flow Function

\[ f^f_n \]

\[ f^f_{n \rightarrow m} \]

\[ f^b_n \]

\[ f^b_{n \rightarrow m} \]

Forward Edge Flow Function

Backward Node Flow Function

\[ f^b_n \]

\[ f^b_{n \rightarrow m} \]

Backward Edge Flow Function
General Data Flow Equations

\[
\begin{align*}
\text{In}_n &= \begin{cases} 
    \text{BI}_{\text{Start}} \sqcap f^b_n(\text{Out}_n) & n = \text{Start} \\
    \bigcap_{m \in \text{pred}(n)} f^f_{m \rightarrow n}(\text{Out}_m) \sqcap f^b_n(\text{Out}_n) & \text{otherwise}
\end{cases} \\
\text{Out}_n &= \begin{cases} 
    \text{BI}_{\text{End}} \sqcap f^f_n(\text{In}_n) & n = \text{End} \\
    \bigcap_{m \in \text{succ}(n)} f^b_{m \rightarrow n}(\text{In}_m) \sqcap f^f_n(\text{In}_n) & \text{otherwise}
\end{cases}
\end{align*}
\]

- Edge flow functions are typically identity
  \[\forall x \in L, \ f(x) = x\]

- If particular flows are absent, the corresponding flow functions are
  \[\forall x \in L, \ f(x) = \top\]

Aug 2017
Modelling Information Flows Using Edge and Node Flow Functions

Forward

\[
f_{k \rightarrow l}^f \circ f_k^f \circ f_{i \rightarrow k}^f
\]

Backward

\[
f_{i \rightarrow k}^b \circ f_k^b \circ f_{j \rightarrow k}^b
\]

Bidirectional

\[
f_{j \rightarrow k}^b \circ f_i^f \circ f_{k \rightarrow l}^f
\]

\[
f_{k \rightarrow l}^f \circ f_k^b \circ f_{k \rightarrow m}^b
\]
Information Flow Paths in PRE

- Information could flow along arbitrary paths
Information Flow Paths in PRE

- Information could flow along arbitrary paths
Information Flow Paths in PRE

- Information could flow along arbitrary paths
Information Flow Paths in PRE

- Information could flow along arbitrary paths
Information Flow Paths in PRE

- Information could flow along arbitrary paths
Information Flow Paths in PRE

- Information could flow along arbitrary paths
Information Flow Paths in PRE

- Information could flow along arbitrary paths
Information Flow Paths in PRE

- Information could flow along arbitrary paths
Information Flow Paths in PRE

- Information could flow along arbitrary paths
- Theoretically predicted number: 144
Information could flow along arbitrary paths
- Theoretically predicted number: 144
- Actual iterations: 5
Information Flow Paths in PRE

- Information could flow along arbitrary paths
- Theoretically predicted number: 144
- Actual iterations: 5
- Not related to depth (1)
Complexity of Worklist Algorithms for Bit Vector Frameworks

- Assume \( n \) nodes and \( r \) entities
- Total number of data flow values = \( 2 \cdot n \cdot r \)
- A data flow value can change at most once
- Complexity is \( \mathcal{O} (n \cdot r) \)
Complexity of Worklist Algorithms for Bit Vector Frameworks

- Assume $n$ nodes and $r$ entities
- Total number of data flow values $= 2 \cdot n \cdot r$
- A data flow value can change at most once
- Complexity is $O(n \cdot r)$
- Must be same for both unidirectional and bidirectional frameworks
  (Number of data flow values does not change!)
Lacuna with Older Estimates of PRE Complexity

- Lacuna with PRE: Complexity
  - $r$ is typically $\mathcal{O}(n)$
  - Assuming that at most one data flow value changes in one traversal
Lacuna with Older Estimates of PRE Complexity

- Lacuna with PRE: Complexity
  - $r$ is typically $O(n)$
  - Assuming that at most one data flow value changes in one traversal
  - Worst case number of traversals = $O(n^2)$
Lacuna with Older Estimates of PRE Complexity

- Lacuna with PRE: Complexity
  - $r$ is typically $O(n)$
  - Assuming that at most one data flow value changes in one traversal
  - Worst case number of traversals = $O(n^2)$

- Practical graphs may have up to 50 nodes
  - Predicted number of traversals: 2,500
  - Practical number of traversals: $\leq 5$
Lacuna with Older Estimates of PRE Complexity

- Lacuna with PRE: Complexity
  - $r$ is typically $\mathcal{O}(n)$
  - Assuming that at most one data flow value changes in one traversal
  - Worst case number of traversals $= \mathcal{O}(n^2)$

- Practical graphs may have up to 50 nodes
  - Predicted number of traversals: 2,500
  - Practical number of traversals: $\leq 5$

- No explanation for about 14 years despite dozens of efforts
Lacuna with Older Estimates of PRE Complexity

- Lacuna with PRE: Complexity
  - $r$ is typically $O(n)$
  - Assuming that at most one data flow value changes in one traversal
  - Worst case number of traversals = $O(n^2)$

- Practical graphs may have up to 50 nodes
  - Predicted number of traversals: 2,500
  - Practical number of traversals: $\leq 5$

- No explanation for about 14 years despite dozens of efforts

- Not much experimentation with performing advanced optimizations involving bidirectional dependency
Complexity of Round Robin Iterative Method

- Buy OTC (Over-The-Counter) medicine
- No U-Turn
- 1 Trip
Complexity of Round Robin Iterative Method

- Buy OTC (Over-The-Counter) medicine  No U-Turn  1 Trip
- Buy cloth. Give it to the tailor for stitching  No U-Turn  1 Trip
Complexity of Round Robin Iterative Method

- Buy OTC (Over-The-Counter) medicine  No U-Turn  1 Trip
- Buy cloth. Give it to the tailor for stitching  No U-Turn  1 Trip
- Buy medicine with doctor’s prescription  1 U-Turn  2 Trips
The diagnosis requires X-Ray

- Buy OTC (Over-The-Counter) medicine: No U-Turn 1 Trip
- Buy cloth. Give it to the tailor for stitching: No U-Turn 1 Trip
- Buy medicine with doctor’s prescription: 1 U-Turn 2 Trips
- Buy medicine with doctor’s prescription: 2 U-Turns 3 Trips
Information Flow Paths and Width of a Graph

- A traversal $u \rightarrow v$ in an ifp is
  - *Compatible* if $u$ is visited *before* $v$ in the chosen graph traversal
  - *Incompatible* if $u$ is visited *after* $v$ in the chosen graph traversal
Information Flow Paths and Width of a Graph

• A traversal $u \rightarrow v$ in an ifp is
  
  ▶ *Compatible* if $u$ is visited before $v$ in the chosen graph traversal
  ▶ *Incompatible* if $u$ is visited after $v$ in the chosen graph traversal

• Every incompatible edge traversal requires one additional iteration
Information Flow Paths and Width of a Graph

- A traversal $u \rightarrow v$ in an ifp is
  - *Compatible* if $u$ is visited *before* $v$ in the chosen graph traversal
  - *Incompatible* if $u$ is visited *after* $v$ in the chosen graph traversal

- Every incompatible edge traversal requires one additional iteration

- Width of a program flow graph with respect to a data flow framework:
  
  _Maximum number of incompatible traversals in any ifp, no part of which is bypassed_
Information Flow Paths and Width of a Graph

- A traversal $u \rightarrow v$ in an ifp is
  - Compatible if $u$ is visited before $v$ in the chosen graph traversal
  - Incompatible if $u$ is visited after $v$ in the chosen graph traversal

- Every incompatible edge traversal requires one additional iteration

- Width of a program flow graph with respect to a data flow framework
  
  Maximum number of incompatible traversals in any ifp, no part of which is bypassed

- Width + 1 iterations are sufficient to converge on MFP solution
  (1 additional iteration may be required for verifying convergence)
Complexity of Bidirectional Bit Vector Frameworks

- Every "incompatible" edge traversal
  $$\Rightarrow$$ One additional graph traversal
Complexity of Bidirectional Bit Vector Frameworks

- Every “incompatible” edge traversal $\Rightarrow$ One additional graph traversal

- Max. Incompatible edge traversals $= Width$ of the graph $= 0$?

- Maximum number of traversals $= 1 +$ Max. incompatible edge traversals
Every “incompatible” edge traversal ⇒ One additional graph traversal

Max. Incompatible edge traversals = \textit{Width} of the graph = 1?

Maximum number of traversals = 1 + Max. incompatible edge traversals
Complexity of Bidirectional Bit Vector Frameworks

- Every “incompatible” edge traversal \( \Rightarrow \) One additional graph traversal
- Max. Incompatible edge traversals = \( \text{Width of the graph} = 2? \)
- Maximum number of traversals = 1 + Max. incompatible edge traversals

Aug 2017
IIT Bombay
Complexity of Bidirectional Bit Vector Frameworks

- Every “incompatible” edge traversal ⇒ One additional graph traversal
- Max. Incompatible edge traversals = Width of the graph = 3?
- Maximum number of traversals = 1 + Max. incompatible edge traversals
Complexity of Bidirectional Bit Vector Frameworks

- Every “incompatible” edge traversal ⇒ One additional graph traversal
- Max. Incompatible edge traversals = Width of the graph = 3?
- Maximum number of traversals = 1 + Max. incompatible edge traversals
Complexity of Bidirectional Bit Vector Frameworks

- Every “incompatible” edge traversal \( \Rightarrow \) One additional graph traversal

- Max. Incompatible edge traversals
  \[ = \text{Width of the graph} = 3? \]

- Maximum number of traversals
  \[ = 1 + \text{Max. incompatible edge traversals} \]
Complexity of Bidirectional Bit Vector Frameworks

- Every "incompatible" edge traversal \( \Rightarrow \text{One additional graph traversal} \)

- Max. Incompatible edge traversals
  \( = \text{Width of the graph} = 3? \)

- Maximum number of traversals
  \( = 1 + \text{Max. incompatible edge traversals} \)
Complexity of Bidirectional Bit Vector Frameworks

- Every “incompatible” edge traversal ⇒ **One additional graph traversal**
- Max. Incompatible edge traversals
  \[= \text{Width of the graph} = 4\]
- Maximum number of traversals =
  \[1 + \text{Max. incompatible edge traversals}\]
Complexity of Bidirectional Bit Vector Frameworks

- Every “incompatible” edge traversal \( \Rightarrow \) One additional graph traversal

- Max. Incompatible edge traversals  
  \[= \text{Width of the graph} = 4\]

- Maximum number of traversals  
  \[= 1 + 4 = 5\]
Width Subsumes Depth

- Depth is applicable only to unidirectional data flow frameworks
- Width is applicable to both unidirectional and bidirectional frameworks
- For a given graph for a unidirectional bit vector framework, \( \text{Width} \leq \text{Depth} \)
  - Width provides a tighter bound
Comparison Between Width and Depth

- Depth is purely a graph theoretic property whereas width depends on control flow graph as well as the data framework.

- Comparison between width and depth is meaningful only:
  - For unidirectional frameworks
  - When the direction of traversal for computing width is the natural direction of traversal.

- Since width excludes bypassed path segments, width can be smaller than depth.
Width and Depth

\[
c = a + b
\]
\[
i = 0
\]

\[
\text{if } (i < m)
\]
\[
j = 0
\]

\[
\text{if } (j < n)
\]
\[
a = i + j
\]
\[
j = j + 1
\]
\[
i = i + 1
\]

Assuming reverse postorder traversal for available expressions analysis

- Depth = 2
**Width and Depth**

Assuming reverse postorder traversal for available expressions analysis:

- **Depth = 2**
- **Information generation point** $n_5$ kills expression “$a + b$”
Width and Depth

Assuming reverse postorder traversal for available expressions analysis

- Depth = 2
- Information generation point $n_5$ kills expression “a + b”
- Information propagation path $n_5 \rightarrow n_4 \rightarrow n_6 \rightarrow n_2$

No Gen or Kill for “a + b” along this path
### Width and Depth

#### Code Snippet

\[
c = a + b \\
i = 0
\]

\[\text{if } (i < m) \]

\[
j = 0
\]

\[\text{if } (j < n) \]

\[
a = i + j \\
j = j + 1
\]

\[
i = i + 1
\]

---

#### Analysis

Assuming reverse postorder traversal for available expressions analysis

- **Depth = 2**
- **Information generation point**
  - \( n_5 \) kills expression \( "a + b" \)
- **Information propagation path**
  - \( n_5 \rightarrow n_4 \rightarrow n_6 \rightarrow n_2 \)
  - No Gen or Kill for \( "a + b" \) along this path
- **Width = 2**
Width and Depth

Assuming reverse postorder traversal for available expressions analysis

- Depth = 2
- Information generation point
  \( n_5 \) kills expression “a + b”
- Information propagation path
  \( n_5 \rightarrow n_4 \rightarrow n_6 \rightarrow n_2 \)
  No Gen or Kill for “a + b” along this path
- Width = 2
- What about “j + 1”?
**Width and Depth**

Assuming reverse postorder traversal for available expressions analysis:

- **Depth = 2**
- **Information generation point**
  - \( n_5 \) kills expression “\( a + b \)”
- **Information propagation path**
  - \( n_5 \rightarrow n_4 \rightarrow n_6 \rightarrow n_2 \)
- No Gen or Kill for “\( a + b \)” along this path
- **Width = 2**
- **What about “\( j + 1 \)”?**
- **Not available on entry to the loop**
Width and Depth

Structures resulting from repeat-until loops with premature exits

- Depth = 3
Width and Depth

Structures resulting from repeat-until loops with premature exits

- Depth = 3
- However, any unidirectional bit vector analysis is guaranteed to converge in $2 + 1$ iterations
Width and Depth

Structures resulting from repeat-until loops with premature exits

- Depth = 3
- However, any unidirectional bit vector analysis is guaranteed to converge in \(2 + 1\) iterations
- ifp \(5 \rightarrow 4 \rightarrow 6\) is bypassed by the edge \(5 \rightarrow 6\)
Width and Depth

Structures resulting from repeat-until loops with premature exits

- Depth = 3
- However, any unidirectional bit vector analysis is guaranteed to converge in $2 + 1$ iterations
- If $5 \rightarrow 4 \rightarrow 6$ is bypassed by the edge $5 \rightarrow 6$
- If $6 \rightarrow 3 \rightarrow 7$ is bypassed by the edge $6 \rightarrow 7$
Width and Depth

Structures resulting from repeat-until loops with premature exits

- Depth = 3
- However, any unidirectional bit vector analysis is guaranteed to converge in $2 + 1$ iterations
  - ifp $5 \rightarrow 4 \rightarrow 6$ is bypassed by the edge $5 \rightarrow 6$
  - ifp $6 \rightarrow 3 \rightarrow 7$ is bypassed by the edge $6 \rightarrow 7$
  - ifp $7 \rightarrow 2 \rightarrow 8$ is bypassed by the edge $7 \rightarrow 8$
Width and Depth

Structures resulting from repeat-until loops with premature exits

- Depth = 3
- However, any unidirectional bit vector analysis is guaranteed to converge in $2 + 1$ iterations
  - ifp $5 \rightarrow 4 \rightarrow 6$ is bypassed by the edge $5 \rightarrow 6$
  - ifp $6 \rightarrow 3 \rightarrow 7$ is bypassed by the edge $6 \rightarrow 7$
  - ifp $7 \rightarrow 2 \rightarrow 8$ is bypassed by the edge $7 \rightarrow 8$
- For forward unidirectional frameworks, width is 1
Width and Depth

Structures resulting from repeat-until loops with premature exits

- **Depth** = 3

- However, any unidirectional bit vector analysis is guaranteed to converge in $2 + 1$ iterations

  - ifp $5 \rightarrow 4 \rightarrow 6$ is bypassed by the edge $5 \rightarrow 6$
  - ifp $6 \rightarrow 3 \rightarrow 7$ is bypassed by the edge $6 \rightarrow 7$
  - ifp $7 \rightarrow 2 \rightarrow 8$ is bypassed by the edge $7 \rightarrow 8$

- For forward unidirectional frameworks, width is 1

- Splitting the bypassing edges and inserting nodes along those edges increases the width
Work List Based Iterative Algorithm

Directly traverses information flow paths

1. $I_{n_0} = B_I$
2. for all $j \neq 0$ do
3.   \{
4.     $I_{n_j} = \top$
5.     Add $j$ to LIST
6.   \}
7. while LIST is not empty do
8.   \{
9.     Let $j$ be the first node in LIST. Remove it from LIST
10.    $temp = \underset{p \in \text{pred}(j)}{\prod} f_p(I_{n_p})$
11.    if $temp \neq I_{n_j}$ then
12.       \{
13.         $I_{n_j} = temp$
14.         Add all successors of $j$ to LIST
15.       \}
16.   \}
Perform work list based iterative analysis for earlier examples. Assume that the work list follows FIFO (First in First Out) policy.

Show the trace of the analysis in the following format:

<table>
<thead>
<tr>
<th>Step</th>
<th>Node</th>
<th>Remaining work list</th>
<th>Out DFV</th>
<th>Change?</th>
<th>Node Added</th>
<th>Resulting work list</th>
</tr>
</thead>
</table>

Perform work list based iterative analysis for earlier examples. Assume that the work list follows FIFO (First in First Out) policy.

Show the trace of the analysis in the following format:
Tutorial Problem for Work List Based Analysis

\[
\begin{align*}
c &= a + b \\
i &= 0
\end{align*}
\]

\[
\begin{align*}
\text{if (i < m)} & \\
j &= 0
\end{align*}
\]

\[
\begin{align*}
\text{if (j < n)} & \\
a &= i + j \\
j &= j + 1
\end{align*}
\]

\[
i = i + 1
\]

For available expressions analysis

- Round robin method needs 3+1 iterations
  Total number of nodes processed = 7 × 4 = 28
- We illustrate work list method for expression \( a + b \)
  (other expressions are unavailable in the first iteration because of BI)
# Tutorial Problem for Work List Based Analysis

<table>
<thead>
<tr>
<th>Step</th>
<th>Node</th>
<th>Remaining work list</th>
<th>$Out$ DFV</th>
<th>Change?</th>
<th>Node Added</th>
<th>Resulting work list</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$n_1$</td>
<td>$n_2, n_3, n_4, n_5, n_6, n_7$</td>
<td>1</td>
<td>No</td>
<td></td>
<td>$n_2, n_3, n_4, n_5, n_6, n_7$</td>
</tr>
<tr>
<td>2</td>
<td>$n_2$</td>
<td>$n_3, n_4, n_5, n_6, n_7$</td>
<td>1</td>
<td>No</td>
<td></td>
<td>$n_3, n_4, n_5, n_6, n_7$</td>
</tr>
<tr>
<td>3</td>
<td>$n_3$</td>
<td>$n_4, n_5, n_6, n_7$</td>
<td>1</td>
<td>No</td>
<td></td>
<td>$n_4, n_5, n_6, n_7$</td>
</tr>
<tr>
<td>4</td>
<td>$n_4$</td>
<td>$n_5, n_6, n_7$</td>
<td>1</td>
<td>No</td>
<td></td>
<td>$n_5, n_6, n_7$</td>
</tr>
<tr>
<td>5</td>
<td>$n_5$</td>
<td>$n_6, n_7$</td>
<td>0</td>
<td>Yes</td>
<td>$n_4$</td>
<td>$n_6, n_7, n_4$</td>
</tr>
<tr>
<td>6</td>
<td>$n_6$</td>
<td>$n_7, n_4$</td>
<td>1</td>
<td>No</td>
<td></td>
<td>$n_7, n_4$</td>
</tr>
<tr>
<td>7</td>
<td>$n_7$</td>
<td>$n_4$</td>
<td>1</td>
<td>No</td>
<td></td>
<td>$n_4$</td>
</tr>
<tr>
<td>8</td>
<td>$n_4$</td>
<td>$n_4$</td>
<td>0</td>
<td>Yes</td>
<td>$n_5, n_6$</td>
<td>$n_5, n_6$</td>
</tr>
<tr>
<td>9</td>
<td>$n_5$</td>
<td>$n_6$</td>
<td>0</td>
<td>No</td>
<td></td>
<td>$n_6$</td>
</tr>
<tr>
<td>10</td>
<td>$n_6$</td>
<td></td>
<td>0</td>
<td>Yes</td>
<td>$n_2$</td>
<td>$n_2$</td>
</tr>
<tr>
<td>11</td>
<td>$n_2$</td>
<td></td>
<td>0</td>
<td>Yes</td>
<td>$n_3, n_7$</td>
<td>$n_3, n_7$</td>
</tr>
<tr>
<td>12</td>
<td>$n_3$</td>
<td>$n_7$</td>
<td>0</td>
<td>Yes</td>
<td>$n_4$</td>
<td>$n_7, n_4$</td>
</tr>
<tr>
<td>13</td>
<td>$n_7$</td>
<td>$n_4$</td>
<td>0</td>
<td>Yes</td>
<td></td>
<td>$n_4$</td>
</tr>
<tr>
<td>14</td>
<td>$n_4$</td>
<td></td>
<td>0</td>
<td>No</td>
<td></td>
<td>Empty $\implies$ End</td>
</tr>
</tbody>
</table>
Part 10

Precise Modelling of General Flows
Complexity of Constant Propagation?

1. $a = b + 1$
2. $b = c + 1$
3. $c = d + 1$
4. $d = 2$
Complexity of Constant Propagation?

Iteration #1

1

2 $a = b + 1$

3 $b = c + 1$

4 $c = d + 1$

5 $d = 2$
Complexity of Constant Propagation?

Iteration #1

\[ a = b + 1 \]
\[ b = c + 1 \]
\[ c = d + 1 \]
\[ d = 2 \]

Iteration #2

\[ a = b + 1 \]
\[ b = c + 1 \]
\[ c = d + 1 \]
\[ d = 2 \]
Complexity of Constant Propagation?

Iteration #1
- $a = b + 1$
- $b = c + 1$
- $c = d + 1$
- $d = 2$

Iteration #2
- $a = b + 1$
- $b = c + 1$
- $c = 3$
- $d = 2$

Iteration #3
- $a = b + 1$
- $b = 4$
- $c = 3$
- $d = 2$
Complexity of Constant Propagation?

Iteration #1

1. $a = b + 1$
2. $b = c + 1$
3. $c = d + 1$
4. $d = 2$

Iteration #2

1. $a = b + 1$
2. $b = c + 1$
3. $c = 3$
4. $d = 2$

Iteration #3

1. $a = b + 1$
2. $b = 4$
3. $c = 3$
4. $d = 2$

Iteration #4

1. $a = 5$
2. $b = 3$
3. $c = 3$
4. $d = 2$
Part 11

Extra Topics
Tarski’s Fixed Point Theorem

Given monotonic $f : L \rightarrow L$ where $L$ is a complete lattice

Define

- $p$ is a fixed point of $f$:
  \[ Fix(f) = \{ p \mid f(p) = p \} \]
- $f$ is reductive at $p$:
  \[ Red(f) = \{ p \mid f(p) \sqsubseteq p \} \]
- $f$ is extensive at $p$:
  \[ Ext(f) = \{ p \mid f(p) \sqsupseteq p \} \]

Then

\[
LFP(f) = \bigsqcap \{ Red(f) \in Fix(f) \}
\]
\[
MFP(f) = \bigsqcup \{ Ext(f) \in Fix(f) \}
\]
Tarski’s Fixed Point Theorem

Given monotonic $f : L \to L$ where $L$ is a complete lattice

Define

- $p$ is a fixed point of $f$: $\text{Fix}(f) = \{p \mid f(p) = p\}$
- $f$ is reductive at $p$: $\text{Red}(f) = \{p \mid f(p) \sqsubseteq p\}$
- $f$ is extensive at $p$: $\text{Ext}(f) = \{p \mid f(p) \sqsupseteq p\}$

Then

$$
\text{LFP}(f) = \bigsqcap \text{Red}(f) \in \text{Fix}(f)
$$
$$
\text{MFP}(f) = \bigsqcup \text{Ext}(f) \in \text{Fix}(f)
$$

Guarantees only existence, not computability of fixed points
Fixed Points of a Function
Fixed Points of a Function

\[ f^n(\top) \]

\[ \text{Red}(f) \]
Fixed Points of a Function

\[\text{Red}(f)\]

\[\text{Ext}(f)\]

\[f^n(\top)\]

\[f^n(\bot)\]
Fixed Points of a Function

\[ \text{Red}(f) \]
\[ \text{Fix}(f) \]
\[ \text{Ext}(f) \]

\[ f^n(\top) \]
\[ MFP(f) \]

\[ f^n(\bot) \]
\[ LFP(f) \]

Aug 2017
Examples of Reductive and Extensive Sets

Finite $L$     Monotonic $f : L \rightarrow L$

\[
\begin{align*}
\top & \xrightarrow{f} \top \\
\downarrow & \quad \downarrow \\
\nu_1 & \xrightarrow{f} \nu_1 \\
\downarrow & \quad \downarrow \\
\nu_2 & \xrightarrow{f} \nu_2 \\
\downarrow & \quad \downarrow \\
\nu_3 & \xrightarrow{f} \nu_3 \\
\downarrow & \quad \downarrow \\
\nu_4 & \xrightarrow{f} \nu_4 \\
\downarrow & \quad \downarrow \\
\bot & \xrightarrow{f} \bot
\end{align*}
\]

\[
\begin{align*}
\text{Red}(f) & = \{\top, \nu_3, \nu_4, \bot\} \\
\text{Ext}(f) & = \{\top, \nu_1, \nu_2, \bot\} \\
\text{Fix}(f) & = \text{Red}(f) \cap \text{Ext}(f) \\
& = \{\top, \bot\} \\
\text{MFP}(f) & = \text{lub}(\text{Ext}(f)) \\
& = \text{lub}(\text{Fix}(f)) \\
& = \top \\
\text{LFP}(f) & = \text{glb}(\text{Red}(f)) \\
& = \text{glb}(\text{Fix}(f)) \\
& = \bot
\end{align*}
\]
Existence of MFP: Proof of Tarski’s Fixed Point Theorem

$Ext(f)$
Existence of MFP: Proof of Tarski’s Fixed Point Theorem

1. Claim 1: Let $X \subseteq L$.
   \[ \forall x \in X,\ p \sqsupseteq x \Rightarrow p \sqsupseteq \bigcup(X). \]
Existence of MFP: Proof of Tarski’s Fixed Point Theorem

1. Claim 1: Let $X \subseteq L$.
   $\forall x \in X, p \supseteq x \Rightarrow p \supseteq \bigcup(X)$.

2. In the following we use $\text{Ext}(f)$ as $X$. 
Existence of MFP: Proof of Tarski’s Fixed Point Theorem

1. Claim 1: Let $X \subseteq L$.
   \[ \forall x \in X, \ p \supseteq x \Rightarrow p \supseteq \bigsqcup(X) . \]

2. In the following we use $\text{Ext}(f)$ as $X$

3. \[ \forall p \in \text{Ext}(f), \ hi \supseteq p \]
Existence of MFP: Proof of Tarski’s Fixed Point Theorem

1. Claim 1: Let \( X \subseteq L \).
   \[
   \forall x \in X, \ p \supseteq x \implies p \supseteq \bigcup(X).
   \]

2. In the following we use \( \text{Ext}(f) \) as \( X \).

3. \( \forall p \in \text{Ext}(f), \ hi \supseteq p \)

4. \( hi \supseteq p \implies f(hi) \supseteq f(p) \supseteq p \) (monotonicity)
   \[
   \implies f(hi) \supseteq hi \quad \text{(claim 1)}
   \]
Existence of MFP: Proof of Tarski’s Fixed Point Theorem

1. Claim 1: Let $X \subseteq L$.
   \[ \forall x \in X, \ p \supseteq x \Rightarrow p \supseteq \bigcup(X). \]

2. In the following we use $\text{Ext}(f)$ as $X$.

3. $\forall p \in \text{Ext}(f), \ hi \supseteq p$.

4. $hi \supseteq p \Rightarrow f(hi) \supseteq f(p) \supseteq p$ (monotonicity)
   \[ \Rightarrow f(hi) \supseteq hi \quad \text{(claim 1)} \]

5. $f$ is extensive at $hi$ also: $hi \in \text{Ext}(f)$.
Existence of MFP: Proof of Tarski’s Fixed Point Theorem

1. Claim 1: Let $X \subseteq L$.
   \[ \forall x \in X, \ p \supseteq x \Rightarrow p \supseteq \bigcup(X). \]

2. In the following we use $\text{Ext}(f)$ as $X$.

3. $\forall p \in \text{Ext}(f), \ hi \supseteq p$.

4. $hi \supseteq p \Rightarrow f(hi) \supseteq f(p) \supseteq p$ (monotonicity)
   \[ \Rightarrow f(hi) \supseteq hi \] (claim 1)

5. $f$ is extensive at $hi$ also: $hi \in \text{Ext}(f)$.

6. $f(hi) \supseteq hi \Rightarrow f^2(hi) \supseteq f(hi)$
   \[ \Rightarrow f(hi) \in \text{Ext}(f) \]
   \[ \Rightarrow hi \supseteq f(hi) \] (from 3)
   \[ \Rightarrow hi = f(hi) \Rightarrow hi \in \text{Fix}(f) \]
Existence of MFP: Proof of Tarski’s Fixed Point Theorem

1. Claim 1: Let $X \subseteq L$.
   \[ \forall x \in X, \ p \sqsupseteq x \Rightarrow p \sqsupseteq \bigcup(X). \]

2. In the following we use $\text{Ext}(f)$ as $X$

3. $\forall p \in \text{Ext}(f), \ hi \sqsupseteq p$

4. $hi \sqsupseteq p \Rightarrow f(hi) \sqsupseteq f(p) \sqsupseteq p$ (monotonicity)
   \[ \Rightarrow f(hi) \sqsupseteq hi \quad \text{(claim 1)} \]

5. $f$ is extensive at $hi$ also: $hi \in \text{Ext}(f)$

6. $f(hi) \sqsupseteq hi \Rightarrow f^2(hi) \sqsupseteq f(hi)$
   \[ \Rightarrow f(hi) \in \text{Ext}(f) \]
   \[ \Rightarrow hi \sqsupseteq f(hi) \quad \text{(from 3)} \]
   \[ \Rightarrow hi = f(hi) \Rightarrow hi \in \text{Fix}(f) \]

7. $\text{Fix}(f) \subseteq \text{Ext}(f)$ (by definition)
   \[ \Rightarrow hi \sqsupseteq p, \ \forall p \in \text{Fix}(f) \]
Existence and Computation of the Maximum Fixed Point

• For monotonic $f : L \to L$
Existence and Computation of the Maximum Fixed Point

• For monotonic $f : L \rightarrow L$
  
  - Existence: $MFP(f) = \bigsqcup Ext(f) \in Fix(f)$
  
  Requires $L$ to be complete
Existence and Computation of the Maximum Fixed Point

- For monotonic $f : L \rightarrow L$
  - Existence: $MFP(f) = \bigsqcup \text{Ext}(f) \in \text{Fix}(f)$
    Requires $L$ to be complete
  - Computation: $MFP(f) = f^{k+1}(\top) = f^k(\top)$ such that
    $f^{j+1}(\top) \neq f^j(\top)$, $j < k$.
    Requires all *strictly descending* chains to be finite
Existence and Computation of the Maximum Fixed Point

- For monotonic $f : L \to L$
  - Existence: $\text{MFP}(f) = \bigsqcup \text{Ext}(f) \in \text{Fix}(f)$
    
    Requires $L$ to be complete
  - Computation: $\text{MFP}(f) = f^{k+1}(\top) = f^k(\top)$ such that $f^{j+1}(\top) \neq f^j(\top)$, $j < k$.

    Requires all strictly descending chains to be finite

- Finite strictly descending and ascending chains
  
  $\Rightarrow$ Completeness of lattice
Existence and Computation of the Maximum Fixed Point

- For monotonic $f : L \to L$
  - Existence: $\text{MFP}(f) = \bigsqcup \text{Ext}(f) \in \text{Fix}(f)$
    Requires $L$ to be complete
  - Computation: $\text{MFP}(f) = f^{k+1}(\top) = f^k(\top)$ such that $f^{j+1}(\top) \neq f^j(\top)$, $j < k$.
    Requires all *strictly descending* chains to be finite

- Finite strictly descending and ascending chains
  $\Rightarrow$ Completeness of lattice

- Completeness of lattice $\nRightarrow$ Finite strictly descending chains
Existence and Computation of the Maximum Fixed Point

- For monotonic $f : L \rightarrow L$
  - Existence: $MFP(f) = \bigsqcup Ext(f) \in Fix(f)$
    Requires $L$ to be complete
  - Computation: $MFP(f) = f^{k+1}(\top) = f^k(\top)$ such that $f^{j+1}(\top) \neq f^j(\top)$, $j < k$.
    Requires all strictly descending chains to be finite

- Finite strictly descending and ascending chains
  $\Rightarrow$ Completeness of lattice

- Completeness of lattice $\not\Rightarrow$ Finite strictly descending chains

$\Rightarrow$ Even if MFP exists, it may not be reachable unless all strictly descending chains are finite
Framework Properties Influencing Complexity

Depends on the loop closure properties of the framework

\[ f^*(x) = x \cap f(x) \cap f^2(x) \cap \ldots \cap f^{k-1}(x) \]

Necessary and sufficient
Framework Properties Influencing Complexity

Depends on the loop closure properties of the framework

\( k \)-Bounded Frameworks

Fast Frameworks \((k = 2)\)

\[ f^2(x) \supseteq f(x) \cap x \]

\[ f^\ast(x) = x \cap f(x) \cap f^2(x) \cap \ldots \cap f^{k-1}(x) \]

Necessary and sufficient
Framework Properties Influencing Complexity

Depends on the loop closure properties of the framework

\[ f^2(x) \supseteq f(x) \]

\[ f^2(x) \supseteq f(x) \cap x \]

\[ f^*(x) = x \cap f(x) \cap f^2(x) \cap \ldots \cap f^{k-1}(x) \]
Framework Properties Influencing Complexity

Depends on the loop closure properties of the framework

\[ f^\ast(x) = x \cap f(x) \cap f^2(x) \cap \ldots \cap f^{k-1}(x) \]

- **k-Bounded Frameworks**
- **Fast Frameworks** \((k = 2)\)
- **Rapid Frameworks**
- **Bit Vector Frameworks**
Complexity of Round Robin Iterative Algorithm

- Unidirectional rapid frameworks

<table>
<thead>
<tr>
<th>Task</th>
<th>Number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Irreducible $G$</td>
</tr>
<tr>
<td>Initialisation</td>
<td>$1$</td>
</tr>
<tr>
<td>Convergence (until $change$ remains true)</td>
<td>$d(G, T) + 1$</td>
</tr>
<tr>
<td>Verifying convergence (change becomes false)</td>
<td>$1$</td>
</tr>
</tbody>
</table>