General Data Flow Frameworks

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

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

September 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 book


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

- Modelling General Flows
- Constant Propagation
- Strongly Live Variables Analysis (after mid-sem)
- Pointer Analyses (after mid-sem)
- Heap Reference Analysis (after mid-sem)
Part 2

Precise Modelling of General Flows
Complexity of Constant Propagation?

1. $a = b + 1$
2. $b = c + 1$
3. $c = d + 1$
4. $d = 2$
5. 

Diagram:

1. 
2. $a = b + 1$ 
3. $b = c + 1$ 
4. $c = d + 1$ 
5. $d = 2$
Complexity of Constant Propagation?

Iteration #1
Complexity of Constant Propagation?

Iteration #1

1

2 a = b + 1

3 b = c + 1

4 c = d + 1

5 d = 2

Iteration #2

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 = 3
d = 2

Iteration #3

a = b + 1
b = 4
c = 3
d = 2
Complexity of Constant Propagation?

Iteration #1

Iteration #2

Iteration #3

Iteration #4
Loop Closures of Flow Functions

Table:

<table>
<thead>
<tr>
<th>Paths Terminating at $p_2$</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>$p_1, p_2, p_3, p_2, p_2, p_3, p_2$</td>
<td>$f(f(f(x))) = f^3(x)$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Loop Closures of Flow Functions

- For static analysis we need to summarize the value at \( p_2 \) by a value which is safe after any iteration.

\[
f^*(x) = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap f^4(x) \sqcap \ldots
\]
Loop Closures of Flow Functions

For static analysis we need to summarize the value at $p_2$ by a value which is safe after any iteration.

$$f^*(x) = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap f^4(x) \sqcap \ldots$$

- $f^*$ is called the loop closure of $f$. 

<table>
<thead>
<tr>
<th>Paths Terminating at $p_2$</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>$p_1, p_2, p_3, p_2, p_3, p_2, p_3, p_2$</td>
<td>$f(f(f(x))) = f^3(x)$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Loop Closure Boundedness

- Boundedness of $f$ requires the existence of some $k$ such that
  \[ f^*(x) = x \sqcap f(x) \sqcap f^2(x) \sqcap \ldots \sqcap f^{k-1}(x) \]

- This follows from the descending chain condition

- For efficiency, we need a constant $k$ that is independent of the size of the lattice
Loop Closures in Bit Vector Frameworks

- Flow functions in bit vector frameworks have constant Gen and Kill

\[
\begin{align*}
  f^*(x) &= x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots \\
  f^2(x) &= f \left( \text{Gen} \cup (x - \text{Kill}) \right) \\
         &= \text{Gen} \cup ((\text{Gen} \cup (x - \text{Kill})) - \text{Kill}) \\
         &= \text{Gen} \cup ((\text{Gen} - \text{Kill}) \cup (x - \text{Kill})) \\
         &= \text{Gen} \cup (\text{Gen} - \text{Kill}) \cup (x - \text{Kill}) \\
         &= \text{Gen} \cup (x - \text{Kill}) = f(x) \\
  f^*(x) &= x \sqcap f(x)
\end{align*}
\]
Loop Closures in Bit Vector Frameworks

- Flow functions in bit vector frameworks have constant Gen and Kill

\[
\begin{align*}
    f^*(x) &= x \cap f(x) \cap f^2(x) \cap f^3(x) \cap \ldots \\
    f^2(x) &= f(Gen \cup (x - Kill)) \\
            &= Gen \cup ((Gen \cup (x - Kill)) - Kill) \\
            &= Gen \cup ((Gen - Kill) \cup (x - Kill)) \\
            &= Gen \cup (Gen - Kill) \cup (x - Kill) \\
            &= Gen \cup (x - Kill) = f(x) \\
    f^*(x) &= x \cap f(x)
\end{align*}
\]

- Loop Closures of Bit Vector Frameworks are 2-bounded.
Loop Closures in Bit Vector Frameworks

- Flow functions in bit vector frameworks have constant Gen and Kill

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

\[
f^2(x) = f(\text{Gen} \cup (x - \text{Kill}))
\]

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

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

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

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

\[
f^*(x) = x \sqcap f(x)
\]

- Loop Closures of Bit Vector Frameworks are 2-bounded.

- Intuition: Since Gen and Kill are constant, same things are generated or killed in every application of \(f\).

Multiple applications of \(f\) are not required unless the input value changes.
Larger Values of Loop Closure Bounds

- Fast Frameworks ≡ 2-bounded frameworks (e.g., bit vector frameworks)
  Both these conditions must be satisfied
  - *Separability*
    Data flow values of different entities are independent
  - *Constant or Identity Flow Functions*
    Flow functions for an entity are either constant or identity

- Non-fast frameworks
  At least one of the above conditions is violated
Separability

\[ f : L \rightarrow L \text{ is } \langle \hat{h}_1, \hat{h}_2, \ldots, \hat{h}_m \rangle \text{ where } \hat{h}_i \text{ computes the value of } \hat{x}_i \]
Separability

\[ f : L \rightarrow L = \langle \hat{h}_1, \hat{h}_2, \ldots, \hat{h}_m \rangle \text{ where } \hat{h}_i \text{ computes the value of } \hat{x}_i \]

<table>
<thead>
<tr>
<th>Separable</th>
<th>Non-Separable</th>
</tr>
</thead>
</table>

Example: All bit vector frameworks
Example: Constant Propagation
Separability

$f : L \rightarrow L$ is $\langle \hat{h}_1, \hat{h}_2, \ldots, \hat{h}_m \rangle$ where $\hat{h}_i$ computes the value of $\hat{x}_i$

**Separable**

$\langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle$

$\downarrow f$

$\langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle$

**Non-Separable**

$\langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle$

$\downarrow f$

$\langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle$

Example: All bit vector frameworks

Example: Constant Propagation
Separability

\( f : L \rightarrow L \) is \( \langle \hat{h}_1, \hat{h}_2, \ldots, \hat{h}_m \rangle \) where \( \hat{h}_i \) computes the value of \( \hat{x}_i \)

**Separable**

\[
\langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle
\]

\[
\langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle
\]

**Non-Separable**

\[
\langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle \xrightarrow{f} \langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle
\]

Example: All bit vector frameworks

Example: Constant Propagation
Separability

\[ f : L \rightarrow L \text{ is } \langle \hat{h}_1, \hat{h}_2, \ldots, \hat{h}_m \rangle \text{ where } \hat{h}_i \text{ computes the value of } \hat{x}_i \]

**Separable**

\[ \langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle \]

\[ \hat{h}_2 \]

\[ \langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle \]

\[ \hat{h} : \hat{L} \rightarrow \hat{L} \]

**Non-Separable**

\[ \langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle \]

\[ f \]

\[ \langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle \]

Example: All bit vector frameworks  
Example: Constant Propagation

Sep 2017
Separability

\( f : L \rightarrow L \text{ is } \langle \hat{h}_1, \hat{h}_2, \ldots, \hat{h}_m \rangle \) where \( \hat{h}_i \) computes the value of \( \hat{x}_i \)

**Separable**

\[ \langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle \]

\[ \hat{h}_2 \]

\[ \langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle \]

**Non-Separable**

\[ \langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle \]

\[ \hat{h}_2 \]

\[ \langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle \]

\[ \hat{h} : \hat{L} \rightarrow \hat{L} \]

Example: All bit vector frameworks

Example: Constant Propagation
Separability

\[ f : L \to L \text{ is } \langle \hat{h}_1, \hat{h}_2, \ldots, \hat{h}_m \rangle \text{ where } \hat{h}_i \text{ computes the value of } \hat{x}_i \]

**Separable**

\[ \langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle \]

\[ \hat{h}_2 \]

\[ \langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle \]

\[ \hat{h} : \hat{L} \to \hat{L} \]

**Non-Separable**

\[ \langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle \]

\[ \hat{h}_2 \]

\[ \langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle \]

\[ \hat{h} : L \to \hat{L} \]

Example: All bit vector frameworks

Example: Constant Propagation
Separability of Bit Vector Frameworks

- $\hat{L}$ is $\{0, 1\}$, $L$ is $\{0, 1\}^m$
- $\hat{\cap}$ is either boolean AND or boolean OR
- $\hat{\top}$ and $\hat{\bot}$ are 0 or 1 depending on $\hat{\cap}$.
- $\hat{h}$ is a *bit function* and could be one of the following:

<table>
<thead>
<tr>
<th>Raise</th>
<th>Lower</th>
<th>Propagate</th>
<th>Negate</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="#" alt="Diagram" /></td>
<td><img src="#" alt="Diagram" /></td>
<td><img src="#" alt="Diagram" /></td>
<td><img src="#" alt="Diagram" /></td>
</tr>
</tbody>
</table>
Separability of Bit Vector Frameworks

- $\hat{L}$ is $\{0, 1\}$, $L$ is $\{0, 1\}^m$
- $\hat{\wedge}$ is either boolean AND or boolean OR
- $\hat{\top}$ and $\hat{\bot}$ are 0 or 1 depending on $\hat{\wedge}$.
- $\hat{h}$ is a bit function and could be one of the following:

<table>
<thead>
<tr>
<th>Raise</th>
<th>Lower</th>
<th>Propagate</th>
<th>Negate</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Diagram" /></td>
<td><img src="image" alt="Diagram" /></td>
<td><img src="image" alt="Diagram" /></td>
<td><img src="image" alt="Diagram" /></td>
</tr>
</tbody>
</table>

Non-monotonicity
Larger Values of Loop Closure Bounds

Composite flow function for the loop is

\[ f(\langle v_a, v_b, v_c, v_d \rangle) = \langle v_b + 1, v_c + 1, v_d + 1, 2 \rangle \]
Larger Values of Loop Closure Bounds

Composite flow function for the loop is

\[ f\left(\langle v_a, v_b, v_c, v_d \rangle\right) = \langle v_b + 1, v_c + 1, v_d + 1, 2 \rangle \]

\( f \) is not 2-bounded because:
Larger Values of Loop Closure Bounds

Composite flow function for the loop is

\[ f(⟨v_a, v_b, v_c, v_d⟩) = ⟨v_b + 1, v_c + 1, v_d + 1, 2⟩ \]

\[ f \] is not 2-bounded because:

\[ f(⟨\hat{T}, \hat{T}, \hat{T}, \hat{T}⟩) = ⟨\hat{T}, \hat{T}, \hat{T}, 2⟩ \]
Larger Values of Loop Closure Bounds

Composite flow function for the loop is

\[ f(\langle v_a, v_b, v_c, v_d \rangle) = \langle v_b + 1, v_c + 1, v_d + 1, 2 \rangle \]

f is not 2-bounded because:

\[ f(\langle \hat{\top}, \hat{\top}, \hat{\top}, \hat{\top} \rangle) = \langle \hat{\top}, \hat{\top}, \hat{\top}, 2 \rangle \]

\[ f^2(\langle \hat{\top}, \hat{\top}, \hat{\top}, \hat{\top} \rangle) = \langle \hat{\top}, \hat{\top}, 3, 2 \rangle \]
Larger Values of Loop Closure Bounds

Composite flow function for the loop is

\[ f(\langle v_a, v_b, v_c, v_d \rangle) = \langle v_b + 1, v_c + 1, v_d + 1, 2 \rangle \]

\[ f \] is not 2-bounded because:

\[
\begin{align*}
 f(\langle \hat{\top}, \hat{\top}, \hat{\top}, \hat{\top} \rangle) & = \langle \hat{\top}, \hat{\top}, \hat{\top}, 2 \rangle \\
 f^2(\langle \hat{\top}, \hat{\top}, \hat{\top}, \hat{\top} \rangle) & = \langle \hat{\top}, \hat{\top}, 3, 2 \rangle \\
 f^3(\langle \hat{\top}, \hat{\top}, \hat{\top}, \hat{\top} \rangle) & = \langle \hat{\top}, 4, 3, 2 \rangle
\end{align*}
\]
Larger Values of Loop Closure Bounds

Composite flow function for the loop is

\[ f(\langle v_a, v_b, v_c, v_d \rangle) = \langle v_b + 1, v_c + 1, v_d + 1, 2 \rangle \]

\( f \) is not 2-bounded because:

\[ f(\langle \hat{T}, \hat{T}, \hat{T}, \hat{T} \rangle) = \langle \hat{T}, \hat{T}, \hat{T}, 2 \rangle \]
\[ f^2(\langle \hat{T}, \hat{T}, \hat{T}, \hat{T} \rangle) = \langle \hat{T}, \hat{T}, 3, 2 \rangle \]
\[ f^3(\langle \hat{T}, \hat{T}, \hat{T}, \hat{T} \rangle) = \langle \hat{T}, 4, 3, 2 \rangle \]
\[ f^4(\langle \hat{T}, \hat{T}, \hat{T}, \hat{T} \rangle) = \langle 5, 4, 3, 2 \rangle \]
Larger Values of Loop Closure Bounds

Composite flow function for the loop is

\[ f(\langle v_a, v_b, v_c, v_d \rangle) = \langle v_b + 1, v_c + 1, v_d + 1, 2 \rangle \]

\( f \) is not 2-bounded because:

\[
\begin{align*}
    f(\langle \hat{1}, \hat{1}, \hat{1}, \hat{1} \rangle) &= \langle \hat{1}, \hat{1}, \hat{1}, 2 \rangle \\
    f^2(\langle \hat{1}, \hat{1}, \hat{1}, \hat{1} \rangle) &= \langle \hat{1}, \hat{1}, 3, 2 \rangle \\
    f^3(\langle \hat{1}, \hat{1}, \hat{1}, \hat{1} \rangle) &= \langle \hat{1}, 4, 3, 2 \rangle \\
    f^4(\langle \hat{1}, \hat{1}, \hat{1}, \hat{1} \rangle) &= \langle 5, 4, 3, 2 \rangle \\
    f^5(\langle \hat{1}, \hat{1}, \hat{1}, \hat{1} \rangle) &= \langle 5, 4, 3, 2 \rangle 
\end{align*}
\]
Part 3

Constant Propagation
Example of Constant Propagation

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

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

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

\[
\begin{align*}
\text{MoP} & \quad \langle \top, \top, \top, \top \rangle \\
\langle 1, 2, 3, \top \rangle \\
\langle \bot, \bot, 3, 2 \rangle \\
\langle \bot, \bot, 3, 2 \rangle \\
\langle 2, 1, 3, 2 \rangle \\
\end{align*}
\]
Example of Constant Propagation

\*

\[
\begin{align*}
  a &= 1 \\
  b &= 2 \\
  c &= a + b \\
  n_1
\end{align*}
\]

\[
\begin{align*}
  a &= 1 \\
  b &= 2 \\
  c &= a + b \\
  d &= a \times b \\
  n_2
\end{align*}
\]

\[
\begin{align*}
  d &= c - 1 \\
  a &= 2 \\
  b &= 1 \\
  c &= a + b \\
  n_3
\end{align*}
\]

MoP

\[
\begin{align*}
  \langle \top, \top, \top, \top \rangle & \quad \langle 1, 2, 3, \top \rangle \\
  \langle \bot, \bot, 3, 2 \rangle & \quad \langle \bot, \bot, 3, \bot \rangle \\
  \langle \bot, \bot, 3, 2 \rangle & \quad \langle \bot, \bot, \bot, \bot \rangle \\
  \langle 2, 1, 3, 2 \rangle & \quad \langle 2, 1, 3, \top \rangle \\
\end{align*}
\]

MFP

\[
\begin{align*}
  \langle \top, \top, \top, \top \rangle & \quad \langle 1, 2, 3, \top \rangle \\
  \langle \bot, \bot, 3, 2 \rangle & \quad \langle \bot, \bot, 3, \bot \rangle \\
  \langle \bot, \bot, \bot, \bot \rangle & \quad \langle \bot, \bot, \bot, \bot \rangle \\
  \langle 2, 1, 3, 2 \rangle & \quad \langle 2, 1, 3, \top \rangle \\
\end{align*}
\]
Component Lattice for Integer Constant Propagation

\[
\begin{array}{c}
\top \\
\text{undef or ud} \\
-\infty \quad \cdots \quad -2 \quad -1 \quad 0 \quad 1 \quad 2 \quad \cdots \quad \infty \\
\text{nonconst or nc} \\
\bot
\end{array}
\]

<table>
<thead>
<tr>
<th>\hat{}</th>
<th>\langle v, ud \rangle</th>
<th>\langle v, nc \rangle</th>
<th>\langle v, c_1 \rangle</th>
</tr>
</thead>
<tbody>
<tr>
<td>\langle v, ud \rangle</td>
<td>\langle v, ud \rangle</td>
<td>\langle v, nc \rangle</td>
<td>\langle v, c_1 \rangle</td>
</tr>
<tr>
<td>\langle v, nc \rangle</td>
<td>\langle v, nc \rangle</td>
<td>\langle v, nc \rangle</td>
<td>\langle v, nc \rangle</td>
</tr>
<tr>
<td>\langle v, c_2 \rangle</td>
<td>\langle v, c_2 \rangle</td>
<td>\langle v, nc \rangle</td>
<td>\text{If } c_1 = c_2 \text{ then } \langle v, c_1 \rangle \text{ else } \langle v, nc \rangle</td>
</tr>
</tbody>
</table>

If \( c_1 = c_2 \) then \( \langle v, c_1 \rangle \) else \( \langle v, nc \rangle \)
Overall Lattice for Integer Constant Propagation

- \( In_n/Out_n \) values are mappings \( \forall \text{Var} \to \hat{L} : In_n, Out_n \in \forall \text{Var} \to \hat{L} \)
- Overall lattice \( L \) is a set of mappings \( \forall \text{Var} \to \hat{L} : L = \forall \text{Var} \to \hat{L} \)
- \( \sqcap \) and \( \hat{\sqcap} \) get defined by \( \sqsubseteq \) and \( \sqsubseteq \)
  - Partial order is restricted to data flow values of the same variable
    Data flow values of different variables are incomparable
    \[
    (x, v_1) \sqsubseteq (y, v_2) \iff x = y \land v_1 \hat{\sqsubseteq} v_2
    \]
  - OR
    \[
    x \mapsto v_1 \sqsubseteq y \mapsto v_2 \iff x = y \land v_1 \hat{\sqsubseteq} v_2
    \]
  - For meet operation, we assume that \( X \) is a total function
    Partial functions are made total by using \( \hat{\top} \) value
    \[
    X \sqcap Y = \{(x, v_1 \hat{\sqcap} v_2) | (x, v_1) \in X, (x, v_2) \in Y\}
    \]
    OR
    \[
    X \sqcap Y = \{x \mapsto v_1 \hat{\sqcap} v_2 | x \mapsto v_1 \in X, x \mapsto v_2 \in Y\}
    \]
Notations for Mappings as Data Flow Values

Accessing and manipulating a mapping \( X \subseteq A \rightarrow B \)

- \( X(a) \) denotes the image of \( a \in A \)
  \( X(a) \in B \)
- \( X[a \mapsto v] \) changes the image of \( a \) in \( X \) to \( v \)

\[
X[a \mapsto v] = (X \setminus \{(a, u) \mid u \in B\}) \cup \{(a, v)\}
\]
Defining Data Flow Equations for Constant Propagation

\[\begin{align*}
B I &= \{\langle y, ud \rangle \mid y \in V a r\} && n = Start \\
L n_n &= \begin{cases} 
B I & n = Start \\
\prod_{p \in \text{pred}(n)} \text{Out}_p & \text{otherwise}
\end{cases} \\
\text{Out}_n &= f_n(L n_n)
\end{align*}\]

\[f_n(X) = \begin{cases} 
X[y \mapsto c] & n \text{ is } y = c, y \in V a r, c \in \text{Const} \\
X[y \mapsto nc] & n \text{ is } \text{input}(y), y \in \text{var} \\
X[y \mapsto X(z)] & n \text{ is } y = z, y \in V a r, z \in V a r \\
X[y \mapsto \text{eval}(e, X)] & n \text{ is } y = e, y \in V a r, e \in \text{Expr} \\
X & \text{otherwise}
\end{cases}\]

\[\begin{align*}
\text{eval}(e, X) &= \begin{cases} 
nc & a \in \text{Opd}(e) \cap V a r, X(a) = nc \\
ud & a \in \text{Opd}(e) \cap V a r, X(a) = ud \\
-X(a) & e \text{ is } -a \\
X(a) \oplus X(b) & e \text{ is } a \oplus b
\end{cases}
\end{align*}\]
Example Program for Constant Propagation

```
input (e);

a = 7; b = 2; f = e;
if (f > 0)
  a = 2;
  if (f ≥ e + 2)
    b = c + 1;
    if (b ≥ 7)
      f = f + 1;
      c = d * a;
    false
  if (f ≥ e + 1)
    f = f + 1;
    d = a + 1;
    f = f + 1
true
false
```

```
e = a + b;
```

```
```
Example Program for Constant Propagation

For readability, we have combined many statements in a single block. However, constant propagation requires every basic block to contain a single statement because of the presence of dependent parts in flow functions.
## Result of Constant Propagation

<table>
<thead>
<tr>
<th></th>
<th>Iteration #1</th>
<th>Changes in iteration #2</th>
<th>Changes in iteration #3</th>
<th>Changes in iteration #4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$In_{n_1}$</td>
<td>$\top, \top, \top, \top, \top, \top$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Out_{n_1}$</td>
<td>$\top, \top, \top, \top, \bot, \top$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$In_{n_2}$</td>
<td>$\top, \top, \top, \top, \bot, \top$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Out_{n_2}$</td>
<td>$7, 2, \top, \top, \bot, \bot$</td>
<td>$1, 2, \top, 3, \bot, \bot$</td>
<td>$\top, 2, 6, 3, \bot, \bot$</td>
<td>$\top, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$In_{n_3}$</td>
<td>$7, 2, \top, \top, \bot, \bot$</td>
<td>$2, 2, \top, 3, \bot, \bot$</td>
<td>$2, 2, 6, 3, \bot, \bot$</td>
<td>$2, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$Out_{n_3}$</td>
<td>$2, 2, \top, \top, \bot, \bot$</td>
<td>$2, 2, \top, 3, \bot, \bot$</td>
<td>$2, 2, 6, 3, \bot, \bot$</td>
<td>$2, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$In_{n_4}$</td>
<td>$2, \top, \top, \top, \bot, \bot$</td>
<td>$2, 2, \top, 3, \bot, \bot$</td>
<td>$2, 2, 6, 3, \bot, \bot$</td>
<td>$2, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$Out_{n_4}$</td>
<td>$2, \top, \top, \top, \bot, \bot$</td>
<td>$2, \top, \top, 3, \bot, \bot$</td>
<td>$2, 7, 6, 3, \bot, \bot$</td>
<td>$2, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$In_{n_5}$</td>
<td>$2, \top, \top, \top, \bot, \bot$</td>
<td>$2, \top, \top, 3, \bot, \bot$</td>
<td>$2, 7, 6, 3, \bot, \bot$</td>
<td>$2, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$Out_{n_5}$</td>
<td>$2, \top, \top, \top, \bot, \bot$</td>
<td>$2, \top, \top, 3, \bot, \bot$</td>
<td>$2, 7, 6, 3, \bot, \bot$</td>
<td>$2, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$In_{n_6}$</td>
<td>$2, \top, \top, \top, \bot, \bot$</td>
<td>$2, \top, \top, 3, \bot, \bot$</td>
<td>$2, 2, 6, 3, \bot, \bot$</td>
<td>$2, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$Out_{n_6}$</td>
<td>$2, \top, \top, \top, \bot, \bot$</td>
<td>$2, \top, \top, 3, \bot, \bot$</td>
<td>$2, 2, 6, 3, \bot, \bot$</td>
<td>$2, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$In_{n_7}$</td>
<td>$2, \top, \top, \top, \bot, \bot$</td>
<td>$2, \top, \top, 3, \bot, \bot$</td>
<td>$2, \top, 6, 3, \bot, \bot$</td>
<td>$2, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$Out_{n_7}$</td>
<td>$2, \top, \top, \top, \bot, \bot$</td>
<td>$2, 2, 6, 3, \bot, \bot$</td>
<td>$2, \bot, 6, 3, \bot, \bot$</td>
<td>$2, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$In_{n_8}$</td>
<td>$2, \top, \top, \top, \bot, \bot$</td>
<td>$2, \top, \top, 3, \bot, \bot$</td>
<td>$2, 2, 6, 3, \bot, \bot$</td>
<td>$2, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$Out_{n_8}$</td>
<td>$2, \top, 4, \top, \bot, \bot$</td>
<td>$2, \top, 4, \top, \bot, \bot$</td>
<td>$2, 2, 6, 4, \bot, \bot$</td>
<td>$2, \bot, 6, 4, \bot, \bot$</td>
</tr>
<tr>
<td>$In_{n_9}$</td>
<td>$2, \top, 4, \top, \bot, \bot$</td>
<td>$2, 2, 6, \top, \bot, \bot$</td>
<td>$2, \top, 6, \top, \bot, \bot$</td>
<td>$2, \bot, 6, \top, \bot, \bot$</td>
</tr>
<tr>
<td>$Out_{n_9}$</td>
<td>$2, \top, 3, \top, \bot, \bot$</td>
<td>$2, 2, 6, 3, \bot, \bot$</td>
<td>$2, \bot, 6, 3, \bot, \bot$</td>
<td>$2, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$In_{n_{10}}$</td>
<td>$\bot, 2, \top, \top, \top, \bot, \bot$</td>
<td>$\bot, 2, \top, 3, \bot, \bot$</td>
<td>$\bot, \top, 6, 3, \bot, \bot$</td>
<td>$\bot, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$Out_{n_{10}}$</td>
<td>$\bot, 2, \top, \top, \top, \bot, \bot$</td>
<td>$\bot, 2, \top, 3, \bot, \bot$</td>
<td>$\bot, \top, 6, 3, \bot, \bot$</td>
<td>$\bot, \bot, 6, 3, \bot, \bot$</td>
</tr>
</tbody>
</table>
Result of Constant Propagation

\( n_1 \) input \((e)\);

\( n_2 \)
- \( a = 7; b = 2; f = e; \)
- if \((f > 0)\)

\( n_3 \)
- \( a = 2; \)
- if \((f \geq e + 2)\)

\( n_4 \)
- \( b = c + 1; \)
- if \((b \geq 7)\)

\( n_5 \)
- \( f = f + 1; \)

\( n_6 \)
- if \((f \geq e + 1)\)

\( n_7 \)
- \( c = d \ast a; \)

\( n_8 \)
- \( d = a + b; \)

\( n_9 \)
- \( d = a + 1; \)
- \( f = f + 1\)

\( n_{10} \)
- \( e = a + b; \)
Result of Constant Propagation

\[ n_1 \text{ input (e);} \]
\[ n_2 \text{ a = 7; b = 2; f = e; if (f > 0)} \]
\[ n_3 \text{ a = 2; if (f ≥ e + 2)} \]
\[ n_4 \text{ b = c + 1; if (b ≥ 7)} \]
\[ n_5 \text{ f = f + 1; } \]
\[ n_6 \text{ if (f ≥ e + 1)} \]
\[ n_7 \text{ c = d * a; } \]
\[ n_8 \text{ d = a + b; } \]
\[ n_9 \text{ d = a + 1; f = f + 1; } \]
\[ n_{10} \text{ e = a + b; } \]
Result of Constant Propagation

- **n1**: input (e);
- **n2**: 
  - a = 7; b = 2; f = e;
  - if (f > 0)
- **n3**: 
  - a = 2;
  - if (f ≥ e + 2)
- **n4**: 
  - b = c + 1;
  - if (b ≥ 7)
- **n5**: 
  - f = f + 1;
- **n7**: 
  - c = d * a;
- **n8**: 
  - d = a + b;
- **n9**: 
  - d = a + 1;
  - f = f + 1
- **n10**: e = a + b;
Result of Constant Propagation

\begin{equation*}
\begin{aligned}
n_1 & \text{ input } (e); \\
n_2 & \text{ } \\
& a = 7; \ b = 2; \ f = e; \\
& \text{if } (f > 0) \\
& a = 2; \\
& \text{if } (f \geq e + 2) \\
& b = c + 1; \\
& \text{if } (b \geq 7) \\
& c = 6 \\
& \text{true} \\
& \text{false} \\
& f = f + 1; \\
& \text{true} \\
& \text{false} \\
& d = a + b; \\
& c = d \ast a; \\
& a = 2, \ d = 3 \\
& d = a + 1; \\
& f = f + 1 \\
& a = 2 \\
& \text{true} \\
& \text{false} \\
& e = a + b; \\
\end{aligned}
\end{equation*}
Monotonicity of Constant Propagation

Proof obligation: $X_1 \sqsubseteq X_2 \Rightarrow f_n(X_1) \sqsubseteq f_n(X_2)$

where,

$$f_n(X) = \begin{cases} 
X[y \mapsto c] & n \text{ is } y = c, y \in \mathbb{Var}, c \in \mathbb{Const} 
\quad (C1) \\
X[y \mapsto nc] & n \text{ is } \text{input}(y), y \in \mathbb{Var} 
\quad (C2) \\
X[y \mapsto X(z)] & n \text{ is } y = z, y \in \mathbb{Var}, z \in \mathbb{Var} 
\quad (C3) \\
X[y \mapsto \text{eval}(e, X)] & n \text{ is } y = e, y \in \mathbb{Var}, e \in \mathbb{Expr} 
\quad (C4) \\
X & \text{otherwise} 
\quad (C5)
\end{cases}$$

- The proof obligation trivially follows for cases C1, C2, C3, and C5
- For case C4, it requires showing

  $X_1 \sqsubseteq X_2 \Rightarrow \text{eval}(e, X_1) \sqsubseteq \text{eval}(e, X_2)$

  which follows from the definition of $\text{eval}(e, X)$
Non-Distributivity of Constant Propagation

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

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

\[ d = c - 1 \\
\quad a = 2 \\
\quad b = 1 \\
\quad c = a + b \]
Non-Distributivity of Constant Propagation

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

Diagram:

$\begin{array}{|l|}
\hline
n_1 & a = 1 \\
& b = 2 \\
& c = a + b \\
\hline
\end{array}$

$\begin{array}{|l|}
\hline
n_2 & a = 1, b = 2 \\
& c = a + b \\
& d = a \times b \\
\hline
\end{array}$

$\begin{array}{|l|}
\hline
n_3 & c = a + b \\
& d = c - 1 \\
& a = 2 \\
& b = 1 \\
& c = a + b \\
\hline
\end{array}$
Non-Distributivity of Constant Propagation

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

- \( x = \langle 1, 2, 3, ? \rangle \) (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 before merging

\[
f(x) \sqcap f(y) = f(\langle 1, 2, 3, ? \rangle) \sqcap f(\langle 2, 1, 3, 2 \rangle)
= \langle 1, 2, 3, 2 \rangle \sqcap \langle 2, 1, 3, 2 \rangle
= \langle \hat{\bot}, \hat{\bot}, 3, 2 \rangle
\]
Non-Distributivity of Constant Propagation

- \( x = \langle 1, 2, 3, ? \rangle \) (Along \( Out_{n_1} \to In_{n_2} \))
- \( y = \langle 2, 1, 3, 2 \rangle \) (Along \( Out_{n_3} \to In_{n_2} \))
- Function application before merging
  \[
  f(x) \sqcap f(y) = f(\langle 1, 2, 3, ? \rangle) \sqcap f(\langle 2, 1, 3, 2 \rangle) = \langle 1, 2, 3, 2 \rangle \sqcap \langle 2, 1, 3, 2 \rangle = \langle \perp, \perp, 3, 2 \rangle
  \]
- Function application after merging
  \[
  f(x \sqcap y) = f(\langle 1, 2, 3, ? \rangle \sqcap \langle 2, 1, 3, 2 \rangle) = f(\langle \perp, \perp, 3, 2 \rangle) = \langle \perp, \perp, \perp, \perp \rangle
  \]
Non-Distributivity of Constant Propagation

- $x = \langle 1, 2, 3, ? \rangle$ (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 before merging
  \[
  f(x) \sqcap f(y) = f(\langle 1, 2, 3, ? \rangle) \sqcap f(\langle 2, 1, 3, 2 \rangle)
  = \langle 1, 2, 3, 2 \rangle \sqcap \langle 2, 1, 3, 2 \rangle
  = \langle \perp, \perp, 3, 2 \rangle
  \]
- Function application after merging
  \[
  f(x \sqcap y) = f(\langle 1, 2, 3, ? \rangle \sqcap \langle 2, 1, 3, 2 \rangle)
  = f(\langle \perp, \perp, 3, 2 \rangle)
  = \langle \perp, \perp, \perp, \perp \rangle
  \]
- $f(x \sqcap y) \sqsubseteq f(x) \sqcap f(y)$
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

\[
\begin{align*}
a = 1 & \quad a = 2 \\
\text{Possible combinations due to merging} & \quad b = 1 \\
b = 2 & \quad b = 2
\end{align*}
\]
Why is Constant Propagation Non-Distributive?

Possible combinations due to merging

\[
\begin{align*}
\text{a} &= 1 & \text{a} &= 2 & \text{b} &= 1 & \text{b} &= 2 \\
\text{c} = \text{a} + \text{b} &= 3
\end{align*}
\]

• Correct combination.
Why is Constant Propagation Non-Distributive?

Possible combinations due to merging

\[
\begin{align*}
    a &= 1 \\
    b &= 2 \\
    c &= a + b = 3
\end{align*}
\]

- Correct combination.
Why is Constant Propagation Non-Distributive?

Possible combinations due to merging

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

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

\[
c = a + b = 2
\]

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

Possible combinations due to merging:

\[
\begin{align*}
    a &= 1 \\
    b &= 2 \\
    c &= a + b = 3 \\
\end{align*}
\]

\[
\begin{align*}
    a &= 2 \\
    b &= 1 \\
    c &= a + b = 3 \\
\end{align*}
\]

- Wrong combination.
- Mutually exclusive information.
- No execution path along which this information holds.
Tutorial Problem on Constant Propagation

How many iterations do we need?

\[
\begin{align*}
\text{n1} & = b \\
\text{n2} & = b \\
\text{n3} & = d = 2 \\
\text{n4} & = b \\
\text{n5} & = c = d \\
\text{n6} & = b \\
\text{n7} & = b = c \\
\text{n8} & = a = b \\
\text{n9} & = a = b \\
\text{n10} & = b
\end{align*}
\]
Tutorial Problem on Constant Propagation

How many iterations do we need?
Tutorial Problem on Constant Propagation

How many iterations do we need?
Tutorial Problem on Constant Propagation

How many iterations do we need?

4
Tutorial Problem on Constant Propagation

How many iterations do we need?
Tutorial Problem on Constant Propagation

How many iterations do we need?
Tutorial Problem on Constant Propagation

How many iterations do we need?

- Every back edge occurs only once in the ifp from $n_3$ to $n_1$ that goes via $n_5$, $n_7$, and $n_9$.
- $5 + 1$ iterations for computing data flow values (+1 iteration to detect convergence)
Tutorial Problem on Constant Propagation

And now how many iterations do we need?
And now how many iterations do we need?

Back edge $n_{10} \rightarrow n_1$ needs to be traversed once each for back edges $n_9 \rightarrow n_8$, $n_7 \rightarrow n_6$, $n_5 \rightarrow n_4$, and $n_3 \rightarrow n_2$ (in that order).

$\Rightarrow 8 + 1$ iterations.
Boundedness of Constant Propagation

\[ a = 1 \]

\[ a = b + 1 \]

\[ b = c + 1 \]

\[ c = a + 1 \]
Boundedness of Constant Propagation

Summary flow function:
(data flow value at node 7)

\[ f(\langle v_a, v_b, v_c \rangle) = \langle 1 \sqcap (v_b + 1), (v_c + 1), (v_a + 1) \rangle \]
Boundedness of Constant Propagation

Summary flow function:
(data flow value at node 7)

\[ f(\langle v_a, v_b, v_c \rangle) = \langle 1 \sqcap (v_b + 1), (v_c + 1), (v_a + 1) \rangle \]

\[ f^0(\top) = \langle \top, \top, \top \rangle \]
\[ f^1(\top) = \langle 1, \top, \top \rangle \]
Boundedness of Constant Propagation

Summary flow function:
(data flow value at node 7)

\[ f(\langle v_a, v_b, v_c \rangle) = \langle 1 \sqcap (v_b + 1),
(v_c + 1),
(v_a + 1) \rangle \]

\[
\begin{align*}
  f^0(\top) &= \langle \hat{1}, \hat{1}, \hat{1} \rangle \\
  f^1(\top) &= \langle 1, \hat{1}, \hat{1} \rangle \\
  f^2(\top) &= \langle 1, \hat{1}, 2 \rangle
\end{align*}
\]
**Boundedness of Constant Propagation**

**Summary flow function:**
(data flow value at node 7)

\[
f(\langle v_a, v_b, v_c \rangle) = \langle 1 \sqcap (v_b + 1), (v_c + 1), (v_a + 1) \rangle
\]

\[
f^0(\langle \top \rangle) = \langle \top, \top, \top \rangle \\
f^1(\langle \top \rangle) = \langle 1, \top, \top \rangle \\
f^2(\langle \top \rangle) = \langle 1, \top, 2 \rangle \\
f^3(\langle \top \rangle) = \langle 1, 3, 2 \rangle
\]
Boundedness of Constant Propagation

Summary flow function:
(data flow value at node 7)

\[
f(\langle v_a, v_b, v_c \rangle) = \langle 1 \sqcap (v_b + 1),
(v_c + 1),
(v_a + 1) \rangle
\]

\[
f^0(\top) = \langle \hat{\top}, \hat{\top}, \hat{\top} \rangle
\]
\[
f^1(\top) = \langle 1, \hat{\top}, \hat{\top} \rangle
\]
\[
f^2(\top) = \langle 1, \hat{\top}, 2 \rangle
\]
\[
f^3(\top) = \langle 1, 3, 2 \rangle
\]
\[
f^4(\top) = \langle \bot, 3, 2 \rangle
\]
Boundedness of Constant Propagation

Summary flow function:
(data flow value at node 7)

\[ f(\langle v_a, v_b, v_c \rangle) = \langle v_a + 1, v_b + 1, v_c + 1 \rangle \]

\[
\begin{align*}
    f^0(\top) &= \langle \top, \top, \top \rangle \\
    f^1(\top) &= \langle 1, \top, \top \rangle \\
    f^2(\top) &= \langle 1, \top, 2 \rangle \\
    f^3(\top) &= \langle 1, 3, 2 \rangle \\
    f^4(\top) &= \langle \bot, 3, 2 \rangle \\
    f^5(\top) &= \langle \bot, 3, \bot \rangle
\end{align*}
\]
Boundedness of Constant Propagation

Summary flow function:
(data flow value at node 7)

\[ f(\langle v_a, v_b, v_c \rangle) = \langle 1 \sqcap (v_b + 1), (v_c + 1), (v_a + 1) \rangle \]

\[
\begin{align*}
  f^0(\top) &= \langle \uparrow, \uparrow, \uparrow \rangle \\
  f^1(\top) &= \langle 1, \uparrow, \uparrow \rangle \\
  f^2(\top) &= \langle 1, \uparrow, 2 \rangle \\
  f^3(\top) &= \langle 1, 3, 2 \rangle \\
  f^4(\top) &= \langle \perp, 3, 2 \rangle \\
  f^5(\top) &= \langle \perp, 3, \perp \rangle \\
  f^6(\top) &= \langle \perp, \perp, \perp \rangle \\
\end{align*}
\]
Boundedness of Constant Propagation

Summary flow function:
(data flow value at node 7)

\[ f(\langle v_a, v_b, v_c \rangle) = \langle 1 \sqcap (v_b + 1), (v_c + 1), (v_a + 1) \rangle \]

\[
\begin{align*}
  f^0(\top) &= \langle \mathbf{\top}, \mathbf{\top}, \mathbf{\top} \rangle \\
  f^1(\top) &= \langle 1, \mathbf{\top}, \mathbf{\top} \rangle \\
  f^2(\top) &= \langle 1, \mathbf{\top}, 2 \rangle \\
  f^3(\top) &= \langle 1, 3, 2 \rangle \\
  f^4(\top) &= \langle \mathbf{\bot}, 3, 2 \rangle \\
  f^5(\top) &= \langle \mathbf{\bot}, 3, \mathbf{\bot} \rangle \\
  f^6(\top) &= \langle \mathbf{\bot}, \mathbf{\bot}, \mathbf{\bot} \rangle \\
  f^7(\top) &= \langle \mathbf{\bot}, \mathbf{\bot}, \mathbf{\bot} \rangle 
\end{align*}
\]
Boundedness of Constant Propagation

\[ f^*(\mathbb{T}) = \prod_{i=0}^{6} f^i(\mathbb{T}) \]
Boundedness of Constant Propagation

The moral of the story:

- The data flow value of every variable could change twice
Boundedness of Constant Propagation

The moral of the story:

- The data flow value of every variable could change twice
- In the worst case, only one change may happen in every step of a function application
Boundedness of Constant Propagation

The moral of the story:

- The data flow value of every variable could change twice
- In the worst case, only one change may happen in every step of a function application
- Maximum number of steps: $2 \times |\overline{Var}|$
Boundedness of Constant Propagation

The moral of the story:

- The data flow value of every variable could change twice
- In the worst case, only one change may happen in every step of a function application
- Maximum number of steps: $2 \times |\forall \text{Var}|$
- Boundedness parameter $k$ is $(2 \times |\forall \text{Var}|) + 1$
Conditional Constant Propagation

An execution trace of the program when the value read for variable e is some number $x \leq 0$ (otherwise the loop will not be entered).

```
input (e);

a = 7; b = 2; f = e;
if (f > 0)

true

a = 2;
if (f \geq e + 2)

true

b = c + 1;
if (b \geq 7)

false

f = f + 1;

false

c = d * a;

false

d = a + b;

true

d = a + 1;
f = f + 1
```

Sep 2017
Conditional Constant Propagation

An execution trace of the program when the value read for variable e is some number $x \leq 0$ (otherwise the loop will not be entered)

<n_1> input (e);

<n_2> $a = 7; b = 2; f = e;$
if ($f > 0$)

<n_3> $a = 2;$
if ($f \geq e + 2$)

<n_4> $b = c + 1;$
if ($b \geq 7$)

<n_5> $f = f + 1;$
if ($f \geq e + 1$)

<n_6> $d = a + b;$
if ($f \geq e + 1$)

<n_7> $c = d \ast a;$
$n_8 d = a + b;$
$n_9 d = a + 1;$
$f = f + 1$

$n_{10} e = a + b;$
Conditional Constant Propagation

An execution trace of the program when the value read for variable e is some number $x \leq 0$ (otherwise the loop will not be entered)
Conditional Constant Propagation

An execution trace of the program when the value read for variable e is some number $x \leq 0$ (otherwise the loop will not be entered)
Conditional Constant Propagation

An execution trace of the program when the value read for variable e is some number \( x \leq 0 \) (otherwise the loop will not be entered)
Conditional Constant Propagation

An execution trace of the program when the value read for variable $e$ is some number $x \leq 0$ (otherwise the loop will not be entered)
Conditional Constant Propagation

An execution trace of the program when the value read for variable \( e \) is some number \( x \leq 0 \) (otherwise the loop will not be entered)
Conditional Constant Propagation

An execution trace of the program when the value read for variable e is some number x ≤ 0 (otherwise the loop will not be entered)
Conditional Constant Propagation

An execution trace of the program when the value read for variable $e$ is some number $x \leq 0$ (otherwise the loop will not be entered)

1. $a = 7; b = 2; f = e$
2. if $(f > 0)$
3. $a = 2$
   - if $(f \geq e + 2)$
4. $b = c + 1$
   - if $(b \geq 7)$
5. $f = f + 1$
6. if $(f \geq e + 1)$
7. $c = d \times a$
8. $d = a + b$
9. $d = a + 1$
   - $f = f + 1$
10. $e = a + b$
Conditional Constant Propagation

An execution trace of the program when the value read for variable e is some number $x \leq 0$ (otherwise the loop will not be entered)

$\langle 2, 2, 6, 3, x, x+2 \rangle$
Conditional Constant Propagation

An execution trace of the program when the value read for variable $e$ is some number $x \leq 0$ (otherwise the loop will not be entered)
Conditional Constant Propagation

An execution trace of the program when the value read for variable \( e \) is some number \( x \leq 0 \) (otherwise the loop will not be entered)
Conditional Constant Propagation

An execution trace of the program when the value read for variable \( e \) is some number \( x \leq 0 \) (otherwise the loop will not be entered).

regardless of the input value of \( e \), \( b \) is constant in the loop (with value 2) and constant propagation cannot discover it.
Lattice for Conditional Constant Propagation

Let \( \langle s, X \rangle \) denote an augmented data flow value where 
\( s \in \{ \text{reachable}, \text{notReachable} \} \) and \( X \in L \).

If we can maintain the invariant \( s = \text{notReachable} \Rightarrow X = \top \), then the meet can be defined as

\[
\langle s_1, X_1 \rangle \sqcap \langle s_2, X_2 \rangle = \langle s_1 \sqcap s_2, X_1 \sqcap X_2 \rangle
\]
Data Flow Equations for Conditional Constant Propagation

\[ \text{In}_n = \begin{cases} \langle \text{reachable}, BI \rangle & \text{if } n \text{ is Start} \\ \prod_{p \in \text{pred}(n)} g_{p \rightarrow n}(\text{Out}_p) & \text{otherwise} \end{cases} \]

\[ \text{Out}_n = \begin{cases} \langle \text{reachable}, f_n(X) \rangle & \text{if } \text{In}_n = \langle \text{reachable}, X \rangle \\ \langle \text{notReachable}, \top \rangle & \text{otherwise} \end{cases} \]

\[ g_{m \rightarrow n}(s, X) = \begin{cases} \langle s, X \rangle & \text{if } \text{label}(m \rightarrow n) \in \text{evalCond}(m, X) \\ \langle \text{notReachable}, \top \rangle & \text{otherwise} \end{cases} \]

- \text{label}(m \rightarrow n) \text{ is } T \text{ or } F \text{ if edge } m \rightarrow n \text{ is a conditional branch}
  
  Otherwise \text{ label}(m \rightarrow n) \text{ is } T

- \text{evalCond}(m, X) \text{ evaluates the condition in block } m \text{ using the data flow values in } X
Compile Time Evaluation of Conditions using the Data Flow Values

<table>
<thead>
<tr>
<th>evalCond(m, X)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ T, F }</td>
<td>Block ( m ) does not have a condition, or some variable in the condition is ( \bot ) in ( X )</td>
</tr>
<tr>
<td>{}</td>
<td>No variable in the condition in block ( m ) is ( \bot ) in ( X ), but some variable is ( \top ) in ( X )</td>
</tr>
<tr>
<td>{ T }</td>
<td>The condition in block ( m ) evaluates to ( T ) with the data flow values in ( X )</td>
</tr>
<tr>
<td>{ F }</td>
<td>The condition in block ( m ) evaluates to ( F ) with the data flow values in ( X )</td>
</tr>
</tbody>
</table>
# Conditional Constant Propagation

<table>
<thead>
<tr>
<th></th>
<th>Iteration #1</th>
<th>Changes in iteration #2</th>
<th>Changes in iteration #3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{In}_{n_1})</td>
<td>(R, \langle \top, \top, \top, \top, \top \rangle)</td>
<td>(R, \langle \top, 2, \top, 3, \perp, \perp \rangle)</td>
<td>(R, \langle \top, 2, 6, 3, \perp, \perp \rangle)</td>
</tr>
<tr>
<td>(\text{Out}_{n_1})</td>
<td>(R, \langle \top, \top, \top, \top, \top \rangle)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\text{In}_{n_2})</td>
<td>(R, \langle \top, \top, \top, \top, \top \rangle)</td>
<td>(R, \langle \top, 2, 6, 3, \perp, \perp \rangle)</td>
<td></td>
</tr>
<tr>
<td>(\text{Out}_{n_2})</td>
<td>(R, \langle 7, 2, \top, \top, \top, \top \rangle)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\text{In}_{n_3})</td>
<td>(R, \langle 7, 2, \top, \top, \top, \top \rangle)</td>
<td>(R, \langle 1, 2, \top, 3, \perp, \perp \rangle)</td>
<td>(R, \langle 1, 2, 6, 3, \perp, \perp \rangle)</td>
</tr>
<tr>
<td>(\text{Out}_{n_3})</td>
<td>(R, \langle 2, 2, \top, \top, \top, \top \rangle)</td>
<td>(R, \langle 2, 2, \top, 3, \perp, \perp \rangle)</td>
<td>(R, \langle 2, 2, 6, 3, \perp, \perp \rangle)</td>
</tr>
<tr>
<td>(\text{In}_{n_4})</td>
<td>(R, \langle 2, 2, \top, \top, \top, \top \rangle)</td>
<td>(R, \langle 2, 2, \top, 3, \perp, \perp \rangle)</td>
<td>(R, \langle 2, 2, 6, 3, \perp, \perp \rangle)</td>
</tr>
<tr>
<td>(\text{Out}_{n_4})</td>
<td>(R, \langle 2, \top, \top, \top, \perp, \perp \rangle)</td>
<td>(R, \langle 2, \top, \top, 3, \perp, \perp \rangle)</td>
<td>(R, \langle 2, 7, 6, 3, \perp, \perp \rangle)</td>
</tr>
<tr>
<td>(\text{In}_{n_5})</td>
<td>(N, \top = \langle \top, \top, \top, \top, \top \rangle)</td>
<td>(R, \langle 2, 2, \top, 3, \perp, \perp \rangle)</td>
<td></td>
</tr>
<tr>
<td>(\text{Out}_{n_5})</td>
<td>(N, \top = \langle \top, \top, \top, \top, \top \rangle)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\text{In}_{n_6})</td>
<td>(R, \langle 2, 2, \top, \top, \top, \top \rangle)</td>
<td>(R, \langle 2, 2, \top, 3, \perp, \perp \rangle)</td>
<td>(R, \langle 2, 2, 6, 3, \perp, \perp \rangle)</td>
</tr>
<tr>
<td>(\text{Out}_{n_6})</td>
<td>(R, \langle 2, 2, \top, \top, \top, \top \rangle)</td>
<td>(R, \langle 2, 2, \top, 3, \perp, \perp \rangle)</td>
<td>(R, \langle 2, 2, 6, 3, \perp, \perp \rangle)</td>
</tr>
<tr>
<td>(\text{In}_{n_7})</td>
<td>(R, \langle 2, 2, \top, \top, \top, \perp, \perp \rangle)</td>
<td>(R, \langle 2, 2, \top, 3, \perp, \perp \rangle)</td>
<td>(R, \langle 2, 2, 6, 3, \perp, \perp \rangle)</td>
</tr>
<tr>
<td>(\text{Out}_{n_7})</td>
<td>(R, \langle 2, 2, \top, \top, \top, \perp, \perp \rangle)</td>
<td>(R, \langle 2, 2, 6, 3, \perp, \perp \rangle)</td>
<td></td>
</tr>
<tr>
<td>(\text{In}_{n_8})</td>
<td>(R, \langle 2, 2, \top, \top, \top, \perp, \perp \rangle)</td>
<td>(R, \langle 2, 2, \top, 3, \perp, \perp \rangle)</td>
<td>(R, \langle 2, 2, 6, 3, \perp, \perp \rangle)</td>
</tr>
<tr>
<td>(\text{Out}_{n_8})</td>
<td>(R, \langle 2, 2, \top, 4, \perp, \perp \rangle)</td>
<td>(R, \langle 2, 2, 6, \perp, \perp \rangle)</td>
<td>(R, \langle 2, 2, 6, 4, \perp, \perp \rangle)</td>
</tr>
<tr>
<td>(\text{In}_{n_9})</td>
<td>(R, \langle 2, 2, \top, 4, \perp, \perp \rangle)</td>
<td>(R, \langle 2, 2, 6, \perp, \perp \rangle)</td>
<td></td>
</tr>
<tr>
<td>(\text{Out}_{n_9})</td>
<td>(R, \langle 2, 2, \top, 3, \perp, \perp \rangle)</td>
<td>(R, \langle 2, 2, 6, 3, \perp, \perp \rangle)</td>
<td></td>
</tr>
<tr>
<td>(\text{In}<em>{n</em>{10}})</td>
<td>(R, \langle 7, 2, \top, \top, \top, \perp, \perp \rangle)</td>
<td>(R, \langle 1, 2, \top, 3, \perp, \perp \rangle)</td>
<td>(R, \langle 1, 1, 6, 3, \perp, \perp \rangle)</td>
</tr>
<tr>
<td>(\text{Out}<em>{n</em>{10}})</td>
<td>(R, \langle 7, 2, \top, \top, \perp, \perp \rangle)</td>
<td>(R, \langle 1, 2, \top, 3, \perp, \perp \rangle)</td>
<td>(R, \langle 1, 1, 6, 3, \perp, \perp \rangle)</td>
</tr>
</tbody>
</table>
Part 4

Strongly Live Variables Analysis
Strongly Live Variables Analysis

• A variable is strongly live if
  ▶ it is used in a statement other than assignment statement, or
    (same as simple liveness)
  ▶ it is used in an assignment statement defining a variable that is strongly live
    (different from simple liveness)

• Killing: An assignment statement, an input statement, or BI
  (this is same as killing in simple liveness)

• Generation: A direct use or a use for defining values that are strongly live
  (this is different from generation in simple liveness)
Understanding Strong Liveness

\[
\begin{align*}
    y &= x \\
    \text{print } (x)
\end{align*}
\]

\[
\begin{align*}
    y &= x \\
    \text{print } (y)
\end{align*}
\]

\[
\begin{align*}
    y &= x \\
    \text{print } (z)
\end{align*}
\]
Understanding Strong Liveness

\[ y = x \]
\[ \text{print}(x) \]

\[ y = x \]
\[ \text{print}(y) \]

\[ y = x \]
\[ \text{print}(z) \]
Understanding Strong Liveness

$y = x$

print $(x)$

$\emptyset$

$y = x$

print $(y)$

$y = x$

print $(z)$
Understanding Strong Liveness

\[
\begin{align*}
y &= x \\
\{x\} \\
\text{print (x)} \\
\emptyset
\end{align*}
\]
Understanding Strong Liveness

\[ y = x \]
\[ \{ x \} \]
\[ \text{print} \ (x) \]
\[ \emptyset \]

\[ y = x \]
\[ \{ x \} \]
\[ \text{print} \ (y) \]

\[ y = x \]
\[ \text{print} \ (z) \]
Understanding Strong Liveness

Simple Liveness

\[ \{x\} \]
\[ y = x \]
\[ \{x\} \]
\[ \text{print } (x) \]
\[ \emptyset \]

Strong Liveness

\[ \{x\} \]
\[ y = x \]
\[ \{x\} \]
\[ \text{print } (y) \]
\[ \emptyset \]

\[ \{x\} \]
\[ y = x \]
\[ \{x\} \]
\[ \text{print } (z) \]
\[ \emptyset \]
Understanding Strong Liveness

Simple Liveness

\[ \{x\} \]

\[ y = x \]

\[ \{x\} \]

\[ \{x\} \]

\[ \text{print} (x) \]

\[ \emptyset \]

\[ \emptyset \]

Strong Liveness

\[ \{x\} \]

\[ y = x \]

\[ \{x\} \]

\[ \{x\} \]

\[ \text{print} (y) \]

\[ \text{Same} \]

\[ y = x \]

\[ \text{print} (z) \]
Understanding Strong Liveness

Simple Liveness

\[ \{x\} \]

\[ \{x\} \]

\[ y = x \]

\[ \{x\} \]

\[ \{x\} \]

\[ \text{print (x)} \]

\[ \emptyset \]

\[ \emptyset \]

Same

Strong Liveness

\[ y = x \]

\[ \{x\} \]

\[ \{x\} \]

\[ \text{print (y)} \]

Strong Liveness

\[ y = x \]

\[ \{x\} \]

\[ \{x\} \]

\[ \text{print (z)} \]
Understanding Strong Liveness

Simple Liveness

\{x\} \rightarrow \{x\}

\{x\} \rightarrow \{x\}

print (x)

\emptyset \rightarrow \emptyset

Strong Liveness

\{x\} \rightarrow \{x\}

\{x\} \rightarrow \{x\}

\{x\} \rightarrow \{x\}

print (x)

\emptyset \rightarrow \emptyset

Same

Strong Liveness

\{x\} \rightarrow \{x\}

\{x\} \rightarrow \{x\}

\{x\} \rightarrow \{x\}

print (y)

\emptyset \rightarrow \emptyset

Strong Liveness

\{x\} \rightarrow \{x\}

\{x\} \rightarrow \{x\}

\{x\} \rightarrow \{x\}

print (z)

\emptyset \rightarrow \emptyset

\emptyset \rightarrow \emptyset
Understanding Strong Liveness

Simple Liveness

\[
\{x\} \downarrow \{x\} \\
\begin{array}{c}
y = x \\
\{x\} \downarrow \{x\} \\
\text{print (x)} \\
\emptyset \downarrow \emptyset
\end{array}
\]

Same

Strong Liveness

\[
\{x\} \downarrow \{x\} \\
\begin{array}{c}
y = x \\
\{x\} \downarrow \{x\} \\
\text{print (y)} \\
\emptyset \downarrow \emptyset
\end{array}
\]

Strong Liveness

\[
\begin{array}{c}
y = x \\
\{y\} \downarrow \emptyset \\
\text{print (z)} \\
\end{array}
\]
Understanding Strong Liveness

Simple Liveness

\[
\{x\} \quad \xrightarrow{y=x} \quad \{x\} \\
\{x\} \quad \xrightarrow{\text{print } (x)} \quad \emptyset
\]

Strong Liveness

\[
\{x\} \quad \xrightarrow{y=x} \quad \{x\} \\
\{x\} \quad \xrightarrow{\text{print } (y)} \quad \emptyset
\]

Same

Strong Liveness

\[
\{x\} \quad \xrightarrow{y=x} \quad \{y\} \\
\{x\} \quad \xrightarrow{\text{print } (z)} \quad \emptyset
\]
Understanding Strong Liveness

Simple Liveness | Strong Liveness
---|---
\{x\} | \{x\}
---|---
y = x | y = x
---|---
\{x\} | \{x\}
---|---
print (x) | print (x)
---|---
\{\} | \{\}
---|---
Same |

Simple Liveness | Strong Liveness
---|---
\{x\} | \{x\}
---|---
y = x | y = x
---|---
\{x\} | \{y\}
---|---
\{\} | \{y\}
---|---
print (y) | print (y)
---|---
\{\} | \{\}
---|---

Simple Liveness | Strong Liveness
---|---
\{y\} | \{y\}
---|---
\{\} | \{\}
---|---

Simple Liveness | Strong Liveness
---|---
\{\} | \{\}
---|---
\{\} | \{\}
---|---

y = x
---|---
p |
---|---
p |
---|---
print (z) | print (z)
---|---

Same
Understanding Strong Liveness

Simple Liveness:
\[ \{ x \} \]
\[ y = x \]
\[ \{ x \} \]
\[ \text{print} \ (x) \]
\[ \emptyset \]

Strong Liveness:
\[ \{ x \} \]
\[ y = x \]
\[ \{ x \} \]
\[ \text{print} \ (x) \]
\[ \emptyset \]

Simple Liveness:
\[ \{ x \} \]
\[ y = x \]
\[ \{ y \} \]
\[ \text{print} \ (y) \]
\[ \emptyset \]

Strong Liveness:
\[ \{ x \} \]
\[ y = x \]
\[ \{ y \} \]
\[ \text{print} \ (y) \]
\[ \emptyset \]

Same

Same

\[ y = x \]
\[ \text{print} \ (z) \]
Understanding Strong Liveness

Simple Liveness | Strong Liveness
---|---
\{x\} | \{x\}
\{x\} | \{x\}
print (x) | print (x)
\emptyset | \emptyset

Same

Simple Liveness | Strong Liveness
---|---
\{x\} | \{x\}
\{x\} | \{y\}
print (y) | print (y)
\emptyset | \emptyset

Same

Strong Liveness

y = x

print (z)
Understanding Strong Liveness

Simple Liveness

\[
\{x\} \xrightarrow{y = x} \{x\} \xrightarrow{\text{print } (x)} \emptyset \]

Strong Liveness

\[
\{x\} \xrightarrow{y = x} \{x\} \xrightarrow{\text{print } (y)} \emptyset \]

Simple Liveness

\[
\{x\} \xrightarrow{y = x} \{x\} \xrightarrow{\text{print } (x)} \emptyset \]

Strong Liveness

\[
\{y\} \xrightarrow{y = x} \{y\} \xrightarrow{\text{print } (y)} \emptyset \]

Strong Liveness

\[
\{y\} \xrightarrow{y = x} \{y\} \xrightarrow{\text{print } (z)} \emptyset \]
Understanding Strong Liveness

Simple Liveness
- \( \{x\} \)
- \( y = x \)
- \( \{x\} \)
- \( \text{print } (x) \)
- \( \emptyset \)

Strong Liveness
- \( \{x\} \)
- \( y = x \)
- \( \{x\} \)
- \( \text{print } (x) \)
- \( \emptyset \)

Simple Liveness
- \( \{x\} \)
- \( y = x \)
- \( \{y\} \)
- \( \text{print } (y) \)
- \( \emptyset \)

Strong Liveness
- \( \{y\} \)
- \( \{y\} \)
- \( \emptyset \)

Simple Liveness
- \( \{x\} \)
- \( y = x \)
- \( \{z\} \)
- \( \text{print } (z) \)
- \( \emptyset \)

Strong Liveness
- \( y = x \)
- \( \{z\} \)
- \( \emptyset \)
Understanding Strong Liveness

Simple Liveness               | Strong Liveness
\{x\}                        | \{x\}
\{x\}                        | \{x\}
\(y = x\)                    |
\{x\}                        | \{x\}
\(\text{print } (x)\)        |
\\emptyset                    | \\emptyset

Same

Simple Liveness               | Strong Liveness
\{x\}                        | \{x\}
\{y\}                        | \{y\}
\(y = x\)                    |
\{y\}                        | \{y\}
\(\text{print } (y)\)        |
\\emptyset                    | \\emptyset

Same

Strong Liveness
\(y = x\)
\{z\}
\(\text{print } (z)\)
\\emptyset
**Understanding Strong Liveness**

<table>
<thead>
<tr>
<th>Simple Liveness</th>
<th>Strong Liveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x}</td>
<td>{x}</td>
</tr>
<tr>
<td><code>y = x</code></td>
<td><code>y = x</code></td>
</tr>
<tr>
<td>{x}</td>
<td>{y}</td>
</tr>
<tr>
<td><code>print (x)</code></td>
<td><code>print (y)</code></td>
</tr>
<tr>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
<tr>
<td>Same</td>
<td>Same</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simple Liveness</th>
<th>Strong Liveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x, y}</td>
<td>{z}</td>
</tr>
<tr>
<td><code>y = x</code></td>
<td><code>y = x</code></td>
</tr>
<tr>
<td>{y}</td>
<td>{z}</td>
</tr>
<tr>
<td><code>print (z)</code></td>
<td><code>print (z)</code></td>
</tr>
<tr>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
<tr>
<td>Same</td>
<td>Same</td>
</tr>
</tbody>
</table>
Understanding Strong Liveness

<table>
<thead>
<tr>
<th>Simple Liveness</th>
<th>Strong Liveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>${x}$</td>
<td>${x}$</td>
</tr>
<tr>
<td>${x}$</td>
<td>${x}$</td>
</tr>
<tr>
<td>$y = x$</td>
<td>$y = x$</td>
</tr>
<tr>
<td>$\varnothing$</td>
<td>$\varnothing$</td>
</tr>
<tr>
<td>$\varnothing$</td>
<td>$\varnothing$</td>
</tr>
<tr>
<td>Same</td>
<td>Same</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simple Liveness</th>
<th>Strong Liveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>${x}$</td>
<td>${x}$</td>
</tr>
<tr>
<td>${x}$</td>
<td>${x}$</td>
</tr>
<tr>
<td>$y = x$</td>
<td>$y = x$</td>
</tr>
<tr>
<td>$\varnothing$</td>
<td>$\varnothing$</td>
</tr>
<tr>
<td>$\varnothing$</td>
<td>$\varnothing$</td>
</tr>
<tr>
<td>Same</td>
<td>Same</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simple Liveness</th>
<th>Strong Liveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>${z, x}$</td>
<td>${z}$</td>
</tr>
<tr>
<td>${z}$</td>
<td>${z}$</td>
</tr>
<tr>
<td>$y = x$</td>
<td>$y = x$</td>
</tr>
<tr>
<td>$\varnothing$</td>
<td>$\varnothing$</td>
</tr>
<tr>
<td>$\varnothing$</td>
<td>$\varnothing$</td>
</tr>
<tr>
<td>Different</td>
<td></td>
</tr>
</tbody>
</table>
Live Variables Analysis: Simple and Strong Liveness

- A variable is live at a program point if its current value is likely to be used later
Live Variables Analysis: Simple and Strong Liveness

- A variable is live at a program point if its current value is likely to be used later.
- We want to compute the smallest set of variables that are live.
Live Variables Analysis: Simple and Strong Liveness

- A variable is live at a program point if its current value is likely to be used later.
- We want to compute the smallest set of variables that are live.

```
a = 1; b = 2
c = 3; n = 6
B1
if a ≤ n
B2
  a = a + 1
B3
  if a ≤ 11
B4
    t1 = a + b
    a = t1 + c
    print "Hello"
  B5
  print "Hi"
B6
```
Live Variables Analysis: Simple and Strong Liveness

- A variable is live at a program point if its current value is likely to be used later.
- We want to compute the smallest set of variables that are live.
- Simple liveness considers every use of a variable as useful.

```
a = 1; b = 2
c = 3; n = 6
B1

if a ≤ n
B2
    {a, b, c, n}
    {a, b, c, n}
    {a, b, c, n}
    {a, b, c, n}
    {a, b, c, n}
    {a, b, c, n}
    {a, b, c, n}

T
    a = a + 1
    B3
    {a, b, c, n}
    {a, b, c, n}

F
    if a ≤ 11
    {a, b, c}
    {a, b, c}
    {a, b, c}
    {a, b, c}
    {a, b, c}
    {a, b, c}

T
    t1 = a + b
    a = t1 + c
    print "Hello"
    B5
    {a, b, c}
    {a, b, c}
    {a, b, c}

F
    F
    print "Hi"
    B6
    ∅

∅
Live Variables Analysis: Simple and Strong Liveness

- A variable is live at a program point if its current value is likely to be used later.
- We want to compute the smallest set of variables that are live.
- Simple liveness considers every use of a variable as useful.
- Strong liveness checks the liveness of the result before declaring the operands to be live.

```
a = 1; b = 2
c = 3; n = 6  B1
if a <= n  B2
    a = a + 1  B3
if a <= 11  B4
    t1 = a + b
    a = t1 + c
    print "Hello"  B5
print "Hi"  B6
```
Live Variables Analysis: Simple and Strong Liveness

- A variable is live at a program point if its current value is likely to be used later.
- We want to compute the smallest set of variables that are live.
- Simple liveness considers every use of a variable as useful.
- Strong liveness checks the liveness of the result before declaring the operands to be live.
- Strong liveness is more precise than simple liveness.
Data Flow Equations for Strongly Live Variables Analysis

\[ In_n = f_n(Out_n) \]

\[ Out_n = \begin{cases} \text{BI} & n \text{ is } \text{End} \\ \bigcup_{s \in \text{succ}(n)} In_s & \text{otherwise} \end{cases} \]

where,

\[ f_n(X) = \begin{cases} (X - \{y\}) \cup (\text{Opd}(e) \cap \text{Var}) & n \text{ is } y = e, e \in \text{Expr}, y \in X \\ X - \{y\} & n \text{ is } \text{input}(y) \\ X \cup \{y\} & n \text{ is } \text{use}(y) \\ X & \text{otherwise} \end{cases} \]
Data Flow Equations for Strongly Live Variables Analysis

\[ \text{Out}_n = \begin{cases} \text{Bl} \cup \bigcup_{s \in \text{succ}(n)} \text{In}_s & \text{n is End} \\ \text{(else)} & \text{otherwise} \end{cases} \]

where,

\[ \text{In}_n = f_n(\text{Out}_n) \]

\[ f_n(X) = \begin{cases} (X - \{y\}) \cup (\text{Opd}(e) \cap \text{Var}) & \text{n is } y = e, e \in \text{Expr}, \ y \in X \\ X - \{y\} & \text{n is input}(y) \\ X \cup \{y\} & \text{n is use}(y) \\ X & \text{otherwise} \end{cases} \]

If \( y \) is not strongly live, the assignment is skipped using the “otherwise” clause.
Properties of Strongly Live Variable Analysis

- What is $\hat{L}$ for strongly live variables analysis?

- Is strongly live variables analysis a bit vector framework?

- Is strongly live variables analysis a separable framework?

- Is strongly live variables analysis distributive? Monotonic?
Properties of Strongly Live Variable Analysis

- What is $\hat{L}$ for strongly live variables analysis?
  - $\hat{L} = \{0, 1\}, 1 \sqsubseteq 0$

- Is strongly live variables analysis a bit vector framework?

- Is strongly live variables analysis a separable framework?

- Is strongly live variables analysis distributive? Monotonic?
Properties of Strongly Live Variable Analysis

• What is $\hat{L}$ for strongly live variables analysis?
  
  $\hat{L} = \{0, 1\}, 1 \sqsubseteq 0$

• Is strongly live variables analysis a bit vector framework?
  
  No because data flow equations cannot be defined only in terms of bit vector operations

• Is strongly live variables analysis a separable framework?

• Is strongly live variables analysis distributive? Monotonic?
Properties of Strongly Live Variable Analysis

- What is $\hat{L}$ for strongly live variables analysis?
  - $\hat{L} = \{0, 1\}, 1 \sqsubseteq 0$

- Is strongly live variables analysis a bit vector framework?
  - No because data flow equations cannot be defined only in terms of bit vector operations

- Is strongly live variables analysis a separable framework?
  - No, because strong liveness of variables occurring in RHS of an assignment may depend on the variable occurring in LHS

- Is strongly live variables analysis distributive? Monotonic?
Properties of Strongly Live Variable Analysis

- What is $\hat{L}$ for strongly live variables analysis?
  - $\hat{L} = \{0, 1\}, 1 \sqsubseteq 0$

- Is strongly live variables analysis a bit vector framework?
  - No because data flow equations cannot be defined only in terms of bit vector operations

- Is strongly live variables analysis a separable framework?
  - No, because strong liveness of variables occurring in RHS of an assignment may depend on the variable occurring in LHS

- Is strongly live variables analysis distributive? Monotonic?
  - Distributive, and hence monotonic
Distributivity of Strongly Live Variables Analysis (1)

We need to prove that

$$\forall X_1, X_2 \in L, \ f_n(X_1 \cup X_2) = f_n(X_1) \cup f_n(X_2)$$
Distributivity of Strongly Live Variables Analysis (1)

We need to prove that

$$\forall X_1, X_2 \in L, \ f_n(X_1 \cup X_2) = f_n(X_1) \cup f_n(X_2)$$

- Intuitively,
  - The value does not depend on the argument $X$
  - Incomparable results cannot be produced
    (A fixed set of variable are excluded or included)
Distributivity of Strongly Live Variables Analysis (1)

We need to prove that

\[ \forall X_1, X_2 \in L, f_n(X_1 \cup X_2) = f_n(X_1) \cup f_n(X_2) \]

• Intuitively,
  ▶ The value does not depend on the argument \( X \)
  ▶ Incomparable results cannot be produced
    (A fixed set of variable are excluded or included)

• Formally,
  ▶ We prove it for \( \text{input}(y), \text{use}(y), y = e \), and empty statements independently
Distributivity of Strongly Live Variables Analysis (2)

- For $\text{input}(y)$ statement:

- For $\text{use}(y)$ statement:

- For empty statement:
Distributivity of Strongly Live Variables Analysis (2)

- For input(y) statement: \[ f_n(X_1 \cup X_2) = (X_1 \cup X_2) - \{y\} = (X_1 - \{y\}) \cup (X_2 - \{y\}) = f_n(X_1) \cup f_n(X_2) \]

- For use(y) statement:

- For empty statement:
Distributivity of Strongly Live Variables Analysis (2)

- For `input(y)` statement: \[ f_n(X_1 \cup X_2) = (X_1 \cup X_2) - \{y\} \]
  \[= (X_1 - \{y\}) \cup (X_2 - \{y\}) \]
  \[= f_n(X_1) \cup f_n(X_2) \]

- For `use(y)` statement: \[ f_n(X_1 \cup X_2) = (X_1 \cup X_2) \cup \{y\} \]
  \[= (X_1 \cup \{y\}) \cup (X_2 \cup \{y\}) \]
  \[= f_n(X_1) \cup f_n(X_2) \]

- For empty statement:
Distributivity of Strongly Live Variables Analysis (2)

- For \(\text{input}(y)\) statement: 
  \[
  f_n(X_1 \cup X_2) = (X_1 \cup X_2) - \{y\} \\
  = (X_1 - \{y\}) \cup (X_2 - \{y\}) \\
  = f_n(X_1) \cup f_n(X_2)
  \]

- For \(\text{use}(y)\) statement: 
  \[
  f_n(X_1 \cup X_2) = (X_1 \cup X_2) \cup \{y\} \\
  = (X_1 \cup \{y\}) \cup (X_2 \cup \{y\}) \\
  = f_n(X_1) \cup f_n(X_2)
  \]

- For empty statement: 
  \[
  f_n(X_1 \cup X_2) = X_1 \cup X_2 = f_n(X_1) \cup f_n(X_2)
  \]
Distributivity of Strongly Live Variables Analysis (3)

For $y = e$ statement: Let $Y = Opd(e) \cap \mathbb{Var}$. There are three cases:

- $y \in X_1, y \in X_2$.

- $y \in X_1, y \not\in X_2$.

- $y \not\in X_1, y \not\in X_2$. 

Sep 2017 IIT Bombay
Distributivity of Strongly Live Variables Analysis (3)

For $y = e$ statement: Let $Y = Opd(e) \cap \mathbb{V}ar$. There are three cases:

- $y \in X_1, y \in X_2$.

$$f_n(X_1 \cup X_2) = ((X_1 \cup X_2) - \{y\}) \cup Y$$

$$= (X_1 - \{y\}) \cup (X_2 - \{y\}) \cup Y$$

$$= ((X_1 - \{y\}) \cup Y) \cup ((X_2 - \{y\}) \cup Y)$$

$$= f_n(X_1) \cup f_n(X_2)$$

- $y \in X_1, y \notin X_2$.

- $y \notin X_1, y \notin X_2$. 
Distributivity of Strongly Live Variables Analysis (3)

For \( y = e \) statement: Let \( Y = Opd(e) \cap \mathbb{Var} \). There are three cases:

- \( y \in X_1, y \in X_2 \).

\[
f_n(X_1 \cup X_2) = ((X_1 \cup X_2) - \{y\}) \cup Y
= (X_1 - \{y\}) \cup (X_2 - \{y\}) \cup Y
= ((X_1 - \{y\}) \cup Y) \cup ((X_2 - \{y\}) \cup Y)
= f_n(X_1) \cup f_n(X_2)
\]

- \( y \in X_1, y \not\in X_2 \).

\[
f_n(X_1 \cup X_2) = ((X_1 \cup X_2) - \{y\}) \cup Y
= ((X_1 - \{y\}) \cup Y) \cup (X_2) \quad (\because y \not\in X_2)
= f_n(X_1) \cup f_n(X_2)
\]

- \( y \not\in X_1, y \not\in X_2 \).
Distributivity of Strongly Live Variables Analysis (3)

For \( y = e \) statement: Let \( Y = Opd(e) \cap \mathbb{V}ar \). There are three cases:

- \( y \in X_1, y \in X_2 \).
  \[
  f_n(X_1 \cup X_2) = \left( (X_1 \cup X_2) - \{y\} \right) \cup Y \\
  = (X_1 - \{y\}) \cup (X_2 - \{y\}) \cup Y \\
  = ((X_1 - \{y\}) \cup Y) \cup ((X_2 - \{y\}) \cup Y) \\
  = f_n(X_1) \cup f_n(X_2)
  \]

- \( y \in X_1, y \notin X_2 \).
  \[
  f_n(X_1 \cup X_2) = \left( (X_1 \cup X_2) - \{y\} \right) \cup Y \\
  = ((X_1 - \{y\}) \cup Y) \cup (X_2) \\
  = f_n(X_1) \cup f_n(X_2) \quad (\because y \notin X_2) \\
  = f_n(X_1) \cup f_n(X_2) \quad y \notin X_2 \Rightarrow f_n(X_2) \text{ is identity}
  \]

- \( y \notin X_1, y \notin X_2 \).
  \[
  f_n(X_1 \cup X_2) = X_1 \cup X_2 = f_n(X_1) \cup f_n(X_2)
  \]
Tutorial Problem for strongly Live Variables Analysis

\[
\begin{align*}
&n_1: a = 0 \\
&n_2: \text{if } a \geq 2 \\
&n_3: a = b \\
&n_4: b = c \\
&n_5: a = a + 1 \\
&n_6: \\
\end{align*}
\]
## Result of Strongly Live Variables Analysis

<table>
<thead>
<tr>
<th>Node</th>
<th>Iteration #1</th>
<th>Iteration #2</th>
<th>Iteration #3</th>
<th>Iteration #4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$Out_n$</td>
<td>$In_n$</td>
<td>$Out_n$</td>
<td>$In_n$</td>
</tr>
<tr>
<td>$n_6$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$n_5$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>${a}$</td>
<td>${a}$</td>
</tr>
<tr>
<td>$n_4$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>${a}$</td>
<td>${a}$</td>
</tr>
<tr>
<td>$n_3$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>${a}$</td>
<td>${b}$</td>
</tr>
<tr>
<td>$n_2$</td>
<td>$\emptyset$</td>
<td>${a}$</td>
<td>${a, b}$</td>
<td>${a, b}$</td>
</tr>
<tr>
<td>$n_1$</td>
<td>${a}$</td>
<td>$\emptyset$</td>
<td>${a, b}$</td>
<td>${b}$</td>
</tr>
</tbody>
</table>
Tutorial Problem: Strongly May-Must Liveness Analysis?

- Instead of viewing liveness information as
  - a map $\mathcal{V}ar \to \{0, 1\}$ with the lattice $\{0, 1\}$,
  view it as
  - a map $\mathcal{V}ar \to \hat{L}$ where $\hat{L}$ is the May-Must Lattice

- Write the data flow equations

- Prove that the flow functions are distributive
Part 5

Pointer Analyses
An Outline of Pointer Analysis Coverage

• The larger perspective
• Comparing Points-to and Alias information
• Flow Insensitive Points-to Analysis
• Flow Sensitive Points-to Analysis
• Pointer Analyses: An Engineer’s Landscape
• Liveness Based Points-to Analysis
• Generalizations to Heap, Arrays, Pointer Arithmetic, and Unions
Code Optimization In Presence of Pointers

Program

1. \( q = p; \)
2. while (\ldots) {
3. \quad q = q\rightarrow next;
4. }
5. \( p\rightarrow data = r1; \)
6. print (\( q\rightarrow data \));
7. \( p\rightarrow data = r2; \)

Memory graph at statement 5

- Is \( p\rightarrow data \) live at the exit of line 5? Can we delete line 5?
## Code Optimization In Presence of Pointers

<table>
<thead>
<tr>
<th>Program</th>
<th>Memory graph at statement 5</th>
</tr>
</thead>
</table>
| 1. \( q = p; \)  
2. do \{
3. \( q = q \rightarrow \text{next}; \)
4. while (…)  
5. \( p \rightarrow \text{data} = r1; \)
6. print \( (q \rightarrow \text{data}); \)
7. \( p \rightarrow \text{data} = r2; \) |  
\[ q \]  
\[ p \rightarrow \text{next} \rightarrow \text{next} \rightarrow \ldots \] |

- Is \( p \rightarrow \text{data} \) live at the exit of line 5? Can we delete line 5?
Code Optimization In Presence of Pointers

Program

1. \( q = p; \)
2. do {
3. \( q = q \rightarrow \text{next}; \)
4. while ( . . . )
5. \( p \rightarrow \text{data} = r1; \)
6. print \( (q \rightarrow \text{data}); \)
7. \( p \rightarrow \text{data} = r2; \)

---

Memory graph at statement 5

- Is \( p \rightarrow \text{data} \) live at the exit of line 5? Can we delete line 5?
- We cannot delete line 5 if \( p \) and \( q \) can be possibly aliased (while loop or do-while loop with a circular list).
### Code Optimization In Presence of Pointers

#### Program

1. \( q = p; \)
2. \( \text{do} \)
   
   \( q = q \rightarrow \text{next}; \)
3. \( \text{while} \) ( . . . )
4. \( p \rightarrow \text{data} = r1; \)
5. \( \text{print} (q \rightarrow \text{data}); \)
6. \( p \rightarrow \text{data} = r2; \)

#### Memory graph at statement 5

- Is \( p \rightarrow \text{data} \) live at the exit of line 5? Can we delete line 5?

- We cannot delete line 5 if \( p \) and \( q \) can be possibly aliased (while loop or do-while loop with a circular list)

- We can delete line 5 if \( p \) and \( q \) are definitely not aliased (do-while loop without a circular list)
Code Optimization In Presence of Pointers

Original Program

\[
\begin{align*}
    a &= 5 \\
    x &= & a \\
    b &= & x
\end{align*}
\]
Code Optimization In Presence of Pointers

Original Program

```
a = 5
x = &a
b = *x
```

Constant Propagation without aliasing

```
a = 5
x = &a
b = *x
```
Code Optimization In Presence of Pointers

Original Program  

```
a = 5
x = &a
b = *x
```

Constant Propagation without aliasing  

```
a = 5
x = &a
b = *x
```

Constant Propagation with aliasing  

```
a = 5
x = &a
b = 5
```
The World of Pointer Analysis

Alias Analysis

- Alias analysis of reference parameters
- Fields of unions
- Array indices

Pointer Analysis

- Alias analysis of data pointers
- Points-to analysis of data and function pointers
Pointer Analysis Musings

- Pointer analysis collects information about indirect accesses in programs
  - Enables precise data analysis
  - Enables precise interprocedural control flow analysis
- Needs to scale to large programs
- Pointer Analysis Musings
  - Which Pointer Analysis should I Use?
    Michael Hind and Anthony Pioli. ISTAA 2000
  - Pointer Analysis: Haven’t we solved this problem yet?
    Michael Hind PASTE 2001
Pointer Analysis Musings

- Pointer analysis collects information about indirect accesses in programs
  - Enables precise data analysis
  - Enable precise interprocedural control flow analysis
- Needs to scale to large programs
- Pointer Analysis Musings
  - Which Pointer Analysis should I Use?
    Michael Hind and Anthony Pioli. ISTAA 2000
  - Pointer Analysis: Haven’t we solved this problem yet?
    Michael Hind PASTE 2001
Pointer Analysis Musings

- Pointer analysis collects information about indirect accesses in programs
  - Enables precise data analysis
  - Enable precise interprocedural control flow analysis

- Needs to scale to large programs

- Pointer Analysis Musings
  - Which Pointer Analysis should I Use?
    Michael Hind and Anthony Pioli. ISTAA 2000
  - Pointer Analysis: Haven’t we solved this problem yet?
    Michael Hind PASTE 2001
  - 2017 .. 😞
The Mathematics of Pointer Analysis

In the most general situation

- Alias analysis is undecidable.

- Flow insensitive alias analysis is NP-hard
  Horwitz [TOPLAS 1997]

- Points-to analysis is undecidable
  Chakravarty [POPL 2003]
The Mathematics of Pointer Analysis

In the most general situation

- Alias analysis is undecidable.
  Landi-Ryder [POPL 1991], Landi [LOPLAS 1992],
  Ramalingam [TOPLAS 1994]

- Flow insensitive alias analysis is NP-hard
  Horwitz [TOPLAS 1997]

- Points-to analysis is undecidable
  Chakravarty [POPL 2003]

*Adjust your expectations suitably to avoid disappointments!*
So what should we expect?
The Engineering of Pointer Analysis

So what should we expect? To quote Hind [PASTE 2001]
So what should we expect? To quote Hind [PASTE 2001]

- “Fortunately many approximations exist”
So what should we expect? To quote Hind [PASTE 2001]

- “Fortunately many approximations exist”
- “Unfortunately too many approximations exist!”
The Engineering of Pointer Analysis

So what should we expect? To quote Hind [PASTE 2001]

- “Fortunately many approximations exist”
- “Unfortunately too many approximations exist!”

*Engineering of pointer analysis is much more dominant than its science*
Pointer Analysis: Engineering or Science?

• Engineering view.
  ► Build quick approximations
  ► The tyranny of (exclusive) OR!
  Precision OR Efficiency?

• Science view.
  ► Build clean abstractions
  ► Can we harness the Genius of AND?
  Precision AND Efficiency?
Parser Analysis: Engineering or Science?

- Engineering view. ► Build quick approximations
  ► The tyranny of (exclusive) OR!
  Precision OR Efficiency?

- Science view. ► Build clean abstractions
  ► Can we harness the Genius of AND?
  Precision AND Efficiency?

- A distinction between approximation and abstraction is subjective
  Our working definition
Pointer Analysis: Engineering or Science?

- Engineering view.
  - Build quick approximations
  - The tyranny of (exclusive) OR!
  Precision OR Efficiency?

- Science view.
  - Build clean abstractions
  - Can we harness the Genius of AND?
  Precision AND Efficiency?

- A distinction between approximation and abstraction is subjective
  Our working definition
  - Abstractions focus on precision and conciseness of modelling
  - Approximations focus on efficiency and scalability
An Outline of Pointer Analysis Coverage

- The larger perspective

- **Comparing Points-to and Alias information**

- Flow Insensitive Points-to Analysis

- Flow Sensitive Points-to Analysis

- Pointer Analyses: An Engineer’s Landscape

- Liveness Based Points-to Analysis

- Generalizations to Heap, Arrays, Pointer Arithmetic, and Unions
Alias Information Vs. Points-to Information

1. $x = \&a$
2. $b = x$
Alias Information Vs. Points-to Information

```
1. x = &a
2. b = x

"x Points-to a"
denoted x → a
```
Alias Information Vs. Points-to Information

1. \( x = \&a \)
   - "\( x \) Points-to \( a \)"
   - denoted \( x \rightarrow a \)

2. \( b = x \)
   - "\( x \) and \( b \) are Aliases"
   - denoted \( x \bowtie b \)
Alias Information Vs. Points-to Information

1. \( x = \&a \) denoted \( x \rightarrow a \)

2. \( b = x \) denoted \( x = b \)

\( x \) and \( b \) are Aliases

Symmetric and Reflexive
Alias Information Vs. Points-to Information

1. \[ x = \&a \]
   - "x Points-to a" denoted \( x \rightarrow a \)
   - Neither Symmetric Nor Reflexive

2. \[ b = x \]
   - "x and b are Aliases" denoted \( x \bowtie b \)
   - Symmetric and Reflexive
Alias Information Vs. Points-to Information

1. \( x = \&a \), denoted \( x \mapsto a \)
   - "\( x \) Points-to \( a \)"
   - Neither Symmetric Nor Reflexive

2. \( b = x \), denoted \( x \equiv b \)
   - "\( x \) and \( b \) are Aliases"
   - Symmetric and Reflexive

- What about transitivity?
Alias Information Vs. Points-to Information

1. \( x = \&a \) denoted \( x \rightarrow a \)

2. \( b = x \) denoted \( x \equiv b \)

- What about transitivity?
  - Points-to: No.
Alias Information Vs. Points-to Information

1. \( a \) is allocated.
2. \( x = \&a \)
3. \( b = x \)

- “\( x \) Points-to \( a \)” denoted \( x \rightarrow a \)
- “\( x \) and \( b \) are Aliases” denoted \( x \sqsupset b \)

- What about transitivity?
  - Points-to: No.
  - Alias: Depends.

Neither Symmetric Nor Reflexive
Symmetric and Reflexive
# Comparing Points-to and Alias Relations (1)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Memory</th>
<th>Points-to</th>
<th>Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = &amp; y$</td>
<td>Before (assume)</td>
<td>Existing</td>
<td>Existing</td>
</tr>
<tr>
<td></td>
<td>After</td>
<td>New</td>
<td>New Direct</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$x \leadsto y$</td>
<td>$x \bowtie &amp; y$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>Before (assume)</td>
<td>Existing</td>
<td>Existing</td>
</tr>
<tr>
<td></td>
<td>After</td>
<td>New</td>
<td>New Direct</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$x \leadsto z$</td>
<td>$x \bowtie &amp; z$</td>
</tr>
</tbody>
</table>

Sep 2017

IIT Bombay
### Comparing Points-to and Alias Relations (1)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Memory</th>
<th>Points-to</th>
<th>Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = &amp; y$</td>
<td>Before (assume)</td>
<td></td>
<td>Existing</td>
</tr>
<tr>
<td></td>
<td>After</td>
<td>$\mathbf{x}$</td>
<td>Existing</td>
</tr>
<tr>
<td>$x = y$</td>
<td>Before (assume)</td>
<td>$\mathbf{x}$ $y$ $z$</td>
<td>Existing $y$ $z$</td>
</tr>
<tr>
<td></td>
<td>After</td>
<td>$\mathbf{x}$ $y$ $z$</td>
<td>New Direct $x$ $\circ$ $&amp; y$</td>
</tr>
</tbody>
</table>

- Indirect aliases. Substitute a name by its aliases for transitivity.
Comparing Points-to and Alias Relations (1)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Memory</th>
<th>Points-to</th>
<th>Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = &amp; y$</td>
<td><img src="image1" alt="Memory Diagram" /></td>
<td>Existing</td>
<td>Existing</td>
</tr>
<tr>
<td>Before (assume)</td>
<td>$x \rightarrow y$</td>
<td>New</td>
<td>New Direct $x \equiv &amp; y$</td>
</tr>
<tr>
<td>After</td>
<td>$x \bullet &amp; y$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x = y$</td>
<td><img src="image2" alt="Memory Diagram" /></td>
<td>Existing</td>
<td>$y \equiv &amp; z$</td>
</tr>
<tr>
<td>Before (assume)</td>
<td>$y \rightarrow z$</td>
<td>New Direct $x \equiv y$</td>
<td></td>
</tr>
<tr>
<td>After</td>
<td>$x \bullet &amp; y \bullet z$</td>
<td>New</td>
<td>New Indirect $x \equiv &amp; z$</td>
</tr>
<tr>
<td></td>
<td>$x \rightarrow z$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Indirect aliases. Substitute a name by its aliases for transitivity
- Derived aliases. Apply indirection operator to aliases (ignored here)

$x \equiv y \Rightarrow \star x \equiv \star y$
Comparing Points-to and Alias Relations (2)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Memory</th>
<th>Points-to</th>
<th>Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>( *x = y )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( x = *y )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Comparing Points-to and Alias Relations (2)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Memory</th>
<th>Points-to</th>
<th>Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>( *x = y )</td>
<td>Before (assume)</td>
<td>( x \mapsto y ) ( y \mapsto z )</td>
<td>( x = &amp; u ) ( y = &amp; z )</td>
</tr>
<tr>
<td>( x = *y )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Sep 2017

IIT Bombay
### Comparing Points-to and Alias Relations (2)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Memory</th>
<th>Points-to</th>
<th>Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>( *x = y )</td>
<td><img src="image" alt="Memory Diagram" /></td>
<td><img src="image" alt="" /></td>
<td><img src="image" alt="" /></td>
</tr>
<tr>
<td>Before (assume)</td>
<td>Existing</td>
<td>Existing ( x \equiv &amp;u )( y \equiv &amp;z )</td>
<td>New Direct ( *x \equiv y )</td>
</tr>
<tr>
<td>After</td>
<td>New</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( x = *y )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sep 2017

IIT Bombay
## Comparing Points-to and Alias Relations (2)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Memory</th>
<th>Points-to</th>
<th>Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>x = y</strong></td>
<td><img src="image" alt="Before Diagram" /></td>
<td><img src="image" alt="Existing Points-to" /></td>
<td><strong>Existing</strong></td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="After Diagram" /></td>
<td><img src="image" alt="New Points-to" /></td>
<td><strong>New Direct</strong></td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Existing Alias" /></td>
<td><img src="image" alt="New Indirect" /></td>
<td><strong>New Indirect</strong></td>
</tr>
<tr>
<td>*<em>x = <em>y</em></em></td>
<td><img src="image" alt="Before Alias" /></td>
<td><img src="image" alt="Existing Alias" /></td>
<td><strong>Existing</strong></td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="After Alias" /></td>
<td><img src="image" alt="New Alias" /></td>
<td><strong>New Alias</strong></td>
</tr>
</tbody>
</table>

*Comparing Points-to and Alias Relations (2)*

- **Before (assume)**
  - Memory: `x`, `y`, `z`, `u`
  - Points-to: `x` points to `u`, `y` points to `z`
  - Aliases: `x` alias `&u`, `y` alias `&z`

- **After**
  - Memory: `x`, `y`, `z`, `u`
  - Points-to: `x` points to `y`, `y` points to `z`, `u` points to `z`
  - Aliases: `x` alias `&u`, `y` alias `&z`

- **Existing**
  - `x` alias `&u`
  - `y` alias `&z`

- **New Direct**
  - `x` alias `y`

- **New Indirect**
  - `u` alias `&z`
  - `y` alias `u`
  - `x` alias `&z`
### Comparing Points-to and Alias Relations (2)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Memory</th>
<th>Points-to</th>
<th>Aliases</th>
</tr>
</thead>
</table>
| $*x = y$        | Before (assume) | Existing | $x \triangleright u$
|                 |        | $y \triangleright z$ | $x \triangleright & u$
|                 | After  | New       | $z \triangleright & u$
|                 |        | $u \triangleright z$ | $y \triangleright & z$
| $x = *y$        | Before (assume) | Existing | $y \triangleright z$
|                 |        | $z \triangleright u$ | $y \triangleright & z$
|                 |        |            | $*y \triangleright & u$
## Comparing Points-to and Alias Relations (2)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Memory</th>
<th>Points-to</th>
<th>Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>x = y</strong> (assume)</td>
<td><img src="image1" alt="Before" /></td>
<td><strong>Existing</strong></td>
<td><strong>New Direct</strong></td>
</tr>
<tr>
<td><em>x = y</em> (assume)</td>
<td><img src="image2" alt="After" /></td>
<td><img src="image3" alt="Existing" /></td>
<td><img src="image4" alt="New Direct" /></td>
</tr>
</tbody>
</table>

| **x = *y** (assume) | ![Before](image5) | ![Existing](image6) | ![New Direct](image7) |
| **x = *y** (assume) | ![After](image8) | ![Existing](image9) | ![New Direct](image10) |
## Comparing Points-to and Alias Relations (2)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Memory</th>
<th>Points-to</th>
<th>Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \ast x = y )</td>
<td>Before (assume)</td>
<td>Existing</td>
<td>( x \sqsupset u )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( y \sqsupset z )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New</td>
<td>( u \sqsupset z )</td>
</tr>
<tr>
<td>After</td>
<td></td>
<td></td>
<td>( x \sqsupset &amp; u )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( y \sqsupset &amp; z )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( x \sqsupset &amp; u )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( y \sqsupset &amp; z )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Statement</th>
<th>Memory</th>
<th>Points-to</th>
<th>Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = \ast y )</td>
<td>Before (assume)</td>
<td>Existing</td>
<td>( y \sqsupset z )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( z \sqsupset &amp; u )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New</td>
<td>( \ast y \sqsupset &amp; u )</td>
</tr>
<tr>
<td>After</td>
<td></td>
<td></td>
<td>( x \sqsupset \ast y )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( x \sqsupset &amp; u )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( x \sqsupset &amp; z )</td>
</tr>
</tbody>
</table>
### Comparing Points-to and Alias Relations (2)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Memory</th>
<th>Points-to</th>
<th>Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>( *x = y )</td>
<td>Before (assume)</td>
<td>Existing</td>
<td>( x \prec &amp; u ) ( y \prec &amp; z )</td>
</tr>
<tr>
<td></td>
<td>After</td>
<td>New Direct</td>
<td>( x \prec *y )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New Indirect</td>
<td>( x \prec &amp; u ) ( x \prec z )</td>
</tr>
<tr>
<td>( x = *y )</td>
<td>Before (assume)</td>
<td>Existing</td>
<td>( y \prec &amp; z ) ( z \prec &amp; u ) ( *y \prec &amp; u )</td>
</tr>
<tr>
<td></td>
<td>After</td>
<td>New Direct</td>
<td>( x \prec &amp; u )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New Indirect</td>
<td>( x \prec &amp; u ) ( x \prec z )</td>
</tr>
</tbody>
</table>

The resulting memories look similar but are different. In the first case we have \( u \prec z \) whereas in the second case the arrow direction is opposite (i.e. \( z \prec u \)).
Comparing Points-to and Alias Relations (3)

- Points-to information records edges in the memory graph

- Alias information records paths in the memory graph
Comparing Points-to and Alias Relations (3)

• Points-to information records edges in the memory graph
  ▶ aliases of the kind $x \equiv &y$
   $x$ holds the address of $y$

• Alias information records paths in the memory graph
  ▶ paths incident on the same node
   $x$ and $y$ hold the same address (and the address is left implicit)
Comparing Points-to and Alias Relations (3)

• Points-to information records edges in the memory graph
  ▶ aliases of the kind $x \equiv &y$
    $x$ holds the address of $y$
  ▶ other aliases can be discovered by composing edges

• Alias information records paths in the memory graph
  ▶ paths incident on the same node
    $x$ and $y$ hold the same address (and the address is left implicit)
Comparing Points-to and Alias Relations (3)

- Points-to information records edges in the memory graph
  - aliases of the kind $x \equiv &y$
  - $x$ holds the address of $y$
  - other aliases can be discovered by composing edges
  - since addresses are explicated, it can represent only those memory locations that can be named at compile time

- Alias information records paths in the memory graph
  - paths incident on the same node
    - $x$ and $y$ hold the same address (and the address is left implicit)
  - since addresses are implicit, it can represent unnamed memory locations too
Comparing Points-to and Alias Relations (3)

- Points-to information records edges in the memory graph
  - aliases of the kind $x \equiv \& y$
    - $x$ holds the address of $y$
  - other aliases can be discovered by composing edges
  - since addresses are explicated, it can represent only those memory locations that can be named at compile time

- Alias information records paths in the memory graph
  - paths incident on the same node
    - $x$ and $y$ hold the same address (and the address is left implicit)
  - since addresses are implicit, it can represent unnamed memory locations too
  - if we have $x \equiv y$ then $\star x \equiv \star y$ is redundant and is not recorded
Comparing Points-to and Alias Relations (3)

- Points-to information records edges in the memory graph
  - aliases of the kind \( x \equiv &y \)
    - \( x \) holds the address of \( y \)
  - other aliases can be discovered by composing edges
  - since addresses are explicated, it can represent only those memory locations that can be named at compile time

  More compact but less general

- Alias information records paths in the memory graph
  - paths incident on the same node
    - \( x \) and \( y \) hold the same address (and the address is left implicit)
  - since addresses are implicit, it can represent unnamed memory locations too
  - if we have \( x \equiv y \) then \( \ast x \equiv \ast y \) is redundant and is not recorded

  More general and more complex
An Outline of Pointer Analysis Coverage

- The larger perspective
- Comparing Points-to and Alias information
- Flow Insensitive Points-to Analysis
  - Next Topic
- Flow Sensitive Points-to Analysis
- Pointer Analyses: An Engineer’s Landscape
- Liveness Based Points-to Analysis
- Generalizations to Heap, Arrays, Pointer Arithmetic, and Unions
Flow Sensitive Vs. Flow Insensitive Pointer Analysis

- Flow insensitive pointer analysis
  - Inclusion based: Andersen’s approach
  - Equality based: Steensgaard’s approach

- Flow sensitive pointer analysis
  - May points-to analysis
  - Must points-to analysis
Flow Insensitivity in Data Flow Analysis

• Assumption: Statements can be executed in any order.

• Instead of computing point-specific data flow information, summary data flow information is computed. The summary information is required to be a safe approximation of point-specific information for each point.

• Kill$_n$(X) component is ignored. If statement $n$ kills data flow information, there is an alternate path that excludes $n$.

The control flow graph is a complete graph (except for the Start and End nodes)
Flow Insensitivity in Data Flow Analysis

Assuming that there are no dependent parts in Gen$_n$ and Kill$_n$ is ignored

Control flow graph

Flow insensitive analysis
Flow Insensitivity in Data Flow Analysis

Assuming that there are no dependent parts in $\text{Gen}_n$ and $\text{Kill}_n$ is ignored.

Control flow graph

Flow insensitive analysis

*Function composition is replaced by function confluence*
Examples of Flow Insensitive Analyses
Examples of Flow Insensitive Analyses

- Type checking/inferencing
  (What about interpreted languages?)
Examples of Flow Insensitive Analyses

- Type checking/inferencing
  (What about interpreted languages?)
- Address taken analysis
  Which variables have their addresses taken?
Examples of Flow Insensitive Analyses

- Type checking/inferencing
  (What about interpreted languages?)

- Address taken analysis
  Which variables have their addresses taken?

- Side effects analysis
  Does a procedure modify a global variable? Reference Parameter?
Flow Insensitivity in Data Flow Analysis

Assuming $\text{Gen}_n(X)$ has dependent parts and $\text{Kill}_n(X)$ is ignored
Flow Insensitivity in Data Flow Analysis

Assuming $\text{Gen}_n(X)$ has dependent parts and $\text{Kill}_n(X)$ is ignored
Flow Insensitivity in Data Flow Analysis

Assuming \( \text{Gen}_n(X) \) has dependent parts and \( \text{Kill}_n(X) \) is ignored.

Allows arbitrary compositions of flow functions in any order
⇒ Flow insensitivity
Flow Insensitivity in Data Flow Analysis

Assuming $\text{Gen}_n(X)$ has dependent parts and $\text{Kill}_n(X)$ is ignored

Examples of dependent parts in $\text{Gen}$

- Pointer analysis for statements
  - $x = y$, $x = *y$, $*x = y$
Flow Insensitivity in Data Flow Analysis

Assuming $Gen_n(X)$ has dependent parts and $Kill_n(X)$ is ignored.

In practice, dependent constraints are collected in a global repository in one pass and then are solved independently.
Notation for Andersen’s and Steensgaard’s Points-to Analysis

- $P_x$ denotes the set of pointees of pointer variable $x$
- $\text{Unify}(x, y)$ unifies locations $x$ and $y$
  - $x$ and $y$ are treated as equivalent locations
  - the pointees of the unified locations are also unified transitively
- $\text{UnifyPTS}(x, y)$ unifies the pointees of $x$ and $y$
  - $x$ and $y$ themselves are not unified
Andersen’s and Steensgaard’s Points-to Analysis

<table>
<thead>
<tr>
<th>Statement</th>
<th>Andersen’s Points-to Sets</th>
<th>Steensgaard’s Points-to Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = &amp; y )</td>
<td>( P_x \supseteq {y} )</td>
<td>( P_x \supseteq {y} ) for some ( z \in P_x )</td>
</tr>
<tr>
<td>( x = y )</td>
<td>( P_x \supseteq P_y )</td>
<td>( \text{UnifyPTS}(x, y) )</td>
</tr>
<tr>
<td>( x = *y )</td>
<td>( P_x \supseteq P_z, \forall z \in P_y )</td>
<td>( \forall z \in P_y, \text{UnifyPTS}(x, z) )</td>
</tr>
<tr>
<td>( *x = y )</td>
<td>( P_z \supseteq P_y, \forall z \in P_x )</td>
<td>( \forall z \in P_x, \text{UnifyPTS}(y, z) )</td>
</tr>
</tbody>
</table>
## Andersen’s and Steensgaard’s Points-to Analysis

<table>
<thead>
<tr>
<th>Statement</th>
<th>Andersen’s Points-to Sets</th>
<th>Steensgaard’s Points-to Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = &amp;\ y )</td>
<td>( P_x \supseteq { y } )</td>
<td>( P_x \supseteq { y } ) for some ( z \in P_x )</td>
</tr>
<tr>
<td>( x = y )</td>
<td>( P_x \supseteq P_y )</td>
<td>( \text{UnifyPTS}(x, y) )</td>
</tr>
<tr>
<td>( x = *y )</td>
<td>( P_x \supseteq P_z, \forall z \in P_y )</td>
<td>( \forall z \in P_y, \text{UnifyPTS}(x, z) )</td>
</tr>
<tr>
<td>( *x = y )</td>
<td>( P_z \supseteq P_y, \forall z \in P_x )</td>
<td>( \forall z \in P_x, \text{UnifyPTS}(y, z) )</td>
</tr>
</tbody>
</table>

**Andersen’s view**

**Steensgaard’s view**
## Andersen’s and Steensgaard’s Points-to Analysis

<table>
<thead>
<tr>
<th>Statement</th>
<th>Andersen’s Points-to Sets</th>
<th>Steensgaard’s Points-to Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = &amp;y$</td>
<td>$P_x \supseteq {y}$</td>
<td>$P_x \supseteq {y}$ for some $z \in P_x$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>$P_x \supseteq P_y$</td>
<td>$\text{UnifyPTS}(x, y)$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>$P_x \supseteq P_z, \forall z \in P_y$</td>
<td>$\forall z \in P_y, \text{UnifyPTS}(x, z)$</td>
</tr>
<tr>
<td>$*x = y$</td>
<td>$P_z \supseteq P_y, \forall z \in P_x$</td>
<td>$\forall z \in P_x, \text{UnifyPTS}(y, z)$</td>
</tr>
</tbody>
</table>

### Andersen’s view
- $x$ points to $y$
- Include $y$ in the points-to set of $x$

### Steensgaard’s view
Andersen’s and Steensgaard’s Points-to Analysis

<table>
<thead>
<tr>
<th>Statement</th>
<th>Andersen’s Points-to Sets</th>
<th>Steensgaard’s Points-to Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = &amp;y )</td>
<td>( P_x \supseteq {y} )</td>
<td>( P_x \supseteq {y} ) for some ( z \in P_x )</td>
</tr>
<tr>
<td>( x = y )</td>
<td>( P_x \supseteq P_y )</td>
<td>( \text{UnifyPTS}(x, y) )</td>
</tr>
<tr>
<td>( x = *y )</td>
<td>( P_x \supseteq P_z, \forall z \in P_y )</td>
<td>( \forall z \in P_y, \text{UnifyPTS}(x, z) )</td>
</tr>
<tr>
<td>( *x = y )</td>
<td>( P_z \supseteq P_y, \forall z \in P_x )</td>
<td>( \forall z \in P_x, \text{UnifyPTS}(y, z) )</td>
</tr>
</tbody>
</table>

**Andersen’s view**
- \( x \) points to \( y \)
- Include \( y \) in the points-to set of \( x \)

**Steensgaard's view**
- Equivalence between: All pointees of \( x \)
- Unify \( y \) and pointees of \( x \)
## Andersen’s and Steensgaard’s Points-to Analysis

<table>
<thead>
<tr>
<th>Statement</th>
<th>Andersen’s Points-to Sets</th>
<th>Steensgaard’s Points-to Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = &amp; y$</td>
<td>$P_x \supseteq {y}$</td>
<td>$P_x \supseteq {y}$, $\text{Unify}(y, z)$ for some $z \in P_x$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>$P_x \supseteq P_y$</td>
<td>$\text{Unify}PTS(x, y)$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>$P_x \supseteq P_z$, $\forall z \in P_y$</td>
<td>$\forall z \in P_y$, $\text{Unify}PTS(x, z)$</td>
</tr>
<tr>
<td>$*x = y$</td>
<td>$P_z \supseteq P_y$, $\forall z \in P_x$</td>
<td>$\forall z \in P_x$, $\text{Unify}PTS(y, z)$</td>
</tr>
</tbody>
</table>

**Andersen’s view**

**Steensgaard’s view**
# Andersen’s and Steensgaard’s Points-to Analysis

<table>
<thead>
<tr>
<th>Statement</th>
<th>Andersen’s Points-to Sets</th>
<th>Steensgaard’s Points-to Sets</th>
</tr>
</thead>
</table>
| $x = \& y$ | $P_x \supseteq \{y\}$ | $P_x \supseteq \{y\}$  
|            |                           | $\text{Unify}(y, z)$ for some $z \in P_x$ |
| $x = y$    | $P_x \subseteq P_y$      | $\text{UnifyPTS}(x, y)$    |
| $x = \ast y$ | $P_x \supseteq P_z$, $\forall z \in P_y$ | $\forall z \in P_y$, $\text{UnifyPTS}(x, z)$ |
| $\ast x = y$ | $P_z \supseteq P_y$, $\forall z \in P_x$ | $\forall z \in P_x$, $\text{UnifyPTS}(y, z)$ |

### Andersen’s view
- $x$ points to pointees of $y$
- Include the pointees of $y$ in the points-to set of $x$

### Steensgaard’s view
Andersen’s and Steensgaard’s Points-to Analysis

<table>
<thead>
<tr>
<th>Statement</th>
<th>Andersen’s Points-to Sets</th>
<th>Steensgaard’s Points-to Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x = &amp; y)</td>
<td>(P_x \supseteq {y})</td>
<td>(P_x \supseteq {y}) for some (z \in P_x)</td>
</tr>
<tr>
<td>(x = y)</td>
<td>(P_x \supseteq P_y)</td>
<td>(\text{UnifyPTS}(x, y))</td>
</tr>
<tr>
<td>(x = *y)</td>
<td>(P_x \supseteq P_z, \forall z \in P_y)</td>
<td>(\forall z \in P_y, \text{UnifyPTS}(x, z))</td>
</tr>
<tr>
<td>(*x = y)</td>
<td>(P_z \supseteq P_y, \forall z \in P_x)</td>
<td>(\forall z \in P_x, \text{UnifyPTS}(y, z))</td>
</tr>
</tbody>
</table>

**Andersen’s view**
- \(x\) points to pointees of \(y\)
- Include the pointees of \(y\) in the points-to set of \(x\)

**Steensgaard’s view**
- Equivalence between: Pointees of \(x\) and pointees of \(y\)
- Unify points-to sets of \(x\) and \(y\)
### Andersen’s and Steensgaard’s Points-to Analysis

<table>
<thead>
<tr>
<th>Statement</th>
<th>Andersen’s Points-to Sets</th>
<th>Steensgaard’s Points-to Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = &amp; y$</td>
<td>$P_x \supset {y}$</td>
<td>$P_x \supset {y}$ for some $z \in P_x$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>$P_x \supset P_y$</td>
<td>$\text{UnifyPTS}(x, y)$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>$P_x \supset P_z$, $\forall z \in P_y$</td>
<td>$\forall z \in P_y$, $\text{UnifyPTS}(x, z)$</td>
</tr>
<tr>
<td>$*x = y$</td>
<td>$P_z \supset P_y$, $\forall z \in P_x$</td>
<td>$\forall z \in P_x$, $\text{UnifyPTS}(y, z)$</td>
</tr>
</tbody>
</table>

**Andersen’s view**

**Steensgaard’s view**
### Andersen’s and Steensgaard’s Points-to Analysis

<table>
<thead>
<tr>
<th>Statement</th>
<th>Andersen’s Points-to Sets</th>
<th>Steensgaard’s Points-to Sets</th>
</tr>
</thead>
</table>
| \( x = \& y \) | \( P_x \supseteq \{ y \} \) | \( P_x \supseteq \{ y \} \)  
\( \text{Unify}(y, z) \) for some \( z \in P_x \) |
| \( x = y \) | \( P_x \supseteq P_y \) | \( \text{UnifyPTS}(x, y) \) |
| \( x = *y \) | \( P_x \supseteq P_z, \forall z \in P_y \) | \( \forall z \in P_y, \text{UnifyPTS}(x, z) \) |
| \( *x = y \) | \( P_z \supseteq P_y, \forall z \in P_x \) | \( \forall z \in P_x, \text{UnifyPTS}(y, z) \) |

### Andersen’s view
- \( x \) points to pointees of pointees of \( y \)
- Include the pointees of pointees of \( y \) in the points-to set of \( x \)

### Steensgaard’s view
## Andersen’s and Steensgaard’s Points-to Analysis

<table>
<thead>
<tr>
<th>Statement</th>
<th>Andersen’s Points-to Sets</th>
<th>Steensgaard’s Points-to Sets</th>
</tr>
</thead>
</table>
| $x = \& y$ | $P_x \supseteq \{y\}$ | $P_x \supseteq \{y\}$  
  
  Unify($y, z$) for some $z \in P_x$ |
| $x = y$ | $P_x \supseteq P_y$ | $\text{UnifyPTS}(x, y)$ |
| $x = *y$ | $P_x \supseteq P_z, \forall z \in P_y$ | $\forall z \in P_y, \text{UnifyPTS}(x, z)$ |
| $*x = y$ | $P_z \supseteq P_y, \forall z \in P_x$ | $\forall z \in P_x, \text{UnifyPTS}(y, z)$ |

**Andersen’s view**

- $x$ points to pointees of pointees of $y$
- Include the pointees of pointees of $y$ in the points-to set of $x$

**Steensgaard’s view**

- Equivalence between: Pointees of $x$ and pointees of pointees of $y$
- Unify points-to sets of $x$ and pointees of $y$
## Andersen’s and Steensgaard’s Points-to Analysis

<table>
<thead>
<tr>
<th>Statement</th>
<th>Andersen’s Points-to Sets</th>
<th>Steensgaard’s Points-to Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x = &amp;y</code></td>
<td><code>P_x ⊇ \{y\}</code></td>
<td><code>P_x ⊇ \{y\}</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>Unify(y, z)</code> for some <code>z ∈ P_x</code></td>
</tr>
<tr>
<td><code>x = y</code></td>
<td><code>P_x ⊇ P_y</code></td>
<td><code>UnifyPTS(x, y)</code></td>
</tr>
<tr>
<td><code>x = *y</code></td>
<td><code>P_x ⊇ P_z, ∀z ∈ P_y</code></td>
<td><code>∀z ∈ P_y, UnifyPTS(x, z)</code></td>
</tr>
<tr>
<td><code>*x = y</code></td>
<td><code>P_z ⊇ P_y, ∀z ∈ P_x</code></td>
<td><code>∀z ∈ P_x, UnifyPTS(y, z)</code></td>
</tr>
</tbody>
</table>

**Andersen’s view**

**Steensgaard’s view**
# Andersen’s and Steensgaard’s Points-to Analysis

<table>
<thead>
<tr>
<th>Statement</th>
<th>Andersen’s Points-to Sets</th>
<th>Steensgaard’s Points-to Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = &amp;y )</td>
<td>( P_x \supseteq {y} )</td>
<td>( P_x \supseteq {y} ) for some ( z \in P_x )</td>
</tr>
<tr>
<td>( x = y )</td>
<td>( P_x \supseteq P_y )</td>
<td>( \text{UnifyPTS}(x, y) )</td>
</tr>
<tr>
<td>( x = *y )</td>
<td>( P_x \supseteq P_z, \forall z \in P_y )</td>
<td>( \forall z \in P_y, \text{UnifyPTS}(x, z) )</td>
</tr>
<tr>
<td>( *x = y )</td>
<td>( P_z \supseteq P_y, \forall z \in P_x )</td>
<td>( \forall z \in P_x, \text{UnifyPTS}(y, z) )</td>
</tr>
</tbody>
</table>

**Andersen’s view**

- Pointees of \( x \) points to pointees of \( y \)
- Include the pointees of \( y \) in the points-to set of the pointees of \( x \)

**Steensgaard’s view**
## Andersen’s and Steensgaard’s Points-to Analysis

<table>
<thead>
<tr>
<th>Statement</th>
<th>Andersen’s Points-to Sets</th>
<th>Steensgaard’s Points-to Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = &amp; y$</td>
<td>$P_x \supseteq {y}$</td>
<td>$P_x \supseteq {y}$ for some $z \in P_x$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>$P_x \supseteq P_y$</td>
<td>$\text{UnifyPTS}(x, y)$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>$P_x \supseteq P_z, \ \forall z \in P_y$</td>
<td>$\forall z \in P_y, \ \text{UnifyPTS}(x, z)$</td>
</tr>
<tr>
<td>$*x = y$</td>
<td>$P_z \supseteq P_y, \ \forall z \in P_x$</td>
<td>$\forall z \in P_x, \ \text{UnifyPTS}(y, z)$</td>
</tr>
</tbody>
</table>

### Andersen’s view
- Pointees of $x$ points to pointees of $y$
- Include the pointees of $y$ in the points-to set of the pointees of $x$

### Steensgaard’s view
- Equivalence between: Pointees of pointees of $x$ and pointees of $y$
- Unify points-to sets of pointees of $x$ and $y$
# Andersen’s and Steensgaard’s Points-to Analysis

<table>
<thead>
<tr>
<th>Statement</th>
<th>Andersen’s Points-to Sets</th>
<th>Steensgaard’s Points-to Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x = &amp;y)</td>
<td>(P_x \supseteq {y})</td>
<td>(P_x \supseteq {y}) (\text{Unify}(y, z)) for some (z \in P_x)</td>
</tr>
<tr>
<td>(x = y)</td>
<td>(P_x \supseteq P_y)</td>
<td>(\text{UnifyPTS}(x, y))</td>
</tr>
<tr>
<td>(x = *y)</td>
<td>(P_x \supseteq P_z, \forall z \in P_y)</td>
<td>(\forall z \in P_y, \text{UnifyPTS}(x, z))</td>
</tr>
<tr>
<td>(*x = y)</td>
<td>(P_z \supseteq P_y, \forall z \in P_x)</td>
<td>(\forall z \in P_x, \text{UnifyPTS}(y, z))</td>
</tr>
</tbody>
</table>
### Andersen’s and Steensgaard’s Points-to Analysis

<table>
<thead>
<tr>
<th>Statement</th>
<th>Andersen’s Points-to Sets</th>
<th>Steensgaard’s Points-to Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = &amp; y$</td>
<td>$P_x \supseteq {y}$</td>
<td>$P_x \supseteq {y}$ \text{ for some } z \in P_x</td>
</tr>
<tr>
<td>$x = y$</td>
<td>$P_x \supseteq P_y$</td>
<td>$\text{UnifyPTS}(x, y)$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>$P_x \supseteq P_z, \ \forall z \in P_y$</td>
<td>$\forall z \in P_y, \text{UnifyPTS}(x, z)$</td>
</tr>
<tr>
<td>$\ast x = y$</td>
<td>$P_z \supseteq P_y, \ \forall z \in P_x$</td>
<td>$\forall z \in P_x, \text{UnifyPTS}(y, z)$</td>
</tr>
</tbody>
</table>

**Inclusion**
### Andersen’s and Steensgaard’s Points-to Analysis

<table>
<thead>
<tr>
<th>Statement</th>
<th>Andersen’s Points-to Sets</th>
<th>Steensgaard’s Points-to Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = &amp; y$</td>
<td>$P_x \supseteq {y}$</td>
<td>$P_x \supseteq {y}$ $\text{Unify}(y, z)$ for some $z \in P_x$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>$P_x \supseteq P_y$</td>
<td>$\text{UnifyPTS}(x, y)$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>$P_x \supseteq P_z, \forall z \in P_y$</td>
<td>$\forall z \in P_y, \text{UnifyPTS}(x, z)$</td>
</tr>
<tr>
<td>$*x = y$</td>
<td>$P_z \supseteq P_y, \forall z \in P_x$</td>
<td>$\forall z \in P_x, \text{UnifyPTS}(y, z)$</td>
</tr>
</tbody>
</table>

**Inclusion**

**Equality**
Inclusion Based (aka Andersen’s) Points-to Analysis: Example 1

Program

1. \(a = \&b\)
2. \(c = a\)
3. \(a = \&d\)
4. \(a = \&e\)
5. \(b = a\)
Inclusion Based (aka Andersen’s) Points-to Analysis: Example 1

Program

1. \( a = \&b \)
2. \( c = a \)
3. \( a = \&d \)
4. \( a = \&e \)
5. \( b = a \)

<table>
<thead>
<tr>
<th>Node</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( P_a \supseteq {b} )</td>
</tr>
<tr>
<td>2</td>
<td>( P_c \supseteq P_a )</td>
</tr>
<tr>
<td>3</td>
<td>( P_a \supseteq {d} )</td>
</tr>
<tr>
<td>4</td>
<td>( P_a \supseteq {e} )</td>
</tr>
<tr>
<td>5</td>
<td>( P_b \supseteq P_a )</td>
</tr>
</tbody>
</table>
Inclusion Based (aka Andersen’s) Points-to Analysis: Example 1

Program

1. a = &b
2. c = a
3. a = &d
4. a = &e
5. b = a

Points-to Graph

Node | Constraint
--- | ---
1 | \( P_a \supseteq \{b\} \)
2 | \( P_c \supseteq P_a \)
3 | \( P_a \supseteq \{d\} \)
4 | \( P_a \supseteq \{e\} \)
5 | \( P_b \supseteq P_a \)
**Inclusion Based (aka Andersen’s) Points-to Analysis: Example 1**

**Program**

1. `a = &b`
2. `c = a`
3. `a = &d`
4. `a = &e`
5. `b = a`

**Points-to Graph**

```
<table>
<thead>
<tr>
<th>Node</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( P_a \supseteq {b} )</td>
</tr>
<tr>
<td>2</td>
<td>( P_c \supseteq P_a )</td>
</tr>
<tr>
<td>3</td>
<td>( P_a \supseteq {d} )</td>
</tr>
<tr>
<td>4</td>
<td>( P_a \supseteq {e} )</td>
</tr>
<tr>
<td>5</td>
<td>( P_b \supseteq P_a )</td>
</tr>
</tbody>
</table>
```
Inclusion Based (aka Andersen’s) Points-to Analysis: Example 1

Program

1. a = &b
2. c = a
3. a = &d
4. a = &e
5. b = a

Points-to Graph

Node | Constraint
---|---
1 | $P_a \supseteq \{b\}$
2 | $P_c \supseteq P_a$
3 | $P_a \supseteq \{d\}$
4 | $P_a \supseteq \{e\}$
5 | $P_b \supseteq P_a$
Inclusion Based (aka Andersen’s) Points-to Analysis: Example 1

Program

<table>
<thead>
<tr>
<th>Node</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$P_a \supseteq {b}$</td>
</tr>
<tr>
<td>2</td>
<td>$P_c \supseteq P_a$</td>
</tr>
<tr>
<td>3</td>
<td>$P_a \supseteq {d}$</td>
</tr>
<tr>
<td>4</td>
<td>$P_a \supseteq {e}$</td>
</tr>
<tr>
<td>5</td>
<td>$P_b \supseteq P_a$</td>
</tr>
</tbody>
</table>

Points-to Graph

- Since $P_a$ has changed, $P_c$ needs to be processed again
Inclusion Based (aka Andersen’s) Points-to Analysis: Example 1

Program

1. a = &b
2. c = a
3. a = &d
4. a = &e
5. b = a

Points-to Graph

Node | Constraint
--- | ---
1 | $P_a \supseteq \{b\}$
2 | $P_c \supseteq P_a$
3 | $P_a \supseteq \{d\}$
4 | $P_a \supseteq \{e\}$
5 | $P_b \supseteq P_a$
Inclusion Based (aka Andersen’s) Points-to Analysis: Example 1

Program

1. \( a = \&b \)
2. \( c = a \)
3. \( a = \&d \)
4. \( a = \&e \)
5. \( b = a \)

Points-to Graph

<table>
<thead>
<tr>
<th>Node</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( P_a \supseteq {b} )</td>
</tr>
<tr>
<td>2</td>
<td>( P_c \supseteq P_a )</td>
</tr>
<tr>
<td>3</td>
<td>( P_a \supseteq {d} )</td>
</tr>
<tr>
<td>4</td>
<td>( P_a \supseteq {e} )</td>
</tr>
<tr>
<td>5</td>
<td>( P_b \supseteq P_a )</td>
</tr>
</tbody>
</table>

- Observe that \( P_c \) is processed for the third time
- Order of processing the sets influences efficiency significantly
- A plethora of heuristics have been proposed
Inclusion Based (aka Andersen’s) Points-to Analysis: Example 1

Program

1. \( a = \& b \)
2. \( c = a \)
3. \( a = \& d \)
4. \( a = \& e \)
5. \( b = a \)

Points-to Graph

Node | Constraint
--- | ---
1 | \( P_a \supseteq \{b\} \)
2 | \( P_c \supseteq P_a \)
3 | \( P_a \supseteq \{d\} \)
4 | \( P_a \supseteq \{e\} \)
5 | \( P_b \supseteq P_a \)
Inclusion Based (aka Andersen’s) Points-to Analysis: Example 1

Program

<table>
<thead>
<tr>
<th>Node</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$P_a \supseteq {b}$</td>
</tr>
<tr>
<td>2</td>
<td>$P_c \supseteq P_a$</td>
</tr>
<tr>
<td>3</td>
<td>$P_a \supseteq {d}$</td>
</tr>
<tr>
<td>4</td>
<td>$P_a \supseteq {e}$</td>
</tr>
<tr>
<td>5</td>
<td>$P_b \supseteq P_a$</td>
</tr>
</tbody>
</table>

Actually:
- c does not point to any location in block 1
- a does not point b in block 5
  (the method ignores the kill due to 3 and 4)
- b does not point to itself at any time
Equality Based (aka Steensgaard’s) Points-to Analysis: Example 1

Program

1. a = &b
2. c = a
3. a = &d
4. a = &e
5. b = a
Equality Based (aka Steensgaard’s) Points-to Analysis: Example 1

Program

1. \( a = \&b \)
2. \( c = a \)
3. \( a = \&d \)
4. \( a = \&e \)
5. \( b = a \)

Node | Constraint
--- | ---
1 | \( P_a \supseteq \{b\} \)
   | \( \text{Unify}(x, d), x \in P_a \)
2 | \( \text{UnifyPTS}(c, a) \)
3 | \( P_a \supseteq \{d\} \)
   | \( \text{Unify}(x, d), x \in P_a \)
4 | \( P_a \supseteq \{e\} \)
   | \( \text{Unify}(x, e), x \in P_a \)
5 | \( \text{UnifyPTS}(b, a) \)
Equality Based (aka Steensgaard’s) Points-to Analysis: Example 1

Program

1. \( a = \&b \)

2. \( c = a \)

3. \( a = \&d \)

4. \( a = \&e \)

5. \( b = a \)

Points-to Graph

Node | Constraint
--- | ---
1 | \( P_a \supseteq \{b\} \)
   | \( \text{Unify}(x, d), x \in P_a \)
2 | \( \text{UnifyPTS}(c, a) \)
3 | \( P_a \supseteq \{d\} \)
   | \( \text{Unify}(x, d), x \in P_a \)
4 | \( P_a \supseteq \{e\} \)
   | \( \text{Unify}(x, e), x \in P_a \)
5 | \( \text{UnifyPTS}(b, a) \)
Equality Based (aka Steensgaard’s) Points-to Analysis: Example 1

Program

1. \( a = &b \)
2. \( c = a \)
3. \( a = &d \)
4. \( a = &e \)
5. \( b = a \)

Points-to Graph

Node | Constraint
--- | ---
1 | \( P_a \supseteq \{ b \} \)
   | \( \text{Unify}(x, d), x \in P_a \)
2 | \( \text{UnifyPTS}(c, a) \)
3 | \( P_a \supseteq \{ d \} \)
   | \( \text{Unify}(x, d), x \in P_a \)
4 | \( P_a \supseteq \{ e \} \)
   | \( \text{Unify}(x, e), x \in P_a \)
5 | \( \text{UnifyPTS}(b, a) \)
Equality Based (aka Steensgaard’s) Points-to Analysis: Example 1

Program

<table>
<thead>
<tr>
<th>Node</th>
<th>Constraint</th>
</tr>
</thead>
</table>
| 1    | $P_a \supseteq \{b\}$  
|      | $\text{Unify}(x, d), x \in P_a$ |
| 2    | $\text{UnifyPTS}(c, a)$ |
| 3    | $P_a \supseteq \{d\}$  
|      | $\text{Unify}(x, d), x \in P_a$ |
| 4    | $P_a \supseteq \{e\}$  
|      | $\text{Unify}(x, e), x \in P_a$ |
| 5    | $\text{UnifyPTS}(b, a)$ |

Points-to Graph
Equality Based (aka Steensgaard’s) Points-to Analysis: Example 1

Program

<table>
<thead>
<tr>
<th>Node</th>
<th>Constraint</th>
</tr>
</thead>
</table>
| 1    | $P_a \supseteq \{b\}$  
Unify($x$, $d$), $x \in P_a$ |
| 2    | UnifyPTS($c$, $a$) |
| 3    | $P_a \supseteq \{d\}$  
Unify($x$, $d$), $x \in P_a$ |
| 4    | $P_a \supseteq \{e\}$  
Unify($x$, $e$), $x \in P_a$ |
| 5    | UnifyPTS($b$, $a$) |

Points-to Graph
Equality Based (aka Steensgaard’s) Points-to Analysis: Example 1

Program

1. \( a = \&b \)
2. \( c = a \)
3. \( a = \&d \)
4. \( a = \&e \)
5. \( b = a \)

Points-to Graph

Node | Constraint
--- | ---
1 | \( P_a \supseteq \{b\} \)<br>Unify\((x, d), x \in P_a\)
2 | Unify\(PTS\)(\(c, a\))
3 | \( P_a \supseteq \{d\} \)<br>Unify\((x, d), x \in P_a\)
4 | \( P_a \supseteq \{e\} \)<br>Unify\((x, e), x \in P_a\)
5 | Unify\(PTS\)(\(b, a\))
Equality Based (aka Steensgaard’s) Points-to Analysis: Example 1

Program

1. \( a = \& b \)

2. \( c = a \)

3. \( a = \& d \)

4. \( a = \& e \)

5. \( b = a \)

Points-to Graph

<table>
<thead>
<tr>
<th>Node</th>
<th>Constraint</th>
</tr>
</thead>
</table>
| 1    | \( P_a \supseteq \{ b \} \)  
Unify\((x, d), x \in P_a\) |
| 2    | UnifyPTS\((c, a)\) |
| 3    | \( P_a \supseteq \{ d \} \)  
Unify\((x, d), x \in P_a\) |
| 4    | \( P_a \supseteq \{ e \} \)  
Unify\((x, e), x \in P_a\) |
| 5    | UnifyPTS\((b, a)\) |
Equality Based (aka Steensgaard’s) Points-to Analysis: Example 1

Program

<table>
<thead>
<tr>
<th>Node</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( P_a \supseteq {b} ) ( Unify(x, d), x \in P_a )</td>
</tr>
<tr>
<td>2</td>
<td>( UnifyPTS(c, a) )</td>
</tr>
<tr>
<td>3</td>
<td>( P_a \supseteq {d} ) ( Unify(x, d), x \in P_a )</td>
</tr>
<tr>
<td>4</td>
<td>( P_a \supseteq {e} ) ( Unify(x, e), x \in P_a )</td>
</tr>
<tr>
<td>5</td>
<td>( UnifyPTS(b, a) )</td>
</tr>
</tbody>
</table>
Equality Based (aka Steensgaard’s) Points-to Analysis: Example 1

Program

1. `a = &b`
2. `c = a`
3. `a = &d`
4. `a = &e`
5. `b = a`

Points-to Graph

<table>
<thead>
<tr>
<th>Node</th>
<th>Constraint</th>
</tr>
</thead>
</table>
| 1    | $P_a \supseteq \{b\}$  
      | $\text{Unify}(x, d), x \in P_a$ |
| 2    | $\text{UnifyPTS}(c, a)$ |
| 3    | $P_a \supseteq \{d\}$  
      | $\text{Unify}(x, d), x \in P_a$ |
| 4    | $P_a \supseteq \{e\}$  
      | $\text{Unify}(x, e), x \in P_a$ |
| 5    | $\text{UnifyPTS}(b, a)$ |
Equality Based (aka Steensgaard’s) Points-to Analysis: Example 1

Program

1. \( a = &b \)

2. \( c = a \)

3. \( a = &d \)

4. \( a = &e \)

5. \( b = a \)

Points-to Graph

<table>
<thead>
<tr>
<th>Node</th>
<th>Constraint</th>
</tr>
</thead>
</table>
| 1    | \( P_a \supseteq \{b\} \)  
\( \text{Unify}(x, d), x \in P_a \) |
| 2    | \( \text{UnifyPTS}(c, a) \) |
| 3    | \( P_a \supseteq \{d\} \)  
\( \text{Unify}(x, d), x \in P_a \) |
| 4    | \( P_a \supseteq \{e\} \)  
\( \text{Unify}(x, e), x \in P_a \) |
| 5    | \( \text{UnifyPTS}(b, a) \) |

- The full blown up points-to graph has far more edges than in the graph created by Andersen’s method
- Far more efficient but far less precise
Comparing Equality and Inclusion Based Analyses (2)

- Andersen’s algorithm is cubic in number of pointers
- Steensgaard’s algorithm is nearly linear in number of pointers
Comparing Equality and Inclusion Based Analyses (2)

- Andersen’s algorithm is cubic in number of pointers
- Steensgaard’s algorithm is nearly linear in number of pointers
  - How can it be more efficient by an orders of magnitude?
## Efficiency of Equality Based Approach

<table>
<thead>
<tr>
<th>Program</th>
<th>Andersen’s approach</th>
<th>Steensgaard’s approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a = &amp; b )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( a = &amp; c )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( b = &amp; d )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( b = &amp; c )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Andersen’s inclusion based wisdom:
  - Add edges and let the number of successors increase
- Steensgaard’s equality based wisdom:
  - Merge multiple successors and maintain a single successor of any node
## Efficiency of Equality Based Approach

<table>
<thead>
<tr>
<th>Program</th>
<th>Andersen’s approach</th>
<th>Steensgaard’s approach</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a = &amp;b</code> <code>a = &amp;c</code> <code>b = &amp;d</code> <code>b = &amp;c</code></td>
<td>![Andersen's graph]</td>
<td>![Steensgaard's graph]</td>
</tr>
</tbody>
</table>

- Andersen’s inclusion based wisdom:
  - Add edges and let the number of successors increase
- Steensgaard’s equality based wisdom:
  - Merge multiple successors and maintain a single successor of any node
## Efficiency of Equality Based Approach

<table>
<thead>
<tr>
<th>Program</th>
<th>Andersen’s approach</th>
<th>Steensgaard’s approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = &amp;b</td>
<td><img src="image1.png" alt="Diagram" /></td>
<td><img src="image2.png" alt="Diagram" /></td>
</tr>
<tr>
<td>a = &amp;c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = &amp;d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = &amp;c</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Andersen’s inclusion based wisdom:
  - Add edges and let the number of successors increase

- Steensgaard’s equality based wisdom:
  - Merge multiple successors and maintain a single successor of any node
### Efficiency of Equality Based Approach

<table>
<thead>
<tr>
<th>Program</th>
<th>Andersen’s approach</th>
<th>Steensgaard’s approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a = &amp; b)</td>
<td><img src="image1.png" alt="Diagram" /></td>
<td><img src="image2.png" alt="Diagram" /></td>
</tr>
<tr>
<td>(a = &amp; c)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(b = &amp; d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(b = &amp; c)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Andersen’s inclusion based wisdom:**
  - Add edges and let the number of successors increase

- **Steensgaard’s equality based wisdom:**
  - Merge multiple successors and maintain a single successor of any node
Efficiency of Equality Based Approach

<table>
<thead>
<tr>
<th>Program</th>
<th>Andersen’s approach</th>
<th>Steensgaard’s approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = &amp;b</td>
<td><img src="image1" alt="Andersen's graph" /></td>
<td><img src="image2" alt="Steensgaard's graph" /></td>
</tr>
<tr>
<td>a = &amp;c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = &amp;d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = &amp;c</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Andersen’s inclusion based wisdom:
  - Add edges and let the number of successors increase

- Steensgaard’s equality based wisdom:
  - Merge multiple successors and maintain a single successor of any node
Efficiency of Equality Based Approach

<table>
<thead>
<tr>
<th>Program</th>
<th>Andersen's approach</th>
<th>Steensgaard’s approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = &amp;b</td>
<td><img src="image1" alt="Andersen's diagram" /></td>
<td><img src="image2" alt="Steensgaard's diagram" /></td>
</tr>
<tr>
<td>a = &amp;c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = &amp;d</td>
<td><img src="image3" alt="Andersen's diagram" /></td>
<td><img src="image4" alt="Steensgaard's diagram" /></td>
</tr>
<tr>
<td>b = &amp;c</td>
<td><img src="image5" alt="Andersen's diagram" /></td>
<td><img src="image6" alt="Steensgaard's diagram" /></td>
</tr>
</tbody>
</table>

- Andersen’s inclusion based wisdom:
  - Add edges and let the number of successors increase

- Steensgaard’s equality based wisdom:
  - Merge multiple successors and maintain a single successor of any node
Efficiency of Equality Based Approach

<table>
<thead>
<tr>
<th>Program</th>
<th>Andersen’s approach</th>
<th>Steensgaard’s approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = &amp;b</td>
<td><img src="image1" alt="Andersen's diagram" /></td>
<td><img src="image2" alt="Steensgaard's diagram" /></td>
</tr>
<tr>
<td>a = &amp;c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = &amp;d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = &amp;c</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Andersen’s inclusion based wisdom:
  - Add edges and let the number of successors increase

- Steensgaard’s equality based wisdom:
  - Merge multiple successors and maintain a single successor of any node
Efficiency of Equality Based Approach

<table>
<thead>
<tr>
<th>Program</th>
<th>Andersen's approach</th>
<th>Steensgaard’s approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = &amp;b</td>
<td><img src="image" alt="Andersen's graph" /></td>
<td><img src="image" alt="Steensgaard's graph" /></td>
</tr>
<tr>
<td>a = &amp;c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = &amp;d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = &amp;c</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Andersen’s inclusion based wisdom:
  - Add edges and let the number of successors increase
- Steensgaard’s equality based wisdom:
  - Merge multiple successors and maintain a single successor of any node
  - Since a larger number of pointers treated are alike and fewer distinctions are maintained, we get much smaller points-to graphs
Efficiency of Equality Based Approach

<table>
<thead>
<tr>
<th>Program</th>
<th>Andersen’s approach</th>
<th>Steensgaard’s approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = &amp;b</td>
<td><img src="image1" alt="Andersen's graph" /></td>
<td><img src="image2" alt="Steensgaard's graph" /></td>
</tr>
<tr>
<td>a = &amp;c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = &amp;d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = &amp;c</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Andersen’s inclusion based wisdom:
  - Add edges and let the number of successors increase

- Steensgaard’s equality based wisdom:
  - Merge multiple successors and maintain a single successor of any node
  - Since a larger number of pointers treated are alike and fewer distinctions are maintained, we get much smaller points-to graphs
  - Efficient *Union-Find* algorithms to merge intersecting subsets
Inclusion Based (aka Andersen’s) Points-to Analysis: Example 2

\[ \begin{align*}
  n_1 & : x = & y \\
  & y = & z \\
  & z = & u
\end{align*} \]

\[ \begin{align*}
  n_2 & : *z = y \\
  n_3 & : z = y
\end{align*} \]

\[ \begin{align*}
  n_4 & : y = & x \\
  & use u \\
  & use x
\end{align*} \]
Inclusion Based (aka Andersen’s) Points-to Analysis: Example 2

**Points-to Graph**

- $x \text{ “points-to” } y$
- $y \text{ “points-to” } z$
- $z \text{ “points-to” } u$

**Constraints on Points-to Sets**

- $P_x \supseteq \{y\}$
- $P_y \supseteq \{z\}$
- $P_z \supseteq \{u\}$

---

$n_1$

- $x = &y$
- $y = &z$
- $z = &u$

$n_2$

- $*z = y$

$n_3$

- $z = y$

$n_4$

- $y = &x$
- use $u$
- use $x$

Points-to Graph: $x \rightarrow y \rightarrow z \rightarrow u$
Inclusion Based (aka Andersen’s) Points-to Analysis: Example 2

- Pointees of z should point to pointees of y also
- u should point to z

Constraints on Points-to Sets

\[ P_x \supseteq \{y\} \]
\[ P_y \supseteq \{z\} \]
\[ P_z \supseteq \{u\} \]

\[ \forall w \in P_z, \ P_w \supseteq P_y \]

Points-to Graph

\[ x \rightarrow y \rightarrow z \rightarrow u \]
Inclusion Based (aka Andersen’s) Points-to Analysis: Example 2

Constraints on Points-to Sets

\[ P_x \supseteq \{y\} \]
\[ P_y \supseteq \{z\} \]
\[ P_z \supseteq \{u\} \]
\[ \forall w \in P_z, \ P_w \supseteq P_y \]

Points-to Graph
Inclusion Based (aka Andersen’s) Points-to Analysis: Example 2

- $z$ should point to pointees of $y$
- $z$ should point to $z$

Constraints on Points-to Sets:

- $P_x \supseteq \{y\}$
- $P_y \supseteq \{z\}$
- $P_z \supseteq \{u\}$

$\forall w \in P_z, \quad P_w \supseteq P_y$

$P_z \supseteq P_y$

Points-to Graph

$\begin{align*}
x &= \& y \\
y &= \& z \\
z &= \& u \\
\end{align*}$

$\begin{align*}
x &= \& y \\
\text{use } u \\
\text{use } x \\
\end{align*}$

$\begin{align*}
x = \& y & \text{ n1} \\
y = \& z & \text{ n2} \\
z = \& u & \text{ n3} \\
\end{align*}$

$\begin{align*}
x &= \& y & \text{ n2} \\
z = \& y & \text{ n3} \\
y &= \& x & \text{ n4} \\
\end{align*}$
Inclusion Based (aka Andersen’s) Points-to Analysis: Example 2

\begin{align*}
  x &= \& y \\
  y &= \& z \\
  z &= \& u \\

  *z &= y \\

  y &= \& x \\

  \text{use } u \\

  \text{use } x
\end{align*}

 Constraints on Points-to Sets

\[
\begin{align*}
  P_x &\supseteq \{y\} \\
  P_y &\supseteq \{z\} \\
  P_z &\supseteq \{u\} \\

  \forall w \in P_z, \quad P_w &\supseteq P_y \\
  P_z &\supseteq P_y
\end{align*}
\]
Inclusion Based (aka Andersen’s) Points-to Analysis: Example 2

Constraints on Points-to Sets

\[
\begin{align*}
P_x & \supseteq \{ y \} \\
P_y & \supseteq \{ z \} \\
P_z & \supseteq \{ u \} \\
\forall w \in P_z, & \quad P_w \supseteq P_y \\
& \quad P_z \supseteq P_y \\
P_y & \supseteq \{ x \}
\end{align*}
\]

Points-to Graph
Inclusion Based (aka Andersen’s) Points-to Analysis: Example 2

Constraints on Points-to Sets

\[
\begin{align*}
P_x & \supseteq \{y\} \\
P_y & \supseteq \{z\} \\
P_z & \supseteq \{u\} \\
\forall w \in P_z, \ P_w & \supseteq P_y \\
P_y & \supseteq \{x\}
\end{align*}
\]

Points-to Graph
Inclusion Based (aka Andersen’s) Points-to Analysis: Example 2

- $z$ and its pointees should point to new pointee of $y$ also
- $u$ and $z$ should point to $x$

Constraints on Points-to Sets

\[
\begin{align*}
  & P_x \supseteq \{y\} \\
  & P_y \supseteq \{z\} \\
  & P_z \supseteq \{u\} \\
  & \forall w \in P_z, \quad P_w \supseteq P_y \\
  & P_z \supseteq P_y \\
  & P_y \supseteq \{x\}
\end{align*}
\]
Inclusion Based (aka Andersen’s) Points-to Analysis: Example 2

Constraints on Points-to Sets

\begin{align*}
P_x & \supseteq \{y\} \\
P_y & \supseteq \{z\} \\
P_z & \supseteq \{u\} \\
\forall w \in P_z, \quad P_w & \supseteq P_y \\
P_z & \supseteq P_y \\
P_y & \supseteq \{x\}
\end{align*}
Inclusion Based (aka Andersen’s) Points-to Analysis: Example 2

Inclusion Based (aka Andersen’s) Points-to Analysis:

Example 2

\[ x = \& y \]
\[ y = \& z \]
\[ z = \& u \]

- Pointees of \( z \) should point to pointees of \( y \)
- \( x \) should point to itself and \( z \)

Constraints on Points-to Sets

\[ P_x \supseteq \{y\} \]
\[ P_y \supseteq \{z\} \]
\[ P_z \supseteq \{u\} \]
\[ \forall w \in P_z, \ P_w \supseteq P_y \]
\[ P_z \supseteq P_y \]
\[ P_y \supseteq \{x\} \]

Points-to Graph

\[ n_1 \]
\[ n_2 \]
\[ n_3 \]
\[ n_4 \]
Inclusion Based (aka Andersen’s) Points-to Analysis: Example 2

\[ x = \& y \]
\[ y = \& z \]
\[ z = \& u \]

\[ *z = y \]

\[ y = \& x \]
\[ \text{use } u \]
\[ \text{use } x \]

Constraints on Points-to Sets

\[ P_x \supseteq \{ y \} \]
\[ P_y \supseteq \{ z \} \]
\[ P_z \supseteq \{ u \} \]

\[ \forall w \in P_z, \quad P_w \supseteq P_y \]
\[ P_z \supseteq P_y \]
\[ P_y \supseteq \{ x \} \]
Equality Based (aka Steensgaard’s) Points-to Analysis: Example 2

- Treat all pointees of a pointer as “equivalent” locations
- Transitive closure
  Pointees of all equivalent locations become equivalent
Equality Based (aka Steensgaard’s) Points-to Analysis: Example 2

- Treat all pointees of a pointer as “equivalent” locations
- Transitive closure
  Pointees of all equivalent locations become equivalent

Effective additional constraints

\[
\text{Unify}(x, y) \\
/* \text{pointees of } x */
\]

\[
\text{Unify}(x, z) \\
/* \text{pointees of } y */
\]

\[
\text{Unify}(x, u) \\
/* \text{pointees of } z */
\]

Andersen’s Points-to Graph
Equality Based (aka Steensgaard’s) Points-to Analysis: Example 2

- Treat all pointees of a pointer as “equivalent” locations
- Transitive closure
  Pointees of all equivalent locations become equivalent

Effective additional constraints

- \( \text{Unify}(x, y) \)
  /* pointees of \( x \) */
- \( \text{Unify}(x, z) \)
  /* pointees of \( y \) */
- \( \text{Unify}(x, u) \)
  /* pointees of \( z \) */

\( \Rightarrow \) \( x, y, z, u \) are equivalent
Equality Based (aka Steensgaard’s) Points-to Analysis: Example 2

- Treat all pointees of a pointer as “equivalent” locations
- Transitive closure
  Pointees of all equivalent locations become equivalent

Effective additional constraints

\[ \text{Unify}(x, y) \text{ /* pointees of } x */ \]
\[ \text{Unify}(x, z) \text{ /* pointees of } y */ \]
\[ \text{Unify}(x, u) \text{ /* pointees of } z */ \]

\[ \Rightarrow x, y, z, u \text{ are equivalent} \]
\[ \Rightarrow \text{Complete graph} \]
## Tutorial Problem for Flow Insensitive Pointer Analysis (1)

<table>
<thead>
<tr>
<th>Program</th>
<th>Inclusion based</th>
<th>Equality based</th>
</tr>
</thead>
<tbody>
<tr>
<td>p = &amp;q</td>
<td></td>
<td></td>
</tr>
<tr>
<td>r = &amp;s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t = &amp;p</td>
<td></td>
<td></td>
</tr>
<tr>
<td>u = p</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*t = r</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tutorial Problem for Flow Insensitive Pointer Analysis (1)

Program
- \( p = \& q \)
- \( r = \& s \)
- \( t = \& p \)
- \( u = p \)
- \( *t = r \)

Inclusion based

Equality based
Tutorial Problem for Flow Insensitive Pointer Analysis (1)

Program

\[ p = \&q \]
\[ r = \&s \]
\[ t = \&p \]
\[ u = p \]
\[ *t = r \]

Inclusion based

Equality based
Tutorial Problem for Flow Insensitive Pointer Analysis (1)

Program

\[ p = \& q \]
\[ r = \& s \]
\[ t = \& p \]
\[ u = p \]
\[ *t = r \]

<table>
<thead>
<tr>
<th>Program</th>
<th>Inclusion based</th>
<th>Equality based</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>u</td>
<td>q</td>
</tr>
<tr>
<td></td>
<td>t</td>
<td>p</td>
</tr>
<tr>
<td></td>
<td></td>
<td>s</td>
</tr>
<tr>
<td></td>
<td>r</td>
<td></td>
</tr>
</tbody>
</table>

Sep 2017 IIT Bombay
Tutorial Problem for Flow Insensitive Pointer Analysis (1)

Program

\[ p = \& q \]
\[ r = \& s \]
\[ t = \& p \]
\[ u = p \]
\[ t = r \]

Inclusion based

Equality based
## Tutorial Problem for Flow Insensitive Pointer Analysis (1)

<table>
<thead>
<tr>
<th>Program</th>
<th>Inclusion based</th>
<th>Equality based</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p = &amp;q</code></td>
<td><img src="image" alt="Inclusion based graph" /></td>
<td><img src="image" alt="Equality based graph" /></td>
</tr>
<tr>
<td><code>r = &amp;s</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>t = &amp;p</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>u = p</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>*t = r</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sep 2017 IIT Bombay
Tutorial Problem for Flow Insensitive Pointer Analysis (1)

Program

- \( p = \&q \)
- \( r = \&s \)
- \( t = \&p \)
- \( u = p \)
- \( *t = r \)

Inclusion based

Equality based

\( u \rightarrow q \)

\( t \rightarrow p \)

\( r \rightarrow s \)
Tutorial Problem for Flow Insensitive Pointer Analysis (1)

Program

- \( p = \& q \)
- \( r = \& s \)
- \( t = \& p \)
- \( u = p \)
- \( *t = r \)

Inclusion based

Equality based
### Tutorial Problem for Flow Insensitive Pointer Analysis (1)

#### Program

- `p = &q`
- `r = &s`
- `t = &p`
- `u = p`
- `∗t = r`

#### Inclusion based

![Inclusion based diagram]

#### Equality based

![Equality based diagram]
Tutorial Problem for Flow Insensitive Pointer Analysis (1)

Program:

\[
\begin{align*}
  p &= \&q \\
  r &= \&s \\
  t &= \&p \\
  u &= p \\
  *t &= r
\end{align*}
\]

Inclusion based:

Equality based:
Tutorial Problem for Flow Insensitive Pointer Analysis (1)

Program

\[ p = \&q \]
\[ r = \&s \]
\[ t = \&p \]
\[ u = p \]
\[ \ast t = r \]

Inclusion based

Equality based
Tutorial Problem for Flow Insensitive Pointer Analysis (1)

Program

\[
\begin{align*}
  p &= \&q \\
  r &= \&s \\
  t &= \&p \\
  u &= p \\
  *t &= r
\end{align*}
\]

Inclusion based

Equality based
Tutorial Problem for Flow Insensitive Pointer Analysis (1)

Program:

\[ p = \&q \]
\[ r = \&s \]
\[ t = \&p \]
\[ u = p \]
\[ *t = r \]

Inclusion based:

Equality based:
## Tutorial Problem for Flow Insensitive Pointer Analysis (1)

<table>
<thead>
<tr>
<th>Program</th>
<th>Inclusion based</th>
<th>Equality based</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p = &amp; q )</td>
<td><img src="image1" alt="Inclusion Diagram" /></td>
<td><img src="image2" alt="Equality Diagram" /></td>
</tr>
<tr>
<td>( r = &amp; s )</td>
<td><img src="image3" alt="Inclusion Diagram" /></td>
<td></td>
</tr>
<tr>
<td>( t = &amp; p )</td>
<td><img src="image4" alt="Inclusion Diagram" /></td>
<td><img src="image5" alt="Equality Diagram" /></td>
</tr>
<tr>
<td>( u = p )</td>
<td><img src="image6" alt="Inclusion Diagram" /></td>
<td></td>
</tr>
<tr>
<td>( *t = r )</td>
<td><img src="image7" alt="Inclusion Diagram" /></td>
<td></td>
</tr>
</tbody>
</table>
## Tutorial Problem for Flow Insensitive Pointer Analysis (1)

<table>
<thead>
<tr>
<th>Program</th>
<th>Inclusion based</th>
<th>Equality based</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p = &amp;q</code></td>
<td><img src="image" alt="Inclusion based diagram" /></td>
<td><img src="image" alt="Equality based diagram" /></td>
</tr>
<tr>
<td><code>r = &amp;s</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>t = &amp;p</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>u = p</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>*t = r</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sep 2017
Tutorial Problem for Flow Insensitive Pointer Analysis (1)

Program

- \( p = \&q \)
- \( r = \&s \)
- \( t = \&p \)
- \( u = p \)
- \( *t = r \)

Inclusion based

Equality based
Tutorial Problem for Flow Insensitive Pointer Analysis (1)

Program

- \( p = \& q \)
- \( r = \& s \)
- \( t = \& p \)
- \( u = p \)
- \( *t = r \)

Inclusion based

Equality based
Tutorial Problems for Flow Insensitive Pointer Analysis (2)

Compute flow insensitive points-to information using inclusion based method as well as equality based method

```c
if (...)  
    p = &x;  
else  
    p = &y;  

x = &a;  
y = &b;  
*p = &c;  
*y = &a;
```
Tutorial Problem for Flow Insensitive Pointer Analysis (3)

Compute flow insensitive points-to information using inclusion based method as well as equality based method.

```
\begin{align*}
\text{n}_1 & \quad b = & \& a; \\
\text{n}_2 & \quad c = b; \\
\text{n}_3 & \quad a = & \& b; \\
\text{n}_4 & \quad a = & \& c; \\
\text{n}_5 & \quad a = *a; \\
\text{n}_6 & \quad *b = c;
\end{align*}
```
An Outline of Pointer Analysis Coverage

- The larger perspective
- Comparing Points-to and Alias information
- Flow Insensitive Points-to Analysis
- **Flow Sensitive Points-to Analysis** Next Topic
- Pointer Analyses: An Engineer’s Landscape
- Liveness Based Points-to Analysis
- Generalizations to Heap, Arrays, Pointer Arithmetic, and Unions
Must Points-to Information

\[ x = \&a \]

Diagram:

1. \[ x = \&a \]
2. \[ 2 \]
3. \[ 3 \]
4. \[ 4 \]
Must Points-to Information

1. \( x = \&a \)

2.  

3.  

4.  

Diagram showing the flow of pointers and values.
May Points-to Information

```
1. x = &a

2. x = &b

3. 

4. 
```

Diagram:
- Node 1: \( x = &a \)
- Node 2: \( x = &b \)
- Node 3: 
- Node 4: 

Initial state:
- \( a \) and \( b \) are initial values.

Transition:
- From node 1 to node 2:
- From node 2 to node 3:
- From node 3 to node 4:
- From node 4 to node 1:
May Points-to Information

1. $x = \&a$
2. $x = \&b$
3. Empty
4. Empty

Diagram:
- Node 1: $x = \&a$
- Node 2: $x = \&b$
- Node 3: Empty
- Node 4: Empty

Dependencies:
- Node 1 depends on Node 2 and Node 3
- Node 2 depends on Node 1
- Node 3 has no dependencies
- Node 4 has no dependencies
Must Alias Information

1. \( x = \&a \)

2. \( b = x \)

3. 

4. 

5. \( y = b \)
Must Alias Information

1. \( x = \&a \)

2. \( b = x \)

3. \( y = b \)

4. \( a \)

5. \( x \)

6. \( y \)

7. \( b \)
Must Alias Information

1. $x = &a$
2. $b = x$
3. $y = b$
Must Alias Information

\[
x \equiv b \text{ and } b \equiv y \Rightarrow x \equiv y
\]
May Alias Information

1. \( x = \&a \)
2. \( b = \&z \)
3. \( b = x \)
4. \( y = b \)
5. (blank)

\[ a \]
\[ x \]
\[ b \]
\[ y \]
\[ z \]
May Alias Information

1. $x = \& a$
2. $b = \& z$
3. $b = x$
4. $y = b$
5. 

Diagram:
- Node 1: $x = \& a$
- Node 2: $b = \& z$
- Node 3: $b = x$
- Node 4: $y = b$
- Node 5: 

Diagram states:
- Node 1: $a$
- Node 2: $a$
- Node 3: $x$
- Node 4: $y$
- Node 5: 

Arrows indicate the flow of alias information.
May Alias Information

1. \( x = &a \)
2. \( b = &z \)
3. \( b = x \)
4. \( y = b \)
5. (Blank)

Diagram:

- Nodes: \( a, x, b, y, z \)
- Edges indicate alias relationships:
  - \( a \sim x \)
  - \( b \sim y \)
  - \( z \sim x \)
May Alias Information

1. $x = \&a$
2. $b = \&z$
3. $b = x$
4. $y = b$
5. (Empty)

Diagram:

- Node 1: $x = \&a$
- Node 2: $b = \&z$
- Node 3: $b = x$
- Node 4: $y = b$
- Node 5: (Empty)

Symbols:
- $a$
- $x$
- $b$
- $y$
- $z$

Arrow Directions:
- From Node 1 to Node 2
- From Node 2 to Node 3
- From Node 3 to Node 4
- From Node 4 to Node 5

Note: The diagram shows the alias relationships between variables $a$, $b$, $y$, and $z$.
May Alias Information

1. \( x = \&a \)
2. \( b = \&z \)
3. \( b = x \)
4. \( y = b \)
5. (Blank)

The diagram illustrates the may alias information for the given code snippet. The variables and their aliases are shown in the rectangles and boxes.
May Alias Information

1. $x = \&a$
2. $b = \&z$
3. $b = x$
4. $y = b$
5. $x \not\leftarrow b$ and $b \not\leftarrow y \not\Rightarrow x \not\leftarrow y$
Strong and Weak Updates

1. $x = \&a$

2. $y = \&b$
   $w = \&c$

3. $z = \&x$

4. $z = \&y$

5. $*z = \&e$
   $*w = \&e$
**Strong and Weak Updates**

1. \( x = \&a \)
2. \( y = \&b \)
   \( w = \&c \)
3. \( z = \&x \)
4. \( z = \&y \)
5. \( *z = \&e \)
   \( *w = \&e \)

**Weak update**: Modification of \( x \) or \( y \) due to \(*z\) in block 5
Strong and Weak Updates

Weak update: Modification of $x$ or $y$ due to $*z$ in block 5

Strong update: Modification of $c$ due to $*w$ in block 5
Strong and Weak Updates

Weak update: Modification of $x$ or $y$ due to $*z$ in block 5

Strong update: Modification of $c$ due to $*w$ in block 5

How is this concept related to May/Must nature of information?
What About Heap Data?

• Compile time entities, abstract entities, or summarized entities

• Three options:
  ▶ Represent all heap locations by a single abstract heap location
  ▶ Represent all heap locations of a particular type by a single abstract heap location
  ▶ Represent all heap locations allocated at a given memory allocation site by a single abstract heap location

• Summarization: Usually based on the length of pointer expression

• *Initially, we will restrict ourselves to stack and static data*
  
  *We will later introduce heap using the allocation site based abstraction*
Lattice for May Points-to Analysis

Let $P \subseteq \text{Var}$ be the set of pointers. Assume $\text{Var} = \{p, q\}$ and $P = \{p\}$

<table>
<thead>
<tr>
<th>Product View</th>
<th>Mapping view</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Lattice for May Points-to Analysis

Let $P \subseteq \mathbb{Var}$ be the set of pointers. Assume $\mathbb{Var} = \{p, q\}$ and $P = \{p\}$

**Product View**

<table>
<thead>
<tr>
<th>Product View</th>
<th>Mapping view</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$(p, p)$</td>
<td>$(p, p)$</td>
</tr>
<tr>
<td>$(p, q)$</td>
<td>$(p, q)$</td>
</tr>
<tr>
<td>$(p, p), (p, q)$</td>
<td></td>
</tr>
</tbody>
</table>

Data flow values $\subseteq P \times \mathbb{Var}$

Lattice $= (2^{P \times \mathbb{Var}}, \supseteq)$
Let \( P \subseteq \mathbb{V}ar \) be the set of pointers. Assume \( \mathbb{V}ar = \{p, q\} \) and \( P = \{p\} \).

<table>
<thead>
<tr>
<th>Product View</th>
<th>Mapping View</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</td>
<td>( {(p, \emptyset)} )</td>
</tr>
<tr>
<td>( {(p, p)} ) ( {(p, q)} )</td>
<td>( {(p, {p})} ) ( {(p, {q})} )</td>
</tr>
<tr>
<td>( {(p, p), (p, q)} )</td>
<td>( {(p, {p, q})} )</td>
</tr>
</tbody>
</table>

Data flow values \( \subseteq P \times \mathbb{V}ar \)

Lattice = \( (2^{P \times \mathbb{V}ar}, \supseteq) \)

Data flow values \( \in P \rightarrow 2^{\mathbb{V}ar} \)

Lattice = \( (P \rightarrow 2^{\mathbb{V}ar}, \sqsubseteq_{map}) \)
Let $P \subseteq \text{Var}$ be the set of pointers. Assume $\text{Var} = \{p, q\}$ and $P = \{p\}$

**Points-to graph as a list of directed edges**

- Product View
  - $\emptyset$
  - $\{(p, p)\}$
  - $\{(p, p), (p, q)\}$
  - Data flow values $\subseteq P \times \text{Var}$
  - Lattice $= (2^{P \times \text{Var}}, \supseteq)$

- Mapping view
  - $\{(p, \emptyset)\}$
  - $\{(p, \{p\})\}$
  - $\{(p, \{p, q\})\}$
  - Data flow values $\in P \rightarrow 2^{\text{Var}}$
  - Lattice $= (P \rightarrow 2^{\text{Var}}, \subseteq_{map})$
Let $\mathbf{P} \subseteq \mathbb{V}_{\text{ar}}$ be the set of pointers. Assume $\mathbb{V}_{\text{ar}} = \{p, q\}$ and $\mathbf{P} = \{p\}$.

<table>
<thead>
<tr>
<th>Product View</th>
<th>Mapping View</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>${(p, \emptyset)}$</td>
</tr>
<tr>
<td>${(p, p)}$</td>
<td>${(p, {p})}$</td>
</tr>
<tr>
<td>${(p, p), (p, q)}$</td>
<td>${(p, {p, q})}$</td>
</tr>
</tbody>
</table>

Data flow values $\subseteq \mathbf{P} \times \mathbb{V}_{\text{ar}}$

Lattice $= (2^{\mathbf{P} \times \mathbb{V}_{\text{ar}}}, \supseteq)$

Points-to graph as a list of directed edges

Data flow values $\in \mathbf{P} \rightarrow 2^{\mathbb{V}_{\text{ar}}}$

Lattice $= (\mathbf{P} \rightarrow 2^{\mathbb{V}_{\text{ar}}}, \sqsubseteq_{\text{map}})$

Points-to graph as a list of adjacency lists
Lattice for Must Points-to Analysis

Let $P \subseteq \mathbb{V}ar$ be the set of pointers. Assume $\mathbb{V}ar = \{p, q, r\}$ and $P = \{p\}$.

A pointer can point to at most one location.
Lattice for Must Points-to Analysis

Let \( P \subseteq \mathbb{V}_{\text{ar}} \) be the set of pointers. Assume \( \mathbb{V}_{\text{ar}} = \{ p, q, r \} \) and \( P = \{ p \} \)

### Mapping View

\[
\begin{align*}
\{ (p, \hat{\top}) \} \\
\{ (p, p) \} & \quad \{ (p, q) \} & \quad \{ (p, r) \} \\
\{ (p, \hat{\bot}) \}
\end{align*}
\]

### Component Lattice

\[
\begin{aligned}
\hat{\top} & \quad \hat{\bot} \\
p & \quad q & \quad r
\end{aligned}
\]

### Set View

\[
\text{Data flow values} = P \rightarrow \mathbb{V}_{\text{ar}} \cup \{ \hat{\top}, \hat{\bot} \}
\]

\[
\text{Lattice} = \left( 2^P \rightarrow \mathbb{V}_{\text{ar}} \cup \{ \hat{\top}, \hat{\bot} \}, \subseteq_{\text{map}} \right)
\]

A pointer can point to at most one location
Lattice for Must Points-to Analysis

Let \( P \subseteq \mathbb{V}ar \) be the set of pointers. Assume \( \mathbb{V}ar = \{p, q, r\} \) and \( P = \{p\} \)

<table>
<thead>
<tr>
<th>Mapping View</th>
<th>Set View</th>
</tr>
</thead>
<tbody>
<tr>
<td>({(p, \hat{\top})})</td>
<td>({(p, p), (p, q), (p, r)})</td>
</tr>
<tr>
<td>({(p, p)}, {(p, q)}, {(p, r)})</td>
<td>({(p, p)}, {(p, q)}, {(p, r)})</td>
</tr>
<tr>
<td>({(p, \hat{\bot})})</td>
<td>(\emptyset)</td>
</tr>
</tbody>
</table>

Data flow values = \( P \rightarrow \mathbb{V}ar \cup \{\hat{\top}, \hat{\bot}\} \)

Lattice = \( \left( 2^{P \rightarrow \mathbb{V}ar \cup \{\hat{\top}, \hat{\bot}\}}, \sqsubseteq_{\text{map}} \right) \)

A pointer can point to at most one location

Restricted subset of \( P \times \mathbb{V}ar \)

\( \cap \) can be used for \( \sqcap \)
Lattice for Combined May-Must Points-to Analysis (1)

- Consider the following abbreviation of the May-Must lattice \( \hat{L} \)

\[
\begin{array}{c}
\text{Unknown} \\
| \\
\text{No} \\
| \\
\text{Must} \\
| \\
\text{May}
\end{array}
\]

abbreviated as

\[
\begin{array}{c}
\text{un} \\
| \\
\text{no} \\
| \\
\text{mt}
\end{array}
\]

- For \( \text{Var} = \{p, q\} \), \( P = \{p\} \), the May-Must points-to lattice is the product

\[ P \times \text{Var} \times \hat{L} \]

- Some elements are prohibited because of the semantics of \( \text{Must} \)
- If we have \((p,p,mt)\) in a data flow value \( X \in P \times \text{Var} \times \hat{L} \), then
  - we cannot have \((p,q,un), (p,q,mt), \text{or} (p,q,my)\) in \( X \)
  - we can only have \((p,q,no)\) in \( X \)
Lattice for Combined May-Must Points-to Analysis (2)

For $\forall \text{Var} = \{p, q\}, \text{P} = \{p\}$, the May-Must points-to lattice is
For $\forall \text{Var} = \{p, q\}$, $P = \{p\}$, the May-Must points-to lattice is

\[
\{(p, p, \text{un}), (p, q, \text{un})\}
\]

\[
\{(p, p, \text{no}), (p, q, \text{no})\}
\]

\[
\{(p, p, \text{mt}), (p, q, \text{mt})\}
\]

\[
\{(p, p, \text{my}), (p, q, \text{my})\}
\]

\[
\{(p, p, \text{my}), (p, q, \text{my})\}
\]

\[
\{(p, p, \text{un}), (p, q, un)\}
\]

\[
\{(p, p, \text{no}), (p, q, un)\}
\]

\[
\{(p, p, \text{mt}), (p, q, un)\}
\]

\[
\{(p, p, \text{no}), (p, q, \text{my})\}
\]

\[
\{(p, p, \text{mt}), (p, q, \text{no})\}
\]

\[
\{(p, p, \text{my}), (p, q, \text{no})\}
\]
For $\text{Var} = \{p, q\}$, $P = \{p\}$, the May-Must points-to lattice is

\[
\{(p,p,un), (p,q,un)\}
\]

\[
\{(p,p,un), (p,q,mt)\}
\]

\[
\{(p,p,mt), (p,q,un)\}
\]

\[
\{(p,p,mt), (p,q,mt)\}
\]

Prohibited

Allowed
Lattice for Combined May-Must Points-to Analysis (2)

For \( \text{Var} = \{p, q\} \), \( \mathbf{P} = \{p\} \), the May-Must points-to lattice is

\[
\{(p, p, \text{un}), (p, q, \text{un})\}
\]

\[
\{(p, p, \text{un}), (p, q, \text{no})\}
\]

\[
\{(p, p, \text{no}), (p, q, \text{no})\}
\]

\[
\{(p, p, \text{mt}), (p, q, \text{no})\}
\]

\[
\{(p, p, \text{mt}), (p, q, \text{un})\}
\]

\[
\{(p, p, \text{my}), (p, q, \text{no})\}
\]

\[
\{(p, p, \text{my}), (p, q, \text{un})\}
\]

\[
\{(p, p, \text{my}), (p, q, \text{my})\}
\]

\[
\{(p, p, \text{no}), (p, q, \text{no})\}
\]

\[
\{(p, p, \text{no}), (p, q, \text{my})\}
\]

\[
\{(p, p, \text{no}), (p, q, mt)\}
\]

\[
\{(p, p, \text{un}), (p, q, my)\}
\]

\[
\{(p, p, \text{my}), (p, q, mt)\}
\]

\[
\{(p, p, \text{mt}), (p, q, my)\}
\]

\[
\{(p, p, \text{my}), (p, q, no)\}
\]

\[
\{(p, p, \text{mt}), (p, q, no)\}
\]

Allowed
Lattice for Combined May-Must Points-to Analysis (2)

For $\forall \text{Var} = \{p, q\}$, $\mathbf{P} = \{p\}$, the May-Must points-to lattice is

\[
\begin{align*}
\{(p,p,un), (p,q,un)\} \\
\{(p,p,un), (p,q,no)\} & \quad \{(p,p,mt), (p,q,no)\} & \quad \{(p,p,my), (p,q,no)\} & \quad \{(p,p,un), (p,q,my)\} & \quad \{(p,p,no), (p,q,mt)\} \\
\{(p,p,no), (p,q,no)\} & \quad \{(p,p,mt), (p,q,no)\} & \quad \{(p,p,my), (p,q,no)\} & \quad \{(p,p,un), (p,q,my)\} & \quad \{(p,p,no), (p,q,mt)\} \\
\{(p,p,my), (p,q,my)\} & \quad \{(p,p,my), (p,q,no)\} & \quad \{(p,p,my), (p,q,my)\} & \quad \{(p,p,my), (p,q,no)\} & \quad \{(p,p,my), (p,q,my)\} \\
\end{align*}
\]
May and Must Analysis for Killing Points-to Information (1)

**May Points-to Analysis**

```plaintext
1 a = &b
2 c = &a
4 *c = &e
```

**Must Points-to Analysis**

```plaintext
3
5
```

Sep 2017

IIT Bombay
May and Must Analysis for Killing Points-to Information (1)

May Points-to Analysis

- \((a, b)\) should be in \(\text{May}In_5\)
  - Holds along path 1-3-4
- Block 4 should not kill \((a, b)\)
- Possible if pointee set of \(c\) is \(\emptyset\)
- However, \(\text{May}In_4\) contains \((c, a)\)

Must Points-to Analysis

\[
\begin{align*}
1 &: a = &b \\
2 &: c = &a \\
4 &: *c = &e \\
5 &:
\end{align*}
\]
May and Must Analysis for Killing Points-to Information (1)

**May Points-to Analysis**
- \((a, b)\) should be in \(\text{MayIn}_5\)
  - Holds along path 1-3-4
- Block 4 should not kill \((a, b)\)
- Possible if pointee set of \(c\) is \(\emptyset\)
- However, \(\text{MayIn}_4\) contains \((c, a)\)

**Must Points-to Analysis**
- \((a, b)\) should not be in \(\text{MustIn}_5\)
  - Does not hold along path 1-2-4
- Block 4 should kill \((a, b)\)
- Possible if pointee set of \(c\) is \(\{a\}\)
- However, \(\text{MustIn}_4\) contains \((a, b)\)
May and Must Analysis for Killing Points-to Information (1)

**May Points-to Analysis**

- $(a, b)$ should be in $\text{MayIn}_5$
  
  Holds along path 1-3-4

- Block 4 should not kill $(a, b)$

- Possible if pointee set of $c$ is $\emptyset$ (Use $\text{MustIn}_4$)

- However, $\text{MayIn}_4$ contains $(c, a)$

For killing points-to information through indirection,

- **Must** points-to analysis should identify pointees of $c$ using $\text{MayIn}_4$

- **May** points-to analysis should identify pointees of $c$ using $\text{MustIn}_4$

**Must Points-to Analysis**

- $(a, b)$ should not be in $\text{MustIn}_5$

  Does not hold along path 1-2-4

- Block 4 should kill $(a, b)$

- Possible if pointee set of $c$ is $\{a\}$ (Use $\text{MayIn}_4$)

- However, $\text{MustIn}_4$ contains $(a, b)$
May and Must Analysis for Killing Points-to Information (2)

- May Points-to analysis should remove a May points-to pair
  - only if it must be removed along all paths

  Kill should remove only strong updates
  \[ \Rightarrow \] should use Must Points-to information

- Must Points-to analysis should remove a Must points-to pair
  - if it can be removed along any path

  Kill should remove all weak updates
  \[ \Rightarrow \] should use May Points-to information
Discovering Must Points-to Information from May Points-to Information

1. \( a = \& b \)
   \( b = \& e \)

2. \( c = \& a \)

3. \( \)  

4. \( \)
Discovering Must Points-to Information from May Points-to Information

1. $a = \& b$
   $b = \& e$

2. $c = \& a$

3. 

4. 

- **Bl.** every pointer points to “?”
Discovering Must Points-to Information from May Points-to Information

- $a = \&b$
- $b = \&e$
- $c = \&a$
- $\ast a = \&e$
- $\ast c = \&d$
- $b = \?$
- $e = \?$
- $c = \?$
- $a = \?$

- BI. every pointer points to "?"

Sep 2017 IIT Bombay
Discovering Must Points-to Information from May Points-to Information

1. \[ a = \& b \]
   \[ b = \& e \]

2. \[ c = \& a \]

3. BI. every pointer points to "?"

4. Perform usual may points-to analysis
Discovering Must Points-to Information from May Points-to Information

1. \( a = \&b \), \( b = \&e \)

2. \( c = \&a \)

3. Perform usual may points-to analysis

4. BI. every pointer points to “?”

\[ a \rightarrow ? \rightarrow b \rightarrow ? \rightarrow c \rightarrow ? \rightarrow e \rightarrow ? \]
Discovering Must Points-to Information from May Points-to Information

- BI. every pointer points to “?”
- Perform usual may points-to analysis

```
1
a = &b
b = &e

2
c = &a

3

4
```
Discovering Must Points-to Information from May Points-to Information

1. \( a = \& b \)
   \( b = \& e \)

2. \( c = \& a \)

3. 

4. 

- BI. every pointer points to “?”
- Perform usual may points-to analysis
- Since c has multiple pointees, it is a MAY relation
Discovering Must Points-to Information from May Points-to Information

1. \( a = \& b \)
   \( b = \& e \)

2. \( c = \& a \)

3. 

4. 

- **Bi.** every pointer points to “?”
- Perform usual may points-to analysis
- Since \( c \) has multiple pointees, it is a MAY relation
- Since \( a \) has a single pointee, it is a MUST relation
Relevant Algebraic Operations on Relations (1)

- Let $P \subseteq \text{Var}$ be the set of pointer variables
- May-points-to information: $\mathcal{A} = \langle 2^{P \times \text{Var}}, \supseteq \rangle$
- Standard algebraic operations on points-to relations
  - Given relation $R \subseteq P \times \text{Var}$ and $X \subseteq P$,
    - Relation **application** $R \times X = \{ v \mid u \in X \land (u, v) \in R \}$
    - Relation **restriction** $(R|_X) R|_X = \{ (u, v) \in R \mid u \in X \}$
Relevant Algebraic Operations on Relations (1)

- Let \( P \subseteq \mathbb{V}ar \) be the set of pointer variables.
- May-points-to information: \( \mathcal{A} = \langle 2^{P \times \mathbb{V}ar}, \supseteq \rangle \)
- Standard algebraic operations on points-to relations:
  Given relation \( R \subseteq P \times \mathbb{V}ar \) and \( X \subseteq P \),
  - Relation application \( R \ X = \{ v \mid u \in X \wedge (u, v) \in R \} \)
    (Find out the pointees of the pointers contained in \( X \))
  - Relation restriction \( (R|_X) \ R|_X = \{(u, v) \in R \mid u \in X\} \)
Relevant Algebraic Operations on Relations (1)

- Let $\mathcal{P} \subseteq \text{Var}$ be the set of pointer variables

- May-points-to information: $\mathcal{A} = \langle 2^{\mathcal{P} \times \text{Var}}, \supseteq \rangle$

- Standard algebraic operations on points-to relations
  
  Given relation $R \subseteq \mathcal{P} \times \text{Var}$ and $X \subseteq \mathcal{P}$,
  
  ▶ Relation *application* $R X = \{ v \mid u \in X \land (u, v) \in R \}$
    (Find out the pointees of the pointers contained in $X$)

  ▶ Relation *restriction* $(R |_X) R |_X = \{ (u, v) \in R \mid u \in X \}$
    (Restrict the relation only to the pointers contained in $X$ by removing points-to information of other pointers)
Let

\[ \text{Var} = \{a, b, c, d, e, f, g, ?\} \]
\[ P = \{a, b, c, d, e\} \]
\[ R = \{(a, b), (a, c), (b, d), (c, e), (c, g), (d, a), (e, ?)\} \]
\[ X = \{a, c\} \]

Then,

\[ R \times X = \{v \mid u \in X \land (u, v) \in R\} \]

\[ R|_X = \{(u, v) \in R \mid u \in X\} \]
Let

\[ \text{Var} = \{a, b, c, d, e, f, g, ?\} \]
\[ P = \{a, b, c, d, e\} \]
\[ R = \{(a, b), (a, c), (b, d), (c, e), (c, g), (d, a), (e, ?)\} \]
\[ X = \{a, c\} \]

Then,

\[ R \times X = \{v \mid u \in X \land (u, v) \in R\} \]
\[ = \{b, c, e, g\} \]
\[ R|_X = \{(u, v) \in R \mid u \in X\} \]
Let

\[ \text{Var} = \{a, b, c, d, e, f, g, ?\} \]
\[ P = \{a, b, c, d, e\} \]
\[ R = \{(a, b), (a, c), (b, d), (c, e), (c, g), (d, a), (e, ?)\} \]
\[ X = \{a, c\} \]

Then,

\[ R \times X = \{v | u \in X \land (u, v) \in R\} \]
\[ = \{b, c, e, g\} \]
\[ R|_X = \{(u, v) \in R | u \in X\} \]
\[ = \{(a, b), (a, c), (c, e), (c, g)\} \]
Points-to Analysis Data Flow Equations

\[ \text{Ain}_{n} = \begin{cases} \mathbb{V} \text{ar} \times \{?\} & \text{if } n \text{ is Start}_{p} \\ \bigcup_{p \in \text{pred}(n)} \text{Aout}_{p} & \text{otherwise} \end{cases} \]

\[ \text{Aout}_{n} = \left( \text{Ain}_{n} - \left( \text{Kill}_{n} \times \mathbb{V} \text{ar} \right) \right) \cup \left( \text{Def}_{n} \times \text{Pointee}_{n} \right) \]

- **Ain/Aout**: sets of mAy points-to pairs
- **Kill}_{n}, **Def}_{n}, and **Pointee}_{n} are defined in terms of **Ain}_{n}
Points-to Analysis Data Flow Equations

\[
Ain_n = \begin{cases} 
\forall var \times \{?\} & n \text{ is } \text{Start}_p \\
\bigcup_{p \in \text{pred}(n)} Aout_p & \text{otherwise}
\end{cases}
\]

\[
Aout_n = (Ain_n - (\text{Kill}_n \times \forall var)) \cup (\text{Def}_n \times \text{Pointee}_n)
\]

- \(Ain/Aout\): sets of mAy points-to pairs
- \(\text{Kill}_n, \text{Def}_n, \text{and Pointee}_n\) are defined in terms of \(Ain_n\)

Pointers whose points-to relations should be removed
Points-to Analysis Data Flow Equations

\[ Ain_n = \begin{cases} 
\mathcal{V} \times \{?\} & \text{n is Start}_p \\
\bigcup_{p \in \text{pred}(n)} Aout_p & \text{otherwise}
\end{cases} \]

\[ Aout_n = \left( Ain_n - (\text{Kill}_n \times \mathcal{V}) \right) \cup (\text{Def}_n \times \text{Pointee}_n) \]

- \( Ain/Aout \): sets of mAy points-to pairs
- \( \text{Kill}_n, \text{Def}_n, \) and \( \text{Pointee}_n \) are defined in terms of \( Ain_n \)

Pointers that are defined (i.e. pointers in which addresses are stored)
Points-to Analysis Data Flow Equations

\[ \text{Ain}_n = \begin{cases} \forall \text{Var} \times \{?\} & \text{n is Start}_p \\ \cup & \text{otherwise} \\ \bigcup_{p \in \text{pred}(n)} \text{Aout}_p \end{cases} \]

\[ \text{Aout}_n = \left( \text{Ain}_n - \left( \text{Kill}_n \times \forall \text{Var} \right) \right) \cup \left( \text{Def}_n \times \text{Pointee}_n \right) \]

- \textit{Ain/Aout}: sets of mAy points-to pairs
- \textit{Kill}_n, \textit{Def}_n, and \textit{Pointee}_n are defined in terms of \textit{Ain}_n
Points-to Analysis Data Flow Equations

\[ \text{Ain}_n = \begin{cases} 
\text{Var} \times \{?\} & \text{n is Start}_p \\
\bigcup_{p \in \text{pred}(n)} \text{Aout}_p & \text{otherwise}
\end{cases} \]

\[ \text{Aout}_n = \left( \text{Ain}_n - \left( \text{Kill}_n \times \text{Var} \right) \right) \bigcup \left( \text{Def}_n \times \text{Pointee}_n \right) \]

- Ain/Aout: sets of mAy points-to pairs
- Kill$_n$, Def$_n$, and Pointee$_n$ are defined in terms of Ain$_n$
Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$ (denoted $A$)

<table>
<thead>
<tr>
<th></th>
<th>$Def_n$</th>
<th>$Kill_n$</th>
<th>$Pointee_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$use \ x$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x = &amp;a$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x = y$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x = \ast y$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\ast x = y$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$ (denoted $A$)

<table>
<thead>
<tr>
<th></th>
<th>$Def_n$</th>
<th>$Kill_n$</th>
<th>$Pointee_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$use \ x$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x = &amp;a$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x = y$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x = *y$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$*x = y$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pointers that are defined (i.e. pointers in which addresses are stored)
Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$ (denoted $A$)

<table>
<thead>
<tr>
<th></th>
<th>$Def_n$</th>
<th>$Kill_n$</th>
<th>$\text{Pointee}_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$use \ x$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x = &amp;a$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x = y$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x = *y$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$*x = y$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pointees (i.e. locations whose addresses are stored)
### Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$ (denoted $A$)

<table>
<thead>
<tr>
<th></th>
<th>$Def_n$</th>
<th>$Kill_n$</th>
<th>$Pointee_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$use \ x$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x = &amp;a$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x = y$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x = *y$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$*x = y$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>other</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pointers whose points-to relations should be removed.
Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$ (denoted $A$)

<table>
<thead>
<tr>
<th></th>
<th>$Def_n$</th>
<th>$Kill_n$</th>
<th>$Pointee_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>use $x$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$x = &amp;a$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x = y$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x = *y$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$*x = y$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Extractor Functions for Points-to Analysis

Values defined in terms of $A_{in_n}$ (denoted $A$)

<table>
<thead>
<tr>
<th></th>
<th>$Def_n$</th>
<th>$Kill_n$</th>
<th>$Pointee_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>use $x$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$x = &amp;a$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>${a}$</td>
</tr>
<tr>
<td>$x = y$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x = *y$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$*x = y$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$ (denoted $A$)

<table>
<thead>
<tr>
<th></th>
<th>$Def_n$</th>
<th>$Kill_n$</th>
<th>$Pointee_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>use $x$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$x = &amp;a$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>${a}$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A{y}$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$*x = y$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pointees of $y$ in $Ain_n$ are the targets of defined pointers
Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$ (denoted $A$)

<table>
<thead>
<tr>
<th></th>
<th>$Def_n$</th>
<th>$Kill_n$</th>
<th>$Pointee_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$use \ x$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$x = &amp;a$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>${a}$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A{y}$</td>
</tr>
<tr>
<td>$x = \ast y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A(A{y} \cap P)$</td>
</tr>
<tr>
<td>$\ast x = y$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pointees of those pointees of $y$ in $Ain_n$ which are pointers
Extractor Functions for Points-to Analysis

Values defined in terms of $A_{in}$ (denoted $A$)

<table>
<thead>
<tr>
<th></th>
<th>$Def_n$</th>
<th>$Kill_n$</th>
<th>$Pointee_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>use $x$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$x = &amp;a$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>${a}$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A{y}$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A(A{y} \cap P)$</td>
</tr>
<tr>
<td>$*x = y$</td>
<td>$A{x} \cap P$</td>
<td>$Must(A){x} \cap P$</td>
<td>$A{y}$</td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pointees of $x$ in $A_{in}$ receive new addresses
### Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$:

<table>
<thead>
<tr>
<th></th>
<th>$Def_n$</th>
<th>$K_{niln}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$use \ x$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$x = &amp;a$</td>
<td>${x}$</td>
<td>${x}$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>${x}$</td>
<td>${x}$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>${x}$</td>
<td>${x}$</td>
</tr>
<tr>
<td>$*x = y$</td>
<td>$A{x} \cap P$</td>
<td>$\text{Must}(A){x} \cap P$</td>
</tr>
<tr>
<td>other</td>
<td></td>
<td>$A{y}$</td>
</tr>
</tbody>
</table>

$Must(R) = \bigcup_{z \in P} \{z\} \times \begin{cases} \{w\} & R\{z\} = \{w\} \land w \neq ? \\ \emptyset & \text{otherwise} \end{cases}$
Extractor Functions for Points-to Analysis

Values defined in terms of $A^m_{in}$:

$$\text{Def}_n \quad K_{lll}_n$$

<table>
<thead>
<tr>
<th></th>
<th>$\text{Def}_n$</th>
<th>$K_{lll}_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>use $x$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$x = &amp; a$</td>
<td>${x}$</td>
<td>${x}$ ${a}$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>${x}$</td>
<td>${x}$ $A{y}$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>${x}$</td>
<td>${x}$ $A(A{y} \cap P)$</td>
</tr>
<tr>
<td>$*x = y$</td>
<td>$A{x} \cap P$</td>
<td>$\text{Must}(A){x} \cap P$ $A{y}$</td>
</tr>
<tr>
<td>other</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

Strong update using must-points-to information computed from $A^m_{in}$:

$$\text{Must}(R) = \bigcup_{z \in P} \{z\} \times \left\{ \begin{array}{l} \{w\} \quad R\{z\} = \{w\} \land w \neq ? \\ \emptyset \quad \text{otherwise} \end{array} \right.$$
Extractor Functions for Points-to Analysis

Values defined in terms of $A_{in}$

<table>
<thead>
<tr>
<th>$use \ x$</th>
<th>$Def_{n}$</th>
<th>$K_{lll_{n}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = &amp;a$</td>
<td>${ x }$</td>
<td>${ x }$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>${ x }$</td>
<td>${ x }$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>${ x }$</td>
<td>${ x }$</td>
</tr>
<tr>
<td>$*x = y$</td>
<td>$A{ x } \cap P$</td>
<td>$Must(A){ x } \cap P$</td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$Must(R) = \bigcup_{z \in P} \{ z \} \times \begin{cases} \{ w \} & R\{ z \} = \{ w \} \land w \neq ? \\ \emptyset & \text{otherwise} \end{cases}$

$z$ has a single pointee $w$ in must-points-to relation

Strong update using must-points-to information computed from $A_{in}$
Extractor Functions for Points-to Analysis

Values defined in terms of \( A_{in} \) (denoted \( A_{in} \)).

<table>
<thead>
<tr>
<th></th>
<th>( Def_n )</th>
<th>( K_{lin}_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>use ( x )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( x = &amp;a )</td>
<td>( { x } )</td>
<td>( { x } )</td>
</tr>
<tr>
<td>( x = y )</td>
<td>( { x } )</td>
<td>( { x } )</td>
</tr>
<tr>
<td>( x = *y )</td>
<td>( { x } )</td>
<td>( { x } )</td>
</tr>
<tr>
<td>( *x = y )</td>
<td>( A { x } \cap P )</td>
<td>( Must ( A ) { x } \cap P )</td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Strong update using must-points-to information computed from \( A_{in} \).

\[
Must(R) = \bigcup_{z \in P} \{ z \} \times \begin{cases} 
\{ w \} & R\{ z \} = \{ w \} \land w \neq ? \\
\emptyset & \text{otherwise}
\end{cases}
\]

\( z \) has no pointee in must-points-to relation
Extractor Functions for Points-to Analysis

Values defined in terms of \( A \in n \) (denoted \( A \))

<table>
<thead>
<tr>
<th>( \text{use } x )</th>
<th>( \text{Def}_n )</th>
<th>( \text{Kill}_n )</th>
<th>( \text{Pointee}_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = &amp; a )</td>
<td>{( x )}</td>
<td>{( x )}</td>
<td>{( a )}</td>
</tr>
<tr>
<td>( x = y )</td>
<td>{( x )}</td>
<td>{( x )}</td>
<td>( A{y} )</td>
</tr>
<tr>
<td>( x = *y )</td>
<td>{( x )}</td>
<td>{( x )}</td>
<td>( A(A{y} \cap P) )</td>
</tr>
<tr>
<td>( ^*x = y )</td>
<td>( A{x} \cap P )</td>
<td>( \text{Must}(A){x} \cap P )</td>
<td>( A{y} )</td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{Must}(R) = \bigcup_{z \in P} \{z\} \times \begin{cases} \{w\} & R\{z\} = \{w\} \land w \neq ? \\
\emptyset & \text{otherwise} \end{cases}
\]
# Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$ (denoted $A$)

<table>
<thead>
<tr>
<th></th>
<th>$Def_n$</th>
<th>$Kill_n$</th>
<th>$Pointee_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$use \ x$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$x = &amp;a$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>${a}$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A{y}$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A(A{y} \cap P)$</td>
</tr>
<tr>
<td>$*x = y$</td>
<td>$A{x} \cap P$</td>
<td>$Must(A){x} \cap P$</td>
<td>$A{y}$</td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$Must(R) = \bigcup_{z \in P} \{z\} \times \begin{cases} \{w\} & R\{z\} = \{w\} \land w \neq ? \\ \emptyset & \text{otherwise} \end{cases}$

Pointees of $y$ in $Ain_n$ are the targets of defined pointers.
Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$ (denoted $A$)

<table>
<thead>
<tr>
<th></th>
<th>$Def_n$</th>
<th>$Kill_n$</th>
<th>$Pointee_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>use x</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$x = &amp;a$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>${a}$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A{y}$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A(A{y} \cap P)$</td>
</tr>
<tr>
<td>$*x = y$</td>
<td>$A{x} \cap P$</td>
<td>$Must(A){x} \cap P$</td>
<td>$A{y}$</td>
</tr>
<tr>
<td>other</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

$$Must(R) = \bigcup_{z \in P} \{z\} \times \begin{cases} \{w\} & R\{z\} = \{w\} \land w \neq ? \\ \emptyset & \text{otherwise} \end{cases}$$
Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$ (denoted $A$)

<table>
<thead>
<tr>
<th></th>
<th>$Def_n$</th>
<th>$Kill_n$</th>
<th>$Pointee_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>use $x$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$x = &amp;a$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>${a}$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A{y}$</td>
</tr>
<tr>
<td>$x = \ast y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A(A{y} \cap P)$</td>
</tr>
<tr>
<td>$\ast x = y$</td>
<td>$A{x} \cap P$</td>
<td>$Must(A){x} \cap P$</td>
<td>$A{y}$</td>
</tr>
<tr>
<td>other</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

$$Must(R) = \bigcup_{z \in P} \{z\} \times \begin{cases} \{w\} & R\{z\} = \{w\} \land w \neq ? \\ \emptyset & \text{otherwise} \end{cases}$$
### Extractor Functions for Points-to Analysis

Values defined in terms of $Ain_n$ (denoted $A$)

<table>
<thead>
<tr>
<th></th>
<th>$Def_n$</th>
<th>$Kill_n$</th>
<th>$Pointee_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$use \ x$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$x = &amp;a$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>${a}$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A{y}$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A(A{y} \cap P)$</td>
</tr>
<tr>
<td>$*x = y$</td>
<td>$A{x} \cap P$</td>
<td>$Must(A){x} \cap P$</td>
<td>$A{y}$</td>
</tr>
<tr>
<td>other</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

$$Must(R) = \bigcup_{z \in P} \{z\} \times \left\{ \begin{array}{ll} \{w\} & R\{z\} = \{w\} \land w \neq \_ \\ \emptyset & \text{otherwise} \end{array} \right.$$
An Example of Flow Sensitive May Points-to Analysis

Assume that the program is type correct.

$n_1$

$x = &y$

$y = &z$

$z = &u$

$n_2$

$*z = y$

$n_3$

$z = y$

$n_4$

$*u = &x$

$n_5$

$*y = &y$
An Example of Flow Sensitive May Points-to Analysis

Assume that the program is type correct

\[\begin{align*}
x &= \& y \\
y &= \& z \\
z &= \& u
\end{align*}\]

\[\begin{align*}
\ast z &= y \\
z &= y
\end{align*}\]
An Example of Flow Sensitive May Points-to Analysis

Assume that the program is type correct

1. \( x = \& y \)
2. \( y = \& z \)
3. \( z = \& u \)
4. \( *z = y \)
5. \( *u = \& x \)
6. \( *y = \& y \)
7. \( \text{Program Output} \)

Sep 2017 IIT Bombay
Assume that the program is type correct.

\[
x = \&y \\
y = \&z \\
z = \&u
\]

\[
x \\y \\z \\u
\]

\[
x = \&y \\
y = \&z \\
z = \&u
\]

\[
x \\y \\z \\u
\]
An Example of Flow Sensitive May Points-to Analysis

Assume that the program is type correct.
An Example of Flow Sensitive May Points-to Analysis

Assume that the program is type correct.

$\quad x = \& y$
$\quad y = \& z$
$\quad z = \& u$

$n_1$

$n_2$  $\ast z = y$

$n_3$  $z = y$

$n_4$  $\ast u = \& x$

$n_5$  $\ast y = \& y$
An Example of Flow Sensitive May Points-to Analysis

Assume that the program is type correct

\[
\begin{align*}
  x &= \&y \\
  y &= \&z \\
  z &= \&u \\
  \star z &= y \\
  \star u &= \&x \\
  \star y &= \&y
\end{align*}
\]

Sep 2017 IIT Bombay
An Example of Flow Sensitive May Points-to Analysis

Assume that the program is type correct

Weak Update
An Example of Flow Sensitive May Points-to Analysis

Assume that the program is type correct

\[ x = \&y \]
\[ y = \&z \]
\[ z = \&u \]

\[ *z = y \]

\[ *u = \&x \]

\[ *y = \&y \]
An Example of Flow Sensitive May Points-to Analysis

Assume that the program is type correct

Weak Update

Strong Update

Sep 2017 IIT Bombay
Compute May and Must points-to information

```c
if (...) 
    p = &x;
else
    p = &y;

x = &a;
y = &b;
*p = &c;
*y = &a;
```
Non-Distributivity of Points-to Analysis

May Points-to

\[ n_1 \]

\[ n_2 : x = &z \]

\[ n_3 : y = &w \]

\[ n_4 : *x = y \]

Must Points-to

\[ n_1 \]

\[ n_2 : \]

\[ b = &c \]

\[ c = &d \]

\[ n_3 : \]

\[ b = &e \]

\[ e = &d \]

\[ n_4 : a = *b \]
Non-Distributivity of Points-to Analysis

May Points-to

Must Points-to

z → w is spurious
Non-Distributivity of Points-to Analysis

May Points-to

\[ n_1 \]

\[ n_2 \rightarrow x = \& z \]

\[ n_3 \rightarrow y = \& w \]

\[ n_4 \rightarrow *x = y \]

\[ z \rightarrow w \text{ is spurious} \]

Must Points-to

\[ n_1 \]

\[ n_2 \rightarrow b = \& c \]

\[ n_3 \rightarrow c = \& d \]

\[ n_4 \rightarrow a = *b \]

\[ b = \& e \]

\[ e = \& d \]

\[ a \rightarrow d \text{ is missing} \]
An Outline of Pointer Analysis Coverage

- The larger perspective
- Comparing Points-to and Alias information
- Flow Insensitive Points-to Analysis
- Flow Sensitive Points-to Analysis
- Pointer Analyses: An Engineer’s Landscape
- Liveness Based Points-to Analysis
- Generalizations to Heap, Arrays, Pointer Arithmetic, and Unions
An Example of Flow Insensitive May Points-to Analysis

\[
\begin{align*}
\text{n}_1: & \quad x = & y \\
& \quad y = & z \\
& \quad z = & u
\end{align*}
\]

\[
\begin{align*}
\text{n}_2: & \quad *z = & y \\
\text{n}_3: & \quad z = & y
\end{align*}
\]

\[
\begin{align*}
\text{n}_4: & \quad y = & x \\
& \quad \text{use } & u \\
& \quad \text{use } & x
\end{align*}
\]

Andersen’s Points-to Graph

Steensgaard’s Points-to Graph
An Example of Flow Insensitive May Points-to Analysis

Andersen’s Points-to Graph

Steensgaard’s Points-to Graph

\[
\begin{align*}
  n_1 & : x = & y \\
       & y = & z \\
       & z = & u \\

  n_2 & : *z = & y \\

  n_3 & : z = & y \\

  n_4 & : y = & x \\
       & use u \\
       & use x
\end{align*}
\]
An Example of Flow Insensitive May Points-to Analysis

An Example of Flow Insensitive May Points-to Analysis

\[ x = \& y \\ y = \& z \\ z = \& u \]

\[ *z = y \]

\[ y = \& x \\ use u \\ use x \]

Andersen’s Points-to Graph

Steensgaard’s Points-to Graph
An Example of Flow Sensitive May Points-to Analysis

For simplicity, we ignore the $BI$ with “?”

```
x = &y
y = &z
z = &u
```

```
*z = y
```

```
y = &x
use u
use x
```
An Example of Flow Sensitive May Points-to Analysis

For simplicity, we ignore the BI with “?”

For example:

\[
\begin{align*}
& x = \&y \\
& y = \&z \\
& z = \&u
\end{align*}
\]
An Example of Flow Sensitive May Points-to Analysis

For simplicity, we ignore the \( BI \) with "?"

\[
\begin{align*}
  x &= & y \\
  y &= & z \\
  z &= & u
\end{align*}
\]

\[
\begin{align*}
  y &= & x \\
  use \ u \\
  use \ x
\end{align*}
\]

\[
\begin{align*}
  \star z &= & y \\
  \star z &= & y
\end{align*}
\]

\[
\begin{align*}
  z &= & y \\
  \emptyset
\end{align*}
\]

\[
\begin{align*}
  x \rightarrow y \rightarrow z \rightarrow u
\end{align*}
\]
An Example of Flow Sensitive May Points-to Analysis

For simplicity, we ignore the $BI$ with “?”

$x = \&y$
$y = \&z$
$z = \&u$

$x \rightarrow y \rightarrow z \rightarrow u$

$\ast z = y$

$\emptyset$

$y = \&x$

$\text{use } u$

$\text{use } x$
An Example of Flow Sensitive May Points-to Analysis

For simplicity, we ignore the BI with "?"

1. $x = \& y$
2. $y = \& z$
3. $z = \& u$

$\forall$
For simplicity, we ignore the BI with “?”
An Example of Flow Sensitive May Points-to Analysis

For simplicity, we ignore the BI with “?”

For simplicity, we ignore the BI with “?”

For simplicity, we ignore the BI with “?”

For simplicity, we ignore the BI with “?”

For simplicity, we ignore the BI with “?”

For simplicity, we ignore the BI with “?”

For simplicity, we ignore the BI with “?”

For simplicity, we ignore the BI with “?”

For simplicity, we ignore the BI with “?”

For simplicity, we ignore the BI with “?”

For simplicity, we ignore the BI with “?”

For simplicity, we ignore the BI with “?”

For simplicity, we ignore the BI with “?”

For simplicity, we ignore the BI with “?”
An Example of Flow Sensitive May Points-to Analysis

For simplicity, we ignore the BI with “?”
An Example of Flow Sensitive May Points-to Analysis

For simplicity, we ignore the BI with "?"

\[
x = \& y \\
y = \& z \\
z = \& u
\]

For simplicity, we ignore the BI with "?"
Context Sensitivity in Interprocedural Analysis

\[ a = \& b \]

\[ c = \& d \]

\[ f_r \]

\[ c_i \]

\[ c_j \]

\[ R_i \]

\[ R_j \]

\[ C_i \]

\[ C_j \]

\[ End_s \]

\[ End_t \]
Context Sensitivity in Interprocedural Analysis

\begin{figure}[h]
\begin{center}
\begin{tikzpicture}
  \node (Start_s) at (0,0) {	extit{Start}_s};
  \node (a_ab) at (0,-2) {$a = \& b$};
  \node (Ci) at (0,-4) {$C_i$};
  \node (Ri) at (0,-6) {$R_i$};
  \node (End_s) at (0,-8) {	extit{End}_s};
  \node (Start_r) at (4,0) {	extit{Start}_r};
  \node (Cj) at (4,-2) {$C_j$};
  \node (Rj) at (4,-4) {$R_j$};
  \node (End_r) at (4,-6) {	extit{End}_r};
  \node (Start_t) at (8,0) {	extit{Start}_t};
  \node (Cj) at (8,-2) {$C_j$};
  \node (Rj) at (8,-4) {$R_j$};
  \node (End_t) at (8,-6) {	extit{End}_t};

  \draw[->] (Start_s) -- (a_ab);
  \draw[->] (a_ab) -- (Ci);
  \draw[->] (Ci) -- (Ri);
  \draw[->] (Ri) -- (End_s);
  \draw[->] (a_ab) -- (b);

  \draw[->] (Start_r) -- (Cj);
  \draw[->] (Cj) -- (Rj);
  \draw[->] (Rj) -- (End_r);
  \draw[->] (Start_t) -- (Cj);
  \draw[->] (Cj) -- (Rj);
  \draw[->] (Rj) -- (End_t);

  \draw[->] (a) -- (b);

  \draw[->] (Ci) -- (a);
  \draw[->] (a) -- (Ci);
  \draw[->] (Ci) -- (b);
  \draw[->] (b) -- (Ci);

  \draw[->] (Ri) -- (f_r);
  \draw[->] (f_r) -- (Ri);

  \draw[->] (Ci) -- (c_i);
  \draw[->] (c_i) -- (Ci);

  \draw[->] (Ci) -- (a); \draw[->] (b) -- (Ci);

\end{tikzpicture}
\end{center}
\end{figure}

\texttt{Start}_s, \texttt{End}_s, \texttt{Start}_r, \texttt{End}_r, \texttt{Start}_t, \texttt{End}_t

\text{context sensitivity in interprocedural analysis}

\text{context sensitivity in interprocedural analysis}

\text{context sensitivity in interprocedural analysis}

\text{context sensitivity in interprocedural analysis}
Context Sensitivity in Interprocedural Analysis
Context Sensitivity in Interprocedural Analysis

\[
\begin{align*}
Start_s & : a = \& b \\
C_i & : c_i \leftarrow a \rightarrow b \rightarrow C_i \\
R_i & : f_r \leftarrow \rightarrow R_i \\
End_s & \\

t & : c = \& d \\
C_j & : c \rightarrow d \rightarrow c_j \\
R_j & : \rightarrow R_j \\
End_t &
\end{align*}
\]
Context Sensitivity in Interprocedural Analysis

\[
\begin{align*}
\text{Start}_s & \quad a = \&b \\
C_i & \quad c_i \\
R_i & \quad f_r \\
\text{End}_s \\
\text{Start}_r & \quad c = \&d \\
C_j & \quad c_j \\
R_j & \quad f_r \\
\text{End}_r \\
\text{Start}_t & \quad a \rightarrow b \\
c & \rightarrow d \\
\text{End}_t
\end{align*}
\]
Context Sensitivity in Interprocedural Analysis

\[ \text{Start}_s \]
\[ a = \& b \]
\[ c_i \]
\[ C_i \]
\[ R_i \]
\[ \text{End}_s \]

\[ \text{Start}_r \]
\[ a \rightarrow b \]
\[ c \rightarrow d \]
\[ f_r \]
\[ \text{End}_r \]

\[ \text{Start}_t \]
\[ c = \& d \]
\[ c_j \]
\[ C_j \]
\[ R_j \]
\[ \text{End}_t \]
Context Sensitivity in Interprocedural Analysis

\[ a = \& b \]

\[ c_i \]

\[ C_i \]

\[ R_i \]

\[ E_{nd_s} \]

\[ \text{Start}_s \]

\[ \text{Start}_r \]

\[ f_r \]

\[ c \rightarrow d \]

\[ b \rightarrow a \]

\[ c_j \]

\[ C_j \]

\[ R_j \]

\[ E_{nd_t} \]

\[ \text{Start}_t \]

\[ c = \& d \]

\[ a \times b \]
Context Sensitivity in Interprocedural Analysis

\[ a = \& b \]

\[ c_i \]

\[ \text{Start}_s \]

\[ \text{End}_s \]

\[ \text{Start}_r \]

\[ \text{End}_r \]

\[ c \rightarrow d \]

\[ \text{Start}_t \]

\[ \text{End}_t \]

\[ c_j \]

\[ f_r \]
Context Sensitivity in Interprocedural Analysis

\[ \text{Start}_s \]
\[ a = \& b \]
\[ c_i \]
\[ C_i \]
\[ R_i \]
\[ \text{End}_s \]

\[ \text{Start}_r \]
\[ \text{End}_r \]
\[ \text{Start}_t \]
\[ c = \& d \]
\[ c_j \]
\[ C_j \]
\[ R_j \]
\[ \text{End}_t \]

\[ f_r \]

\[ a \rightarrow b \]
\[ b \rightarrow a \]
\[ c \rightarrow d \]
\[ d \rightarrow c \]
We will revisit this concept and study it in details in the fourth module (interprocedural data flow analysis) of the course.
Context Sensitivity in the Presence of Recursion

```
Start_s
\[C_i\]
\[R_i\]
End_s
```

- Call (c)
- Return (r)
- Stop calling (s)
Context Sensitivity in the Presence of Recursion

- Paths from $Start_s$ to $End_s$ should constitute a context free language $c^n s r^n$

Diagram:

- $Start_s$ to $Call (c)$ to $Ci$
- $Ci$ to $Stop Calling (s)$ to $Ri$
- $Ri$ to $Return (r)$ to $End_s$
Context Sensitivity in the Presence of Recursion

Paths from $Start_s$ to $End_s$ should constitute a context free language $c^n s r^n$

Many interprocedural analyses treat cycle of recursion as an SCC and approximate paths by a regular language $c^* s r^*$
Context Sensitivity in the Presence of Recursion

- Paths from $Start_s$ to $End_s$ should constitute a context free language $c^n s r^n$
- Many interprocedural analyses treat cycle of recursion as an SCC and approximate paths by a regular language $c^* s r^*$
- We do not know any practical points-to analysis that is fully context sensitive
  Most context sensitive approaches
Context Sensitivity in the Presence of Recursion

- Paths from $Start_s$ to $End_s$ should constitute a context free language $c^n s r^n$.
- Many interprocedural analyses treat cycle of recursion as an SCC and approximate paths by a regular language $c^* s r^*$.
- We do not know any practical points-to analysis that is fully context sensitive.
  - Most context sensitive approaches either do not consider recursion, or...
Context Sensitivity in the Presence of Recursion

- Paths from $Start_s$ to $End_s$ should constitute a context free language $c^n s r^n$
- Many interprocedural analyses treat cycle of recursion as an SCC and approximate paths by a regular language $c^* s r^*$
- We do not know any practical points-to analysis that is fully context sensitive

Most context sensitive approaches
  - either do not consider recursion, or
  - do not consider recursive pointer manipulation (e.g. “$p = p \rightarrow n$”), or
Context Sensitivity in the Presence of Recursion

- Paths from Start$_s$ to End$_s$ should constitute a context free language $c^n s r^n$
- Many interprocedural analyses treat cycle of recursion as an SCC and approximate paths by a regular language $c^* s r^*$
- We do not know any practical points-to analysis that is fully context sensitive
  - Most context sensitive approaches
    - either do not consider recursion, or
    - do not consider recursive pointer manipulation (e.g. “$p \leftarrow p \rightarrow n$”), or
    - are context insensitive in recursion
Context Sensitivity in the Presence of Recursion

- Paths from \textit{Start}_s to \textit{End}_s should constitute a context free language $c^n s r^n$.

- Many interprocedural analyses treat cycle of recursion as an SCC and approximate paths by a regular language $c^* s r^*$.

- We do not know any practical points-to analysis that is fully context sensitive.

- Most context sensitive approaches either do not consider recursion, or do not consider recursive pointer manipulation (e.g. \textquotedblleft$p = p \rightarrow n$	extquotedblright), or are context insensitive in recursion.

We will revisit this concept and study it in details in the fourth module (interprocedural data flow analysis) of the course.
Pointer Analysis: An Engineer’s Landscape

Flow Sensitivity Increases

Context Sensitivity Increases

$\text{FI} \subseteq \text{FI}_{\text{SSA}} \subseteq \text{FI}_{\text{NoKill}}$
Pointer Analysis: An Engineer’s Landscape

Data Structures: BDDs, probabilistic

Flow Sensitivity Increases

FS
FS_{NoKill}
F_{SSA}
F_{<}
F_{=}

Context Sensitivity Increases

CI  CI_{ObjSens}  CS_{Reclns}  CS
Pointer Analysis: An Engineer’s Landscape

Methods: parallel, on demand, randomized
Data Structures: BDDs, probabilistic

Flow Sensitivity Increases

FS
FS_{NoKill}
FI_{SSA}
FI_{C}
FI_{=} 

Context Sensitivity Increases

CI CI_{ObjSens} CS_{RecIns} CS
Pointer Analysis: An Engineer’s Landscape

- **Flow Sensitivity Increases**
  - $FS_{\text{NoKill}}$
  - $FS$
  - $FS_{\text{SSA}}$
  - $FS_{\subseteq}$
  - $FS_{=}$

- **Context Sensitivity Increases**
  - $Cl$
  - $Cl_{\text{ObjSens}}$
  - $CS_{\text{RecIns}}$
  - $CS$

- **Refinement:** Levelwise, bootstrapping
- **Methods:** parallel, on demand, randomized
- **Data Structures:** BDDs, probabilistic

Sep 2017
IIT Bombay
Pointer Analysis: An Engineer’s Landscape

Refinement: Levelwise, bootstrapping
Methods: parallel, on demand, randomized
Data Structures: BDDs, probabilistic

Flow Sensitivity Increases

Over Crowed Area

Context Sensitivity Increases
Pointer Analysis: An Engineer’s Landscape

- Refinement: Levelwise, bootstrapping
- Methods: parallel, on demand, randomized
- Data Structures: BDDs, probabilistic

Flow Sensitivity Increases

- $FS_{NoKill}$
- $FI_{SSA}$
- $FI_{ci}$
- $FI_{eq}$

Over Crowded Area

Context Sensitivity Increases

- $CI$
- $CI_{ObjSens}$
- $CS_{Reclns}$
- $CS$

Still Vacant
Pointer Analysis: An Engineer’s Landscape

Flow Sensitivity Increases

Over Crowed Area

That’s the corner we are trying to occupy :-)
An Outline of Pointer Analysis Coverage

- The larger perspective
- Comparing Points-to and Alias information
- Flow Insensitive Points-to Analysis
- Flow Sensitive Points-to Analysis
- Pointer Analyses: An Engineer’s Landscape
- **Liveness Based Points-to Analysis** Next Topic
- Generalizations to Heap, Arrays, Pointer Arithmetic, and Unions
Our Motivating Example for FCPA

For simplicity, we ignore the BI with "?"

\[ n_1 \]
\[
\begin{align*}
  x &= \& y \\
  y &= \& z \\
  z &= \& u
\end{align*}
\]

\[ \emptyset \]

\[ n_2 \]
\[
\begin{align*}
  *z &= y
\end{align*}
\]

\[ n_3 \]
\[
  z &= y
\]

\[ n_4 \]
\[
\begin{align*}
  y &= \& x \\
  &\text{use } u \\
  &\text{use } x
\end{align*}
\]
For simplicity, we ignore the BI with "?"
Is All This Information Useful?

For simplicity, we ignore the BI with “?”

\[ x = \& y \]
\[ y = \& z \]
\[ z = \& u \]

\[ *z = y \]
\[ z = y \]

\[ y = \& x \]
use \( u \)
use \( x \)

Sep 2017

IIT Bombay
Is All This Information Useful?

For simplicity, we ignore the $BI$ with “?”

$x = \& y$
$y = \& z$
$z = \& u$

$n_1$

$n_2$

$n_3$

$n_4$

$y = \& x$
use $u$
use $x$

$x \rightarrow y \rightarrow z \rightarrow u$

$x \rightarrow y \rightarrow z \rightarrow u$

$x \rightarrow y \rightarrow z \rightarrow u$

$x \rightarrow y \rightarrow z \rightarrow u$

For simplicity, we ignore the $BI$ with “?”
For simplicity, we ignore the BI with “?”.
Is All This Information Useful?

For simplicity, we ignore the BI with "?"
For simplicity, we ignore the BI with "?\"
The L and P of LFCPA

Mutual dependence of liveness and points-to information

- Define points-to information only for live pointers
- For pointer indirections, define liveness information using points-to information
The F and C of LFCPA

- Use call strings method for full flow and context sensitivity
- Use value contexts for efficient interprocedural analysis
Use of Strong Liveness

- Simple liveness considers every use of a variable as useful
- Strong liveness checks the liveness of the result before declaring the operands to be live
Use of Strong Liveness

- Simple liveness considers every use of a variable as useful.
- Strong liveness checks the liveness of the result before declaring the operands to be live.
- Strong liveness is more precise than simple liveness.
## Extractor Functions for LFCPA

### Unchanged from earlier points-to analysis

<table>
<thead>
<tr>
<th></th>
<th>Def&lt;sub&gt;n&lt;/sub&gt;</th>
<th>Kill&lt;sub&gt;n&lt;/sub&gt;</th>
<th>Pointee&lt;sub&gt;n&lt;/sub&gt;</th>
<th>Ref&lt;sub&gt;n&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>use x</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>Def&lt;sub&gt;n&lt;/sub&gt; ∩ Lout&lt;sub&gt;n&lt;/sub&gt; ≠ ∅</td>
</tr>
<tr>
<td>x = &amp;a</td>
<td>{x}</td>
<td>{x}</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>x = y</td>
<td>{x}</td>
<td>{x}</td>
<td>{a}</td>
<td></td>
</tr>
<tr>
<td>x = *y</td>
<td>{x}</td>
<td>{x}</td>
<td>A{y}</td>
<td></td>
</tr>
<tr>
<td>*x = y</td>
<td>A{x} ∩ P</td>
<td>Must(A){x} ∩ P</td>
<td>A(y)</td>
<td></td>
</tr>
<tr>
<td>other</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td></td>
</tr>
</tbody>
</table>

### Generation of strong liveness

- Lin/Lout: set of Live pointers, Ain/Aout: sets of mAy points-to pairs
- Ref<sub>n</sub>, Kill<sub>n</sub>, Def<sub>n</sub>, and Pointee<sub>n</sub> are defined in terms of Ain<sub>n</sub>
Extractor Functions for LFCPA

Unchanged from earlier points-to analysis

<table>
<thead>
<tr>
<th>Operation</th>
<th>Def&lt;sub&gt;n&lt;/sub&gt;</th>
<th>Kill&lt;sub&gt;n&lt;/sub&gt;</th>
<th>Pointee&lt;sub&gt;n&lt;/sub&gt;</th>
<th>Ref&lt;sub&gt;n&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>use x</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>x = &amp;a</td>
<td>{x}</td>
<td>{x}</td>
<td>{a}</td>
<td></td>
</tr>
<tr>
<td>x = y</td>
<td>{x}</td>
<td>{x}</td>
<td>A{y}</td>
<td></td>
</tr>
<tr>
<td>x = *y</td>
<td>{x}</td>
<td>{x}</td>
<td>A(A{y} ∩ P)</td>
<td></td>
</tr>
<tr>
<td>*x = y</td>
<td>A{x} ∩ P</td>
<td>Must(A){x} ∩ P</td>
<td>A{y}</td>
<td></td>
</tr>
<tr>
<td>other</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td></td>
</tr>
</tbody>
</table>

Generation of strong liveness

Pointers that become live
#### Extractor Functions for LFCPA

<table>
<thead>
<tr>
<th></th>
<th>$Def_n$</th>
<th>$Kill_n$</th>
<th>$Pointee_n$</th>
<th>$Ref_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>use $x$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td></td>
</tr>
<tr>
<td>$x = &amp;a$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>${a}$</td>
<td></td>
</tr>
<tr>
<td>$x = y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A{y}$</td>
<td></td>
</tr>
<tr>
<td>$x = *y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A(A{y} \cap P)$</td>
<td></td>
</tr>
<tr>
<td>$*x = y$</td>
<td>$A{x} \cap P$</td>
<td>$Must(A){x} \cap P$</td>
<td>$A{y}$</td>
<td></td>
</tr>
<tr>
<td>other</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td></td>
</tr>
</tbody>
</table>

- **Unchanged from earlier points-to analysis**
- **Generation of strong liveness**

Defined pointers must be live at the exit for the read pointers to become live.

*Must: the set of points-to for which there is no read before the points-to.*
### Extractor Functions for LFCPA

<table>
<thead>
<tr>
<th></th>
<th>Def&lt;sub&gt;n&lt;/sub&gt;</th>
<th>Kill&lt;sub&gt;n&lt;/sub&gt;</th>
<th>Pointee&lt;sub&gt;n&lt;/sub&gt;</th>
<th>Ref&lt;sub&gt;n&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>use x</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>x = &amp;a</td>
<td>{x}</td>
<td>{x}</td>
<td>{a}</td>
<td>(\text{otherwise})</td>
</tr>
<tr>
<td>x = y</td>
<td>{x}</td>
<td>{x}</td>
<td>(A{y})</td>
<td></td>
</tr>
<tr>
<td>x = *y</td>
<td>{x}</td>
<td>{x}</td>
<td>(A(A{y} \cap P))</td>
<td></td>
</tr>
<tr>
<td>*x = y</td>
<td>(A{x} \cap P)</td>
<td>(\text{Must}(A){x} \cap P)</td>
<td>(A{y})</td>
<td></td>
</tr>
<tr>
<td>other</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td></td>
</tr>
</tbody>
</table>

#### Unchanged from earlier points-to analysis

#### Generation of strong liveness

- Some pointers are unconditionally live

---

Sep 2017

IIT Bombay
## Extractor Functions for LFCPA

### Table: Extractor Functions for LFCPA

<table>
<thead>
<tr>
<th></th>
<th>$\text{Def}_n$</th>
<th>$\text{Kill}_n$</th>
<th>$\text{Pointee}_n$</th>
<th>$\text{Ref}_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{use } x$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>${x}$</td>
</tr>
<tr>
<td>$x = &amp;a$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>${a}$</td>
<td>${x}$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A{y}$</td>
<td></td>
</tr>
<tr>
<td>$x = *y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A(A{y} \cap P)$</td>
<td></td>
</tr>
<tr>
<td>$x = y$</td>
<td>$A{x} \cap P$</td>
<td>$\text{Must}(A){x} \cap P$</td>
<td>$A{y}$</td>
<td></td>
</tr>
<tr>
<td>other</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td></td>
</tr>
</tbody>
</table>

**Unchanged from earlier points-to analysis**

**Generation of strong liveness**

- $x$ is unconditionally live
## Extractor Functions for LFCPA

### Unchanged from earlier points-to analysis

<table>
<thead>
<tr>
<th>Operation</th>
<th>(\text{Def}_n)</th>
<th>(\text{Kill}_n)</th>
<th>(\text{Pointee}_n)</th>
<th>(\text{Ref}_n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>use (x)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>({x})</td>
</tr>
<tr>
<td>(x = &amp; a)</td>
<td>({x})</td>
<td>({x})</td>
<td>({a})</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>(x = y)</td>
<td>({x})</td>
<td>({x})</td>
<td>(A{y})</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>(x = \ast y)</td>
<td>({x})</td>
<td>({x})</td>
<td>(A(A{y} \cap P))</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>(\ast x = y)</td>
<td>(A{x} \cap P)</td>
<td>(\text{Must}(A){x} \cap P)</td>
<td>(A{y})</td>
<td>({x})</td>
</tr>
<tr>
<td>other</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
</tr>
</tbody>
</table>

### Generation of strong liveness

- \(\text{Def}_n \cap \text{Lout}_n \neq \emptyset\)
- otherwise
## Extractor Functions for LFCPA

### Unchanged from earlier points-to analysis

<table>
<thead>
<tr>
<th></th>
<th>$Def_n$</th>
<th>$Kill_n$</th>
<th>$Pointee_n$</th>
<th>$Ref_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$use\ x$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>${x}$</td>
</tr>
<tr>
<td>$x = &amp; a$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>${a}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A{y}$</td>
<td>${y}$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A(A{y} \cap P)$</td>
<td></td>
</tr>
<tr>
<td>$*x = y$</td>
<td>$A{x} \cap P$</td>
<td>$Must(A){x} \cap P$</td>
<td>$A{y}$</td>
<td></td>
</tr>
<tr>
<td>other</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td></td>
</tr>
</tbody>
</table>

### Generation of strong liveness

- $Def_n \cap Lout_n \neq \emptyset$
- otherwise

- $y$ is live if defined pointers are live

- Sep 2017 IIT Bombay
### Extractor Functions for LFCPA

<table>
<thead>
<tr>
<th></th>
<th>Def&lt;sub&gt;n&lt;/sub&gt;</th>
<th>Kill&lt;sub&gt;n&lt;/sub&gt;</th>
<th>Pointee&lt;sub&gt;n&lt;/sub&gt;</th>
<th>Ref&lt;sub&gt;n&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>use x</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>{x}</td>
</tr>
<tr>
<td>x = &amp;a</td>
<td>{x}</td>
<td>{x}</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>x = y</td>
<td>{x}</td>
<td>{x}</td>
<td>A{x}</td>
<td>∅</td>
</tr>
<tr>
<td>x = *y</td>
<td>{x}</td>
<td>{x}</td>
<td>A(A{x} ∩ P)</td>
<td>∅</td>
</tr>
<tr>
<td>*x = y</td>
<td>A{x} ∩ P</td>
<td>Must(A){x} ∩ P</td>
<td>A{y}</td>
<td></td>
</tr>
<tr>
<td>other</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td></td>
</tr>
</tbody>
</table>

- **Unchanged from earlier points-to analysis**
- **Generation of strong liveness**
Extractor Functions for LFCPA

<table>
<thead>
<tr>
<th></th>
<th>Def$_n$</th>
<th>Kill$_n$</th>
<th>Pointee$_n$</th>
<th>Ref$_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>use $x$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>${x}$</td>
</tr>
<tr>
<td>$x = &amp;a$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>${a}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A{y}$</td>
<td>${y}$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A(A{y} \cap P)$</td>
<td>${y} \cup A{y} \cap P$</td>
</tr>
<tr>
<td>$*x = y$</td>
<td>$A{x} \cap P$</td>
<td>$Must(A){x} \cap P$</td>
<td>$A{y}$</td>
<td></td>
</tr>
<tr>
<td>other</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td></td>
</tr>
</tbody>
</table>

- **Unchanged from earlier points-to analysis**
- **Generation of strong liveness**

- $y$ and its pointees in $Ain_n$ are live if defined pointers are live.
### Extractor Functions for LFCPA

#### Unchanged from earlier points-to analysis

<table>
<thead>
<tr>
<th></th>
<th>Def(_n)</th>
<th>Kill(_n)</th>
<th>Pointee(_n)</th>
<th>Ref(_n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>use (x)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>({x})</td>
</tr>
<tr>
<td>(x = &amp;a)</td>
<td>({x})</td>
<td>({x})</td>
<td>({a})</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>(x = y)</td>
<td>({x})</td>
<td>({x})</td>
<td>(A{y})</td>
<td>({y})</td>
</tr>
<tr>
<td>(x = *y)</td>
<td>({x})</td>
<td>({x})</td>
<td>(A(A{y} \cap P))</td>
<td>({y} \cup A{y} \cap P)</td>
</tr>
<tr>
<td>(*x = y)</td>
<td>(A{x} \cap P)</td>
<td>Must((A){x} \cap P)</td>
<td>(A{y})</td>
<td></td>
</tr>
<tr>
<td>other</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td></td>
</tr>
</tbody>
</table>

## Generation of strong liveness

```markdown
must (Def\(_n\) \cap Lout\(_n\) \neq \emptyset) otherwise
```
### Extractor Functions for LFCPA

<table>
<thead>
<tr>
<th></th>
<th>Def$_n$</th>
<th>Kill$_n$</th>
<th>Pointee$_n$</th>
<th>Ref$_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>use $x$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>${x}$</td>
</tr>
<tr>
<td>$x = &amp;a$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>${a}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A{y}$</td>
<td>${y}$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A(A{y} \cap P)$</td>
<td>${y} \cup A{y} \cap P$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>$A{x} \cap P$</td>
<td>$Must(A){x} \cap P$</td>
<td>$A{y}$</td>
<td>${x, y}$</td>
</tr>
<tr>
<td>other</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

- **Unchanged from earlier points-to analysis**
- **Generation of strong liveness**

- $y$ is live if defined pointers are live.
### Extractor Functions for LFCPA

<table>
<thead>
<tr>
<th></th>
<th>$Def_n$</th>
<th>$Kill_n$</th>
<th>$Pointee_n$</th>
<th>$Ref_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>use x</strong></td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>${x}$, ${x}$</td>
</tr>
<tr>
<td>$x = &amp;a$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>${a}$</td>
<td>$\emptyset$, $\emptyset$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A{y}$</td>
<td>${y}$, $\emptyset$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A(A{y} \cap P)$</td>
<td>${y} \cup A{y} \cap P$, $\emptyset$</td>
</tr>
<tr>
<td>$*x = y$</td>
<td>$A{x} \cap P$</td>
<td>$Must(A){x} \cap P$</td>
<td>$A{y}$</td>
<td>${x, y}$, ${x}$</td>
</tr>
<tr>
<td><strong>other</strong></td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$, $\emptyset$</td>
</tr>
</tbody>
</table>

- **Unchanged from earlier points-to analysis**
- **Generation of strong liveness**

$x$ is unconditionally live
## Extractor Functions for LFCPA

### Unchanged from earlier points-to analysis

<table>
<thead>
<tr>
<th>Operation</th>
<th>$Def_n$</th>
<th>$Kill_n$</th>
<th>$Pointee_n$</th>
<th>$Ref_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$use \ x$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>${x}$</td>
</tr>
<tr>
<td>$x = &amp; a$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>${a}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>${y}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$A({y} \cap P)$</td>
<td>${y} \cup A{y} \cap P$</td>
</tr>
<tr>
<td>$*x = y$</td>
<td>$A{x} \cap P$</td>
<td>$Must(A){x} \cap P$</td>
<td>$A{y}$</td>
<td>${x, y}$</td>
</tr>
<tr>
<td>other</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

### Generation of strong liveness

$Def_n \cap Lout_n \neq \emptyset$ otherwise
Deriving *Must* Points-to for LFCPA

For $*x = y$, unless the pointees of $x$ are known

- points-to propagation should be blocked
- liveness propagation should be blocked

to ensure monotonicity

$$\text{Must}(R) = \bigcup_{x \in P} \{x\} \times \begin{cases} \forall \text{Var} & R\{x\} = \emptyset \lor R\{x\} = \{?\} \\ \{y\} & R\{x\} = \{y\} \land y \neq ? \\ \emptyset & \text{otherwise} \end{cases}$$
**LFCPA Data Flow Equations**

\[
L_{out,n} = \begin{cases}
\emptyset & \text{if } n \text{ is } End_p \\
\bigcup_{s \in \text{succ}(n)} L_{in,s} & \text{otherwise}
\end{cases}
\]

\[
L_{in,n} = \left( L_{out,n} - K_{ill,n} \right) \cup R_{ef,n}
\]

\[
A_{in,n} = \begin{cases}
L_{in,n} \times \{?\} & \text{if } n \text{ is } Start_p \\
\left( \bigcup_{p \in \text{pred}(n)} A_{out,p} \right) \biggr| L_{in,n} & \text{otherwise}
\end{cases}
\]

\[
A_{out,n} = \left( \left( A_{in,n} - \left( K_{ill,n} \times \text{Var} \right) \right) \cup \left( D_{ef,n} \times P_{ointee,n} \right) \right) \biggr| L_{out,n}
\]

- \textbf{Lin}/\textbf{Lout}: set of Live pointers
- \textbf{Ain}/\textbf{Aout}: definitions remain unchanged except for restriction to liveness
**LFCPA Data Flow Equations**

\[
L_{out_n} = \begin{cases} 
\emptyset & \text{if } n \text{ is } End_p \\
\bigcup_{s \in \text{succ}(n)} L_{in_s} & \text{otherwise}
\end{cases}
\]

\[
L_{in_n} = (L_{out_n} - \text{Kill}_n) \cup \text{Ref}_n
\]

\[
A_{in_n} = \begin{cases} 
L_{in_n} \times \{?\} & \text{if } n \text{ is } \text{Start}_p \\
\left( \bigcup_{p \in \text{pred}(n)} A_{out_p} \right) \bigcap L_{in_n} & \text{otherwise}
\end{cases}
\]

\[
A_{out_n} = \left( (A_{in_n} - (\text{Kill}_n \times \text{Var})) \cup (\text{Def}_n \times \text{Pointee}_n) \right) \bigcap L_{out_n}
\]

- \(L_{in}/L_{out}\): set of Live pointers
- \(A_{in}/A_{out}\): definitions remain unchanged except for restriction to liveness

*Kill_n defined in terms of A_{in_n}
**LFCPA Data Flow Equations**

\[
L_{\text{out}}(n) = \begin{cases} 
\emptyset & \text{if } n \text{ is } \text{End}_p \\
\bigcup_{s \in \text{succ}(n)} L_{\text{in}}(s) & \text{otherwise}
\end{cases}
\]

\[
L_{\text{in}}(n) = (L_{\text{out}}(n) - \text{Kill}_n) \cup \text{Ref}_n
\]

\[
A_{\text{in}}(n) = \begin{cases} 
L_{\text{in}}(n) \times \{?\} & \text{if } n \text{ is } \text{Start}_p \\
\left( \bigcup_{p \in \text{pred}(n)} A_{\text{out}}(p) \right) & \text{otherwise}
\end{cases}
\]

\[
A_{\text{out}}(n) = \left( \left( A_{\text{in}}(n) - \left( \text{Kill}_n \times \text{Var} \right) \right) \cup \left( \text{Def}_n \times \text{Pointee}_n \right) \right) \bigg| L_{\text{out}}(n)
\]

- *Lin/Lout*: set of Live pointers
- *Ain/Aout*: definitions remain unchanged except for restriction to liveness
LFCPA Data Flow Equations

\[
Lout_n = \begin{cases} 
\emptyset & \text{if } n \text{ is } \text{End}_p \\
\bigcup_{s \in \text{succ}(n)} Lin_s & \text{otherwise}
\end{cases}
\]

\[
Lin_n = (Lout_n - \text{Kill}_n) \cup \text{Ref}_n
\]

\[
Ain_n = \begin{cases} 
Lin_n \times \{?\} & \text{if } n \text{ is } \text{Start}_p \\
\left(\bigcup_{p \in \text{pred}(n)} Aout_p\right) \big| Lin_n & \text{otherwise}
\end{cases}
\]

\[
Aout_n = \left(\left(Ain_n - (\text{Kill}_n \times \text{Var})\right) \cup (\text{Def}_n \times \text{Pointee}_n)\right) \big| Lout_n
\]

- **Lin/Lout**: set of Live pointers
- **Ain/Aout**: definitions remain unchanged except for restriction to liveness
LFCPA Data Flow Equations

\[ L_{out}^{n} = \begin{cases} \emptyset & \text{if } n \text{ is } \text{End}_{p} \\ \bigcup_{s \in \text{succ}(n)} L_{in}^{s} & \text{otherwise} \end{cases} \]

\[ L_{in}^{n} = (L_{out}^{n} - K_{ill}^{n}) \cup R_{ef}^{n} \]

\[ A_{in}^{n} = \begin{cases} L_{in}^{n} \times \{?\} & \text{if } n \text{ is } \text{Start}_{p} \\ \left( \bigcup_{p \in \text{pred}(n)} A_{out}^{p} \right) \bigg| L_{in}^{n} & \text{otherwise} \end{cases} \]

\[ A_{out}^{n} = \left( \left( A_{in}^{n} - (K_{ill}^{n} \times \text{Var}) \right) \cup \left( \text{Def}^{n} \times \text{Pointee}^{n} \right) \right) \bigg| L_{out}^{n} \]

- \( L_{in}/L_{out} \): set of Live pointers
- \( A_{in}/A_{out} \): definitions remain unchanged except for restriction to liveness

\( BI \) restricted to live pointers
**LFCPA Data Flow Equations**

\[
L_{\text{out}}_n = \begin{cases} 
\emptyset & \text{if } n \text{ is } \text{End}_p \\
\bigcup_{s \in \text{succ}(n)} L_{\text{in}}_s & \text{otherwise}
\end{cases}
\]

\[
L_{\text{in}}_n = (L_{\text{out}}_n - K_{\text{ill}}_n) \cup R_{\text{ef}}_n
\]

\[
A_{\text{in}}_n = \begin{cases} 
L_{\text{in}}_n \times \{?\} & \text{if } n \text{ is } \text{Start}_p \\
\left( \bigcup_{p \in \text{pred}(n)} A_{\text{out}}_p \right) \big| L_{\text{in}}_n & \text{otherwise}
\end{cases}
\]

\[
A_{\text{out}}_n = \left( (A_{\text{in}}_n - (K_{\text{ill}}_n \times V_{\text{ar}})) \cup (D_{\text{ef}}_n \times P_{\text{ointee}}_n) \right) \big| L_{\text{out}}_n
\]

- *Lin/Lout*: set of Live pointers
- *Ain/Aout*: definitions remain unchanged except for restriction to liveness
Motivating Example Revisited

- For convenience, we show complete sweeps of liveness and points-to analysis repeatedly
- This is not required by the computation
- The data flow equations define a single fixed point computation
First Round of Liveness Analysis and Points-to Analysis

\[
\begin{align*}
  x &= \&y \\
  y &= \&z \\
  z &= \&u
\end{align*}
\]

\[
\begin{align*}
  *z &= y & n_2 \\
  z &= y & n_3
\end{align*}
\]

\[
\begin{align*}
  y &= \&x \\
  \text{use } u \\
  \text{use } x & n_4
\end{align*}
\]
First Round of Liveness Analysis and Points-to Analysis

\[
\begin{align*}
x &= \&y \\
y &= \&z \\
z &= \&u \\
\end{align*}
\]

\[
\begin{align*}
*z &= y \\
z &= y \\
y &= \&x \\
use\ u \\
use\ x \\
\end{align*}
\]
First Round of Liveness Analysis and Points-to Analysis

\[ \begin{align*}
    x &= \& y \\
y &= \& z \\
z &= \& u
\end{align*} \]

\[ \begin{align*}
    z &= y & n_3 \\
y &= \& x & \{u, x\} \\
    \text{use } u \\
    \text{use } x
\end{align*} \]

Liveness Analysis
First Round of Liveness Analysis and Points-to Analysis

\[
x = \& y \\
y = \& z \\
z = \& u
\]

\[
* z = y \\
z = y \\
y = \& x
\]

\[
\{ u, x \} \\
\{ u, x \} \\
\{ u, x \}
\]
First Round of Liveness Analysis and Points-to Analysis

\[ x = \&y \]
\[ y = \&z \]
\[ z = \&u \]

Liveness Analysis

\{z\} \rightarrow \{u, x\} \rightarrow \{u, x\} \rightarrow \{u, x\} \rightarrow \{u, x\}

\[ \text{use } u \]
\[ \text{use } x \]
First Round of Liveness Analysis and Points-to Analysis

\[ x = \& y \]
\[ y = \& z \]
\[ z = \& u \]

Liveness Analysis

\[ \{u, x\} \]

\[ \ast z = y \]
\[ z = y \]

\[ \{u, x\} \]

\[ y = \& x \]
\[ use u \]
\[ use x \]

\[ \{u, x\} \]

Strong liveness: y is not made live because z is not live
First Round of Liveness Analysis and Points-to Analysis

$x = \& y$
$y = \& z$
$z = \& u$

$n_1$

$\{u, x, z\}$

$\{z\}$

$*z = y$

$n_2$

$\{u, x\}$

$z = y$

$n_3$

$\{u, x\}$

$y = \& x$

$use u$

$use x$

$n_4$
First Round of Liveness Analysis and Points-to Analysis

\[
\begin{align*}
  x &= &\& y \\
  y &= &\& z \\
  z &= &\& u \\
  n_1
\end{align*}
\]

\[
\begin{align*}
  *z &= y \\
  z &= y \\
  z &= \& u \\
  n_2
\end{align*}
\]

\[
\begin{align*}
  y &= &\& x \\
  use u \\
  use x \\
  n_3
\end{align*}
\]

\[
\begin{align*}
  n_4
\end{align*}
\]
First Round of Liveness Analysis and Points-to Analysis

\[
\begin{align*}
  x &= \& y \\
  y &= \& z \\
  z &= \& u
\end{align*}
\]

\[
\begin{align*}
  &\{u\} \\
  &n_1 \\

  &\{u, x, z\} \\

  z &= y \\
  &n_3 \\

  &\{u, x\} \\

  *z &= y \\
  &n_2 \\

  &\{u, x\} \\

  y &= \& x \\
  use u \\
  use x
\end{align*}
\]
First Round of Liveness Analysis and Points-to Analysis

\[ x = \& y \]
\[ y = \& z \]
\[ z = \& u \]

\[
\begin{align*}
&\text{Points-to Analysis} \\
&\text{First Round of Liveness Analysis and Points-to Analysis}
\end{align*}
\]
First Round of Liveness Analysis and Points-to Analysis

\( x = \& y \)
\( y = \& z \)
\( z = \& u \)

\( n_1 \)
\( \{ u \} \)
\( ? \)

\( \{ u, x, z \} \)
\( \{ u, x \} \)
\( x \rightarrow y \)
\( z \rightarrow u \)
\( ? \)

\( n_2 \)
\( \{ z \} \)
\( \{ u, x \} \)

\( \{ u, x \} \)
\( \{ u, x \} \)

\( \{ u, x \} \)
\( \{ u, x \} \)

\( \{ u, x \} \)
\( use u \)
\( use x \)

\( n_4 \)
First Round of Liveness Analysis and Points-to Analysis

\[
x = &y \\
y = &z \\
z = &u
\]

\[
\{z\} \quad x = &y \\
\{u, x\} \quad z = y \\
\{u, x\} \quad y = &x
\]

\[
\{u\} \quad u \rightarrow ? \\
\{u, x, z\} \quad x \rightarrow y \quad z \rightarrow u \rightarrow ? \\
\{u, x\} \quad x \rightarrow y \quad u \rightarrow ? \\
\{u, x\} \quad y \rightarrow &x
\]

Points-to Analysis

Points-to Analysis
First Round of Liveness Analysis and Points-to Analysis

\[ x = \&y \]
\[ y = \&z \]
\[ z = \&u \]

\[ \{u\} \quad u \rightarrow ? \]

\[ n_1 \]

\[ \{u, x, z\} \quad x \rightarrow y \quad z \rightarrow u \rightarrow ? \]

\[ \{z\} \]

\[ \{u, x\} \quad x \rightarrow y \]

\[ *z = y \quad n_2 \]

\[ \{u, x\} \quad x \rightarrow y \]

\[ z = y \quad n_3 \]

\[ \{u, x\} \quad x \rightarrow y \]

\[ \{u, x, z\} \quad x \rightarrow y \quad u \rightarrow ? \]

\[ \{u, x\} \quad x \rightarrow y \]

\[ \{u, x\} \quad x \rightarrow y \]

\[ \{u, x\} \quad x \rightarrow y \]

\[ \{u, x\} \quad x \rightarrow y \]

\[ \{u, x\} \quad x \rightarrow y \]

\[ \{u, x\} \quad x \rightarrow y \]

\[ y = \&x \]

\[ use u \]

\[ use x \]

\[ n_4 \]
First Round of Liveness Analysis and Points-to Analysis

1. \( x = \& y \)
2. \( y = \& z \)
3. \( z = \& u \)

Points-to Analysis:
- \( \{u\} \)
  - \( u \rightarrow ? \)
- \( \{u, x, z\} \)
  - \( x \rightarrow y \)
  - \( z \rightarrow u \rightarrow ? \)
- \( \{u, x\} \)
  - \( x \rightarrow y \)
  - \( u \rightarrow ? \)
- \( \{u, x\} \)
  - \( x \rightarrow y \)
  - \( u \rightarrow ? \)
- \( \{u, x\} \)
  - \( u \rightarrow ? \)

Use:
- \( use u \)
- \( use x \)
First Round of Liveness Analysis and Points-to Analysis

\[
x = &y \\
y = &z \\
z = &u
\]

Points-to Analysis

\[\text{z } \rightarrow \text{u} \quad \{z\}\]

\[\text{\*z } \rightarrow \text{y} \quad n_2\]

\[\text{z } \rightarrow \text{y} \quad n_3\]

\[\text{y } \rightarrow \text{&x} \quad \text{use u} \quad \text{use x} \quad n_4\]

Points-to Analysis

\[\{u\} \quad \text{u } \rightarrow \text{?}\]

\[\{u, x, z\} \quad \text{x } \rightarrow \text{y} \quad \text{z } \rightarrow \text{u } \rightarrow \text{?}\]

\[\{u, x\} \quad \text{u } \rightarrow \text{?} \quad \text{use x}\]

\[\{u, x\} \quad \text{z } \rightarrow \text{y} \quad \text{u } \rightarrow \text{?}\]
First Round of Liveness Analysis and Points-to Analysis

\[ x = \& y \]
\[ y = \& z \]
\[ z = \& u \]

\[ z \to u \]
\[ \{z\} \]

\[ *z = y \]
\[ \{u, x\} \to y \]
\[ z \to u \to ? \]

\[ y = \& x \]
\[ use u \]
\[ use x \]

\[ \{u, x\} \to y \]
\[ u \to ? \]

\[ \{u\} \to u \to ? \]
Second Round of Liveness Analysis and Points-to Analysis

\[
x = \&y \\
y = \&z \\
z = \&u
\]

\[n_1\]

\[
\{u, x, z\} \quad x \rightarrow y \\
z \rightarrow u \\
\{u, x\}
\]

\[n_2\]

\[
\{u\} \quad u \rightarrow ? \\
z = y \\
\{u, x\}
\]

\[n_3\]

\[
\{u, x\} \quad x \rightarrow y \\
u \rightarrow ? \\
\{u, x\}
\]

\[n_4\]

\[
y = \&x \\
use u \\
use x \quad \{u, x\}
\]

\[
\{u, x\} \quad x \rightarrow y \\
u \rightarrow ? \\
\{u, x\}
\]
Second Round of Liveness Analysis and Points-to Analysis

x = &y
y = &z
z = &u

\{u\} u → ?
\{u, x, z\} x → y  z → u → ?
\{u, x\} x → y  u → ?
\{u, x\} x → y  u → ?
\{u, x\} x → y  u → ?

z → u
\{x, y, z\}
Second Round of Liveness Analysis and Points-to Analysis

\[
x = &y \\
y = &z \\
z = &u
\]

Liveness Analysis

\[
\begin{align*}
&z = &u &\{x, y, z\} \\
x = &y &\{u, x\} \\
*z = y &\quad n_2 \\
z = y &\quad n_3 \\
y = &x &\{u, x\} \\
use u &\quad n_4
\end{align*}
\]
Second Round of Liveness Analysis and Points-to Analysis

\[
x = &y \\
y = &z \\
z = &u \\
\]

Points-to Analysis

\[z \rightarrow u \quad \{x, y, z\} \]

\[*z = y \quad n_2 \]

\[\{u, x\} \]

\[y = &x \\
use u \\
use x \quad n_4 \]

Points-to Analysis

\[\{u, x, y, z\} \rightarrow y \rightarrow z \rightarrow u \rightarrow ? \]

\[\{u, x\} \rightarrow y \rightarrow u \rightarrow ? \]

\[\{u, x\} \rightarrow y \rightarrow u \rightarrow ? \]

\[\{u, x\} \rightarrow y \rightarrow u \rightarrow ? \]
Second Round of Liveness Analysis and Points-to Analysis

\[ x = &y \]
\[ y = &z \]
\[ z = &u \]

\[ *z = y \]
\[ y = &x \]

Points-to Analysis

- Points-to Analysis
  - \( \{u\} \)
  - \( x \rightarrow y \rightarrow z \rightarrow u \rightarrow ? \)
  - \( n_1 \)
  - \( \{u, x, y, z\} \)

- Points-to Analysis
  - \( \{u, x\} \)
  - \( x \rightarrow y \rightarrow u \rightarrow ? \)
  - \( n_2 \)

- Points-to Analysis
  - \( \{u, x\} \)
  - \( x \rightarrow y \rightarrow u \rightarrow ? \)
  - \( n_3 \)

- Points-to Analysis
  - \( \{u, x\} \)
  - \( x \rightarrow y \rightarrow u \rightarrow ? \)
  - \( n_4 \)

- Points-to Analysis
  - \( \{u, x\} \)
  - \( x \rightarrow y \rightarrow u \rightarrow ? \)
Second Round of Liveness Analysis and Points-to Analysis

\[
\begin{align*}
  x &= \& y \\
  y &= \& z \\
  z &= \& u \\
  \text{use } u \\
  \text{use } x
\end{align*}
\]
Second Round of Liveness Analysis and Points-to Analysis

\( x = \& y \)
\( y = \& z \)
\( z = \& u \)

Points-to Analysis

\( *z = y \)

Points-to Analysis

\( y = \& x \)
use \( u \)
use \( x \)

Points-to Analysis

\( x \rightarrow y \rightarrow z \rightarrow u \rightarrow ? \)

Points-to Analysis

\( n_1 \)
\( n_2 \)
\( n_3 \)
\( n_4 \)
LFCPA Implementation

- LTO framework of GCC 4.6.0
- Naive prototype implementation
  (Points-to sets implemented using linked lists)
- Implemented FCPA without liveness for comparison
- Comparison with GCC's flow and context insensitive method
- SPEC 2006 benchmarks
## Analysis Time

<table>
<thead>
<tr>
<th>Program</th>
<th>kLoC</th>
<th>Call Sites</th>
<th>Liveness</th>
<th>Points-to</th>
<th>Time in milliseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>L-FCPA</td>
<td></td>
<td>FCPA</td>
</tr>
<tr>
<td><strong>Liveness</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Points-to</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lbm</td>
<td>0.9</td>
<td>33</td>
<td>0.55</td>
<td>0.52</td>
<td>1.9</td>
</tr>
<tr>
<td>mcf</td>
<td>1.6</td>
<td>29</td>
<td>1.04</td>
<td>0.62</td>
<td>9.5</td>
</tr>
<tr>
<td>libquantum</td>
<td>2.6</td>
<td>258</td>
<td>2.0</td>
<td>1.8</td>
<td>5.6</td>
</tr>
<tr>
<td>bzip2</td>
<td>3.7</td>
<td>233</td>
<td>4.5</td>
<td>4.8</td>
<td>28.1</td>
</tr>
<tr>
<td>parser</td>
<td>7.7</td>
<td>1123</td>
<td>$1.2 \times 10^3$</td>
<td>145.6</td>
<td>$4.3 \times 10^5$</td>
</tr>
<tr>
<td>sjeng</td>
<td>10.5</td>
<td>678</td>
<td>858.2</td>
<td>99.0</td>
<td>$3.2 \times 10^4$</td>
</tr>
<tr>
<td>hmmer</td>
<td>20.6</td>
<td>1292</td>
<td>90.0</td>
<td>62.9</td>
<td>$2.9 \times 10^5$</td>
</tr>
<tr>
<td>h264ref</td>
<td>36.0</td>
<td>1992</td>
<td>$2.2 \times 10^5$</td>
<td>$2.0 \times 10^5$</td>
<td>?</td>
</tr>
</tbody>
</table>
# Unique Points-to Pairs

<table>
<thead>
<tr>
<th>Program</th>
<th>kLoC</th>
<th>Call Sites</th>
<th>Unique points-to pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>L-FCPA</td>
</tr>
<tr>
<td>lbm</td>
<td>0.9</td>
<td>33</td>
<td>12</td>
</tr>
<tr>
<td>mcf</td>
<td>1.6</td>
<td>29</td>
<td>41</td>
</tr>
<tr>
<td>libquantum</td>
<td>2.6</td>
<td>258</td>
<td>49</td>
</tr>
<tr>
<td>bzip2</td>
<td>3.7</td>
<td>233</td>
<td>60</td>
</tr>
<tr>
<td>parser</td>
<td>7.7</td>
<td>1123</td>
<td>531</td>
</tr>
<tr>
<td>sjeng</td>
<td>10.5</td>
<td>678</td>
<td>267</td>
</tr>
<tr>
<td>hmmer</td>
<td>20.6</td>
<td>1292</td>
<td>232</td>
</tr>
<tr>
<td>h264ref</td>
<td>36.0</td>
<td>1992</td>
<td>1683</td>
</tr>
</tbody>
</table>
Points-to Information is Small and Sparse

% of Points-to Pairs per Basic Block

- **libm**
- **mcf**
- **libquantum**
- **bzip2**
- **sjeng**
- **hmmner**
- **parser**
- **h264ref**

**LFCPA**
- 0
- 1-4
- 5-8
- 9+

**FCPA**
- 0
- 1-4
- 5-8
- 9+
LFCPA Observations

- Usable pointer information is very small and sparse
- Data flow propagation in real programs seems to involve only a small subset of all possible data flow values
- Earlier approaches reported inefficiency and non-scalability because they computed far more information than the actual usable information
LFCPA Conclusions

- Building quick approximations and compromising on precision may not be necessary for efficiency.
- Building clean abstractions to separate the necessary information from redundant information is much more significant.
LFCPA Conclusions

- Building quick approximations and compromising on precision may not be necessary for efficiency.
- Building clean abstractions to separate the necessary information from redundant information is much more significant.

Our experience of points-to analysis shows that:

- Use of liveness reduced the pointer information . . .
- which reduced the number of contexts required . . .
- which reduced the liveness and pointer information . . .
LFCPA Conclusions

- Building quick approximations and compromising on precision may not be necessary for efficiency.

- Building clean abstractions to separate the necessary information from redundant information is much more significant.

Our experience of points-to analysis shows that:

- Use of liveness reduced the pointer information...
- which reduced the number of contexts required...
- which reduced the liveness and pointer information...

- Approximations should come after building abstractions rather than before.
LFCPA Lessons: The Larger Perspective

- exhaustive computation
- computation restricted to usable information
- incremental computation
- demand driven computation
LFCPA Lessons: The Larger Perspective

- Exhaustive computation
- Computation restricted to usable information
- Incremental computation
- Demand driven computation

Maximum Computation

Minimum Computation
LFCPA Lessons: The Larger Perspective

- Exhaustive computation
- Computation restricted to usable information
- Incremental computation
- Demand-driven computation

Maximum Computation
Early Computation

Minimum Computation
Late Computation

Sep 2017 IIT Bombay
LF CPA Lessons: The Larger Perspective

- exhaustive computation
- computation restricted to usable information
- incremental computation
- demand driven computation

What should be computed?

Maximum Computation

When should it be computed?

Early Computation

Minimum Computation

Late Computation

Sep 2017
**LF CPA Lessons: The Larger Perspective**

- **exhaustive computation**
- **computation restricted to usable information**
- **incremental computation**
- **demand driven computation**

What should be computed?

Maximum Computation

Minimum Computation

When should it be computed?

Early Computation

Late Computation

Do not compute what you don’t need!

Who defines what is needed?
**LFCPA Lessons: The Larger Perspective**

- **exhaustive computation**
- **computation restricted to usable information**
- **incremental computation**
- **demand driven computation**

---

- **Maximum Computation**
  - What should be computed?
  - Early Computation

- **Minimum Computation**
  - Late Computation

---

- **When should it be computed?**

---

- **Do not compute what you don’t need!**
- **Who defines what is needed?** Client
LFCPA Lessons: The Larger Perspective

- **exhaustive computation**
- **computation restricted to usable information**
- **incremental computation**
- **demand driven computation**

**What should be computed?**
- **Maximum Computation**
- **Minimum Computation**

**When should it be computed?**
- **Early Computation**
- **Late Computation**

*Do not compute what you don’t need!*

*Who defines what is needed?*  
**Algorithm, Data Structure**
LFCPA Lessons: The Larger Perspective

- **exhaustive computation**
- **computation restricted to usable information**
- **incremental computation**
- **demand driven computation**

**Maximum Computation**

**Early Computation**

Avoid computing some values because

- they have been computed before, or
- they can just be “adjusted”, or
- they are equivalent to some other values

E.g. Value based termination of call strings, Work list based methods, BDDs

Do not compute what you don’t need!

Who defines what is needed? **Algorithm, Data Structure**
LFCPA Lessons: The Larger Perspective

- exhaustive computation
- computation restricted to usable information
- incremental computation
- demand driven computation

What should be computed?

Maximum Computation

When should it be computed?

Minimum Computation

Early Computation

Late Computation

Do not compute what you don’t need!

Who defines what is needed?

Definition of Analysis

Sep 2017 IIT Bombay
LFCPA Lessons: The Larger Perspective

- exhaustive computation
- computation restricted to usable information
- incremental computation
- demand driven computation

What should be computed?

- Maximum Computation
- Early Computation
- When should it be computed?

- Minimum Computation
- Late Computation

Do not compute what you don’t need!

Who defines what is needed? No One!
LFCPA Lessons: The Larger Perspective

- exhaustive computation
- computation restricted to usable information
- incremental computation
- demand driven computation

What should be computed?

Maximum Computation

What is needed?

Minimum Computation

When should it be computed?

Early Computation

Who defines what is needed?

Late Computation

Do not compute what you don’t need!

These seem orthogonal and may be used together

Sep 2017 IIT Bombay
Tutorial Problems for FCPA and LFCPA

- Perform may points-to analysis by deriving must info using “?” in BI
- Perform liveness based points-to analysis

```
1 b = &a
2 c = b
3 a = &b
4 a = &c
5 a = *a
6 *b = c
7 use c

1 y = &z
2 z = &w
3 x = &u
4 x = &v
5 t = *y
6 *x = t
7 use u
```
An Outline of Pointer Analysis Coverage

- The larger perspective
- Comparing Points-to and Alias information
- Flow Insensitive Points-to Analysis
- Flow Sensitive Points-to Analysis
- Pointer Analyses: An Engineer’s Landscape
- Liveness Based Points-to Analysis
- Generalizations to Heap, Arrays, Pointer Arithmetic, and Unions
Original LFCPA Formulation

Data flow equations
\(Lin/Lout, \ Ain/Aout\)

Extractors for statements
\(Def, \ Kill, \ Ref, \ Pointee\)

Lattices
\(2^P \times Var, \ 2^P\)

Named locations
Variables \(Var, \ Pointers \ P,\)
Formulating Generalizations in LFCPA

Data flow equations
$Lin/Lout, Ain/Aout$

Extractors for statements
$Def, Kill, Ref, Pointee$

Extractors for pointer expressions
$\text{lval}, \text{rval}, \text{deref}, \text{ref}$

Lattices
$2^{S \times T}, 2^S$

Named locations

Variables $\text{Var}$, Pointers $\text{P}$, Allocation Sites $\text{H}$, Fields $F, pF, npF$, Offsets $C$
Generalization for Heap and Structures

- Grammar.

$$\alpha := malloc \mid \&\beta \mid \beta$$
$$\beta := x \mid \beta.f \mid \beta \rightarrow f \mid \&\beta$$

where $\alpha$ is a pointer expression, $x$ is a variable, and $f$ is a field

- Memory model: Named memory locations. No numeric addresses

$$S = P \cup H \cup S_p$$  \hspace{1cm} \text{(source locations)}
$$T = \mathbb{Var} \cup H \cup S_m \cup \{?\}$$  \hspace{1cm} \text{(target locations)}
$$S_p = R \times npF^* \times pF$$  \hspace{1cm} \text{(pointers in structures)}
$$S_m = R \times npF^* \times (pF \cup npF)$$  \hspace{1cm} \text{(other locations in structures)}
typedef struct B {
    ... 
    struct B *f;
} sB;
typedef struct A {
    ... 
    struct B g;
} sA;

sA *a;
sB *x, *y, b;

1. a = (sA*) malloc (sizeof(sA));
2. y = &a->g;
3. b.f = y;
4. x = &b;
5. y.f = &x;
6. return x->f->f;

### Pointer Expression Table

<table>
<thead>
<tr>
<th>Pointer Expression</th>
<th>l-value</th>
<th>r-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
<td>b</td>
</tr>
<tr>
<td>x \rightarrow f</td>
<td>b.f</td>
<td>o₁.g.f</td>
</tr>
<tr>
<td>x \rightarrow f \rightarrow f</td>
<td>o₁.g.f</td>
<td>b</td>
</tr>
</tbody>
</table>
L- and R-values of Pointer Expressions

\[ lval(\alpha, A) = \begin{cases} 
\{\sigma\} & (\alpha \equiv \sigma) \land (\sigma \in \text{Var}) \\
\{\sigma.f \mid \sigma \in lval(\beta, A)\} & \alpha \equiv \beta.f \\
\{\sigma.f \mid \sigma \in rval(\beta, A), \sigma \neq \?\} & \alpha \equiv \beta \rightarrow f \\
\{\sigma \mid \sigma \in rval(\beta, A), \sigma \neq \?\} & \alpha \equiv *\beta \\
\emptyset & \text{otherwise}
\end{cases} \]

\[ rval(\alpha, A) = \begin{cases} 
lval(\beta, A) & \alpha \equiv \&\beta \\
\{o_i\} & \alpha \equiv malloc \land o_i = get_heap_loc() \\
A(lval(\alpha, A) \cap S) & \text{otherwise}
\end{cases} \]
Defining Extractor Functions

- Pointer assignment statement $lhs_n = rhs_n$
  
  $Def_n = lval(lhs_n, Ain_n)$
  
  $Kill_n = lval(lhs_n, Must(Ain_n))$
  
  $Ref_n = \begin{cases} 
  \text{deref}(lhs_n, Ain_n) & \text{if } Def_n \cap Lout_n = \emptyset \\
  \text{deref}(lhs_n, Ain_n) \cup \text{ref}(rhs_n, Ain_n) & \text{otherwise} 
  \end{cases}$
  
  $Pointee_n = rval(rhs_n, Ain_n)$

- Use $\alpha$ statement

  $Def_n = Kill_n = Pointee_n = \emptyset$

  $Ref_n = \text{ref}(\alpha, Ain_n)$

- Any other statement

  $Def_n = Kill_n = Ref_n = Pointee_n = \emptyset$
Extensions for Handling Arrays and Pointer Arithmetic

- Grammar.
  \[
  \alpha := \text{malloc} \mid \&\beta \mid \beta \mid \&\beta + e \\
  \beta := x \mid \beta.f \mid \beta \to f \mid \ast\beta \mid \beta[e] \mid \beta + e
  \]

- Memory model: Named memory locations. No numeric addresses
  - No address calculation
  - R-values of index expressions retained for each dimension
    If \( rval(x) = 10 \), then \( lval(a.f[5][2 + x].g) = a.f.5.12.g \)
  - Sizes of the array elements ignored

\[
S = P \cup H \cup G_p \\
T = \mathbb{V}ar \cup H \cup G_m \cup \{?\} \\
G_p = R \times (C \cup npF)^* \times (C \cup pF) \\
G_m = R \times (C \cup npF)^* \times (C \cup pF \cup npF)
\]

(source locations) (target locations) (pointers in aggregates) (locations in aggregates)
Extending L-Value Computation to Arrays and Pointer Arithmetic

- Pointer arithmetic does not have an l-value
- For handling arrays
  - evaluate index expressions using \( \text{eval} \) and accumulate offsets
  - if \( e \) cannot be evaluated at compile time, \( \text{eval} = \perp_{\text{eval}} \)
    (i.e. array accesses in that dimension are treated as index-insensitive)

\[
\text{lval}(\alpha, A) = \begin{cases} 
\{ \sigma \} & (\alpha \equiv \sigma) \land (\sigma \in \text{Var}) \\
\{ \sigma.f \mid \sigma \in \text{lval}(\beta, A) \} & \alpha \equiv \beta.f \\
\{ \sigma.f \mid \sigma \in \text{rval}(\beta, A), \sigma \neq ? \} & \alpha \equiv \beta \rightarrow f \\
\{ \sigma \mid \sigma \in \text{rval}(\beta, A), \sigma \neq ? \} & \alpha \equiv \ast \beta \\
\{ \sigma.\text{eval} \mid \sigma \in \text{lval}(\beta, A) \} & \alpha \equiv \beta[e] \\
\emptyset & \text{otherwise}
\end{cases}
\]
Extending R-Value Computation to Arrays and Pointer Arithmetic

For handling pointer arithmetic

- If the r-value of the pointer is an array location, add `eval` to the offset
- Otherwise, over-approximate the pointees to all possible locations

$$rval(\alpha, A) = \begin{cases} 
\text{lval}(\beta, A) & \alpha \equiv \& \beta \\
\{o_i\} & \alpha \equiv \text{malloc} \land o_i = \text{get_heap_loc}() \\
T & (\alpha \equiv \beta + e) \land \\
(\exists \sigma \in rval(\beta, A), \sigma \neq \sigma'.c, \sigma' \in T, c \in C) \\
\bigcup \{\sigma.(c + evale)\} & (\alpha \equiv \beta + e) \land \\
(\sigma.c \in rval(\beta, A)) \land (c \in C) \\
A(\text{lval}(\alpha, A) \cap S) & \text{otherwise}
\end{cases}$$
Part 6

Heap Reference Analysis
Motivating Example for Heap Liveness Analysis

If the while loop is not executed even once.

1. \(w = x\)  
   
2. while (\(x.data < \text{max}\))
3. \(x = x.rptr\)
4. \(y = x.lptr\)
5. \(z = \text{New class_of_z}\)
6. \(y = y.lptr\)
7. \(z.sum = x.data + y.data\)
If the while loop is executed once.

1. \( w = x \quad // \ x \text{ points to } m_a \)
2. \( \text{while (}x.\text{data} < \text{max}) \)
3. \( x = x.\text{rptr} \)
4. \( y = x.\text{lptr} \)
5. \( z = \text{New class}\_\text{of}\_z \)
6. \( y = y.\text{lptr} \)
7. \( z.\text{sum} = x.\text{data} + y.\text{data} \)
Motivating Example for Heap Liveness Analysis

If the while loop is executed twice.

1. \( w = x \)  // \( x \) points to \( m_a \)
2. \( \text{while } (x.data < \text{max}) \)
3. \( x = x.rptr \)
4. \( y = x.lptr \)
5. \( z = \text{New class\_of\_z} \)
6. \( y = y.lptr \)
7. \( z.sum = x.data + y.data \)
The Moral of the Story

- Mappings between access expressions and l-values keep changing

- This is a *rule* for heap data
  For stack and static data, it is an *exception*!

- Static analysis of programs has made significant progress for stack and static data.

What about heap data?
- Given two access expressions at a program point, do they have the same l-value?
- Given the same access expression at two program points, does it have the same l-value?
Our Solution

```java
y = z = null
1 w = x
w = null
2 while (x.data < max)
   { x.lptr = null
3   x = x.rptr   }
x.rptr = x.lptr.rptr = null
x.lptr.lptr.lptr = null
x.lptr.lptr.rptr = null
4 y = x.lptr
x.lptr = y.rptr = null
y.lptr.lptr = y.lptr.rptr = null
5 z = New class of z
z.lptr = z.rptr = null
6 y = y.lptr
y.lptr = y.rptr = null
7 z.sum = x.data + y.data
x = y = z = null
```
Our Solution

```
y = z = null
1 w = x
    w = null
2 while (x.data < max)
    { x.lptr = null
3    x = x.rptr
    x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
    x.lptr.lptr.rptr = null
4    y = x.lptr
    x.lptr = y.rptr = null
    y.lptr.lptr = y.lptr.rptr = null
5    z = New class of z
    z.lptr = z.rptr = null
6    y = y.lptr
    y.lptr = y.rptr = null
7    z.sum = x.data + y.data
    x = y = z = null
```
Our Solution

\[
y = z = \text{null}\\
1\quad w = x\\
\quad w = \text{null}\\
2\quad \text{while} \ (x.\text{data} < \text{max})\\
\quad \quad \{
\quad \quad x.\text{lptr} = \text{null}\\
\quad \quad x = x.\text{rptr}
\quad \}
\quad x.\text{rptr} = x.\text{lptr}.\text{rptr} = \text{null}\\
\quad x.\text{lptr}.\text{lptr}.\text{lptr} = \text{null}\\
\quad x.\text{lptr}.\text{lptr}.\text{rptr} = \text{null}\\
3\quad y = x.\text{lptr}\\
\quad x.\text{lptr} = y.\text{rptr} = \text{null}\\
\quad y.\text{lptr}.\text{lptr} = y.\text{lptr}.\text{rptr} = \text{null}\\
4\quad z = \text{New class of} \ z\\
\quad z.\text{lptr} = z.\text{rptr} = \text{null}\\
5\quad y = y.\text{lptr}\\
\quad y.\text{lptr} = y.\text{rptr} = \text{null}\\
6\quad z.\text{sum} = x.\text{data} + y.\text{data}\\
\quad x = y = z = \text{null}
\]

While loop is not executed even once
Our Solution

```
y = z = null
1 w = x
   w = null
2 while (x.data < max)
   {  x.lptr = null
      x = x.rptr
      x.rptr = x.lptr.rptr = null
   }
  x.lptr.lptr.lptr = null
  x.lptr.lptr.rptr = null
3 x.rptr = x.lptr = y.rptr = null
4 y = x.lptr
   x.lptr = y.rptr = null
   y.lptr.lptr = y.lptr.rptr = null
5 z = New class of z
   z.lptr = z.rptr = null
6 y = y.lptr
   y.lptr = y.rptr = null
7 z.sum = x.data + y.data
   x = y = z = null
```

While loop is not executed even once
Our Solution

```
y = z = null
1  w = x
   w = null
2  while (x.data < max)
   {   x.lptr = null
3      x = x.rptr
       x.rptr = x.lptr.rptr = null
       x.lptr.lptr.lptr = null
       x.lptr.lptr.rptr = null
4     y = x.lptr
     x.lptr = y.rptr = null
     y.lptr.lptr = y.lptr.rptr = null
5    z = New class of z
    z.lptr = z.rptr = null
6    y = y.lptr
    y.lptr = y.rptr = null
7    z.sum = x.data + y.data
    x = y = z = null
```
Our Solution

y = z = null
1  w = x
    w = null
2  while (x.data < max)
    {   x.lptr = null
    3      x = x.rptr   }
    x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
    x.lptr.lptr.rptr = null
4  y = x.lptr
    x.lptr = y.rptr = null
    y.lptr.lptr = y.lptr.rptr = null
5  z = New class_of_z
    z.lptr = z.rptr = null
6  y = y.lptr
    y.lptr = y.rptr = null
7  z.sum = x.data + y.data
    x = y = z = null

While loop is not executed even once
Our Solution

y = z = null

1. w = x
   w = null

2. while (x.data < max)
   {
      x.lptr = null
      x = x.rptr
   }
   x.rptr = x.lptr.rptr = null
   x.lptr.lptr.lptr = null
   x.lptr.lptr.rptr = null

3. y = x.lptr
   x.lptr = y.rptr = null
   y.lptr.lptr = y.lptr.rptr = null

4. z = New class_of_z
   z.lptr = z.rptr = null

5. y = y.lptr
   y.lptr = y.rptr = null

6. z.sum = x.data + y.data
   x = y = z = null

While loop is not executed even once
Our Solution

```plaintext
y = z = null
1 w = x
   w = null
2 while (x.data < max)
   { x.lptr = null
   x.rptr = x.lptr.rptr = null
   x.lptr.lptr.lptr = null
   x.lptr.lptr.rptr = null
3   x = x.rptr
   x.rptr = x.lptr.rptr = null
   x.lptr.lptr.lptr = null
   x.lptr.lptr.rptr = null
4 y = x.lptr
   x.lptr = y.rptr = null
   y.lptr.lptr = y.lptr.rptr = null
5 z = New class_of_z
   z.lptr = z.rptr = null
6 y = y.lptr
   y.lptr = y.rptr = null
7 z.sum = x.data + y.data
   x = y = z = null
```

While loop is not executed even once
Our Solution

1. \( y = z = \text{null} \)
   \[ w = x \]
   \[ w = \text{null} \]

2. while \((x.\text{data} < \text{max})\)
   \[
   \begin{align*}
   x.\text{lptr} &= \text{null} \\
   x &= x.\text{rptr}
   \end{align*}
   \]
   \[ x.\text{rptr} = x.\text{lptr}.\text{rptr} = \text{null} \]
   \[ x.\text{lptr}.\text{lptr}.\text{lptr} = \text{null} \]
   \[ x.\text{lptr}.\text{lptr}.\text{rptr} = \text{null} \]

3. \( y = x.\text{lptr} \)
   \[ x.\text{lptr} = y.\text{rptr} = \text{null} \]
   \[ y.\text{lptr}.\text{lptr} = y.\text{lptr}.\text{rptr} = \text{null} \]

4. \( z = \text{New class of } z \)
   \[ z.\text{lptr} = z.\text{rptr} = \text{null} \]

5. \( y = y.\text{lptr} \)
   \[ y.\text{lptr} = y.\text{rptr} = \text{null} \]

6. \( z.\text{sum} = x.\text{data} + y.\text{data} \)
   \[ x = y = z = \text{null} \]
**Our Solution**

```
y = z = null
1    w = x
       w = null
2    while (x.data < max)
       {
3       x = x.rptr    }
       x.rptr = x.lptr.rptr = null
       x.lptr.lptr.lptr = null
       x.lptr.lptr.rptr = null
4    y = x.lptr
       x.lptr = y.rptr = null
       y.lptr.lptr = y.lptr.rptr = null
5    z = New class of z
       z.lptr = z.rptr = null
6    y = y.lptr
       y.lptr = y.rptr = null
7    z.sum = x.data + y.data
       x = y = z = null
```

While loop is executed twice
Some Observations

1
   y = z = null
   w = x
   w = null

2
   while (x.data < max)
   {
      x.lptr = null
      x = x.rptr
   }
   x.rptr = x.lptr.rptr = null
   x.lptr.lptr.lptr = null
   x.lptr.lptr.rptr = null

3
   y = x.lptr
   x.lptr = y.rptr = null
   y.lptr.lptr = y.lptr.rptr = null

4
   z = New class of z
   z.lptr = z.rptr = null

5
   y = y.lptr
   y.lptr = y.rptr = null

6
   z.sum = x.data + y.data
   x = y = z = null

Node \( i \) is live but link \( a \rightarrow i \) is nullified
Some Observations

- The memory address that $x$ holds when the execution reaches a given program point is not an invariant of program execution.

```plaintext
y = z = null
1 w = x
   w = null
2 while (x.data < max)
   {
      x.lptr = null
      x = x.rptr
   }
   x.rptr = x.lptr.rptr = null
   x.lptr.lptr.lptr = null
   x.lptr.lptr.rptr = null
3 y = x.lptr
   x.lptr = y.rptr = null
   y.lptr.lptr = y.lptr.rptr = null
4 y = x.lptr
   x.lptr = y.rptr = null
   y.lptr.lptr = y.lptr.rptr = null
5 z = New class of z
   z.lptr = z.rptr = null
6 y = y.lptr
   y.lptr = y.rptr = null
7 z.sum = x.data + y.data
   x = y = z = null
```

Stack

Heap

Sep 2017
Some Observations

- The memory address that \( x \) holds when the execution reaches a given program point is not an invariant of program execution.

- Whether we dereference lptr out of \( x \) or rptr out of \( x \) at a given program point is an invariant of program execution.

```plaintext
y = z = null

1  w = x
   w = null

2  while (x.data < max)
   {  x.lptr = null
       x = x.rptr
   }
   x.rptr = x.lptr.rptr = null
   x.lptr.lptr.lptr = null
   x.lptr.lptr.rptr = null

3  y = x.lptr
   x.lptr = y.rptr = null
   y.lptr.lptr = y.lptr.rptr = null

4  z = New class of z
   z.lptr = z.rptr = null

5  y = y.lptr
   y.lptr = y.rptr = null

6  z.sum = x.data + y.data

7  x = y = z = null
```
Some Observations

1. The memory address that $x$ holds when the execution reaches a given program point is not an invariant of program execution.
2. Whether we dereference lptr out of $x$ or rptr out of $x$ at a given program point is an invariant of program execution.
3. A static analysis can discover only invariants.

```plaintext
y = z = null
w = x
w = null
while (x.data < max) {
    x.lptr = null
    x = x.rptr
} x.rptr = x.lptr.rptr = null
x.lptr.lptr.lptr = null
x.lptr.rptr = null
y = x.lptr
x.lptr = y.rptr = null
y.lptr.lptr = y.lptr.rptr = null
z = New class_of_z
z.lptr = z.rptr = null
y = y.lptr
y.lptr = y.rptr = null
z.sum = x.data + y.data
x = y = z = null
```
Some Observations

New access expressions are created. Can they cause exceptions?

```plaintext
y = z = null
1 w = x
   w = null
2 while (x.data < max)
   { x.lptr = null
     x = x.rptr
   }
   x.rptr = x.lptr.rptr = null
   x.lptr.lptr.lptr = null
   x.lptr.lptr.rptr = null
3 y = x.lptr
   x.lptr = y.rptr = null
   y.lptr.lptr = y.lptr.rptr = null
4 z = New class of z
   z.lptr = z.rptr = null
5 y = y.lptr
   y.lptr = y.rptr = null
6 y = y.lptr
   y.lptr = y.rptr = null
7 z.sum = x.data + y.data
   x = y = z = null
```
An Overview of Heap Reference Analysis

- A reference (called a *link*) can be represented by an *access path*.

  Eg. “\( x \rightarrow lptr \rightarrow rptr \)”

- A link may be accessed in multiple ways

- Setting links to null

  - **Alias Analysis.** Identify all possible ways of accessing a link
  
  - **Liveness Analysis.** For each program point, identify “dead” links (i.e. links which are not accessed after that program point)
  
  - **Availability and Anticipability Analyses.** Dead links should be reachable for making null assignment.
  
  - **Code Transformation.** Set “dead” links to null
Assumptions

For simplicity of exposition

- Java model of heap access
  - Root variables are on stack and represent references to memory in heap.
  - Root variables cannot be pointed to by any reference.

- Simple extensions for C++
  - Root variables can be pointed to by other pointers.
  - Pointer arithmetic is not handled.
Key Idea #1: Access Paths Denote Links

- Root variables: $x, y, z$
- Field names: $\text{rptr}, \text{lptr}$
- Access path: $x \rightarrow \text{rptr} \rightarrow \text{lptr}$
  Semantically, sequence of “links”
- Frontier: name of the last link
- Live access path: If the link corresponding to its frontier is used in future
What Makes a Link Live?

Assuming that a statement must be executed, if nullifying a link read in the statement can change the semantics of the program, then the link is live.

Reading a link for accessing the contents of the corresponding target object:

<table>
<thead>
<tr>
<th>Example</th>
<th>Objects read</th>
<th>Live access paths</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sum = x.rptr.data</code></td>
<td><code>x, O1, O2</code></td>
<td><code>x, x -&gt; rptr</code></td>
</tr>
<tr>
<td><code>if (x.rptr.data &lt; sum)</code></td>
<td><code>x, O1, O2</code></td>
<td><code>x, x -&gt; rptr</code></td>
</tr>
</tbody>
</table>
What Makes a Link Live?

Assuming that a statement must be executed, if nullifying a link read in the statement can change the semantics of the program, then the link is live.

Reading a link for copying the contents of the corresponding target object:

<table>
<thead>
<tr>
<th>Example</th>
<th>Objects read</th>
<th>Live access paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y = x.rptr$</td>
<td>$x, O_1$</td>
<td>$x, x.rptr$</td>
</tr>
</tbody>
</table>

---

*Sep 2017 IIT Bombay*
What Makes a Link Live?

Assuming that a statement must be executed, if nullifying a link read in the statement can change the semantics of the program, then the link is live.

Reading a link for *copying the contents of the corresponding target object*:

<table>
<thead>
<tr>
<th>Example</th>
<th>Objects read</th>
<th>Live access paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y = x.rptr$</td>
<td>$x, O_1$</td>
<td>$x, x.rptr$</td>
</tr>
<tr>
<td>$x.lptr = y$</td>
<td>$x, O_1, y$</td>
<td>$x, y$</td>
</tr>
</tbody>
</table>
What Makes a Link Live?

Assuming that a statement must be executed, if nullifying a link read in the statement can change the semantics of the program, then the link is live.

Reading a link for comparing the address of the corresponding target object:

<table>
<thead>
<tr>
<th>Example</th>
<th>Objects read</th>
<th>Live access paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>if ((x.\text{lptr} == \text{null}))</td>
<td>(x, O_1)</td>
<td>(x, x \rightarrow \text{lptr})</td>
</tr>
</tbody>
</table>
What Makes a Link Live?

Assuming that a statement must be executed, if nullifying a link read in the statement can change the semantics of the program, then the link is live.

Reading a link for comparing the address of the corresponding target object:

<table>
<thead>
<tr>
<th>Example</th>
<th>Objects read</th>
<th>Live access paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>if (x.lptr == null)</td>
<td>x, O₁</td>
<td>x, x→lptr</td>
</tr>
<tr>
<td>if (y == x.lptr)</td>
<td>x, O₁, y</td>
<td>x, x→lptr, y</td>
</tr>
</tbody>
</table>
Liveness Analysis

Program

Statement involving memory references

Live Access Paths

Effect of the statement on the access paths

Live Access Paths

Semantic Information

Sep 2017
Key Idea #2: Transfer of Access Paths

\[ x = x.n \]
Key Idea #2: Transfer of Access Paths

\[ x = x.n \]
Key Idea #2: Transfer of Access Paths

\[ x = x.n \]

\[ \ldots = x.r.d \]
Key Idea #2: Transfer of Access Paths

\[ x = x.n \]

\( \{ x, x \rightarrow r \} \)

\[ \ldots = x.r.d \]
Key Idea #2: Transfer of Access Paths

\[ x = x.n \]

\[ \ldots = x.r.d \]

\{ x, x \rightarrow r \}
Key Idea #2: Transfer of Access Paths

\[ x = x.n \]

\[ \ldots = x.r.d \]

Analysis

\[ \{ x, x \rightarrow r \} \]
Key Idea #2: Transfer of Access Paths

\[ x = x.n \]

\[ \ldots = x.r.d \]

\{ x, x \rightarrow r \}
Key Idea #2: Transfer of Access Paths

\[ x = x.n \]

\[ \ldots = x.r.d \]

Generated: \( \{x, x \rightarrow n, x \rightarrow n \rightarrow r\} \)

Killed: \( \{x, x \rightarrow r\} \)

\( x \) after the assignment is same as \( x \rightarrow n \) before the assignment.
Key Idea #3: Liveness Closure Under Link Aliasing

\[ x = y \]

\[ \ldots = x.n.d \]
Key Idea #3: Liveness Closure Under Link Aliasing

\[ x = y \]

\[ \ldots = x.n.d \]

\[ x \text{ and } y \text{ are node aliases} \]
\[ x.n \text{ and } y.n \text{ are link aliases} \]
\[ x \rightarrow n \text{ is live } \Rightarrow y \rightarrow n \text{ is live} \]
Key Idea #3: Liveness Closure Under Link Aliasing

$x = y$

$x$ and $y$ are node aliases

$x.n$ and $y.n$ are link aliases

$x \rightarrow n$ is live $\Rightarrow$ $y \rightarrow n$ is live

Nullifying $y \rightarrow n$ will have the side effect of nullifying $x \rightarrow n$
Explicit and Implicit Liveness

\[ x = y \]

\[ \ldots = x.n.d \]

\[ x \rightarrow n \text{ is live} \Rightarrow y \rightarrow n \text{ is live} \]
Explicit and Implicit Liveness

\[
x = y
\]

\[
\ldots = x.n.d
\]

\[
x \rightarrow n \text{ is live} \Rightarrow y \rightarrow n \text{ is live}
\]

\[
y \rightarrow n \text{ is implicitly live}
\]

\[
x \rightarrow n \text{ is explicitly live}
\]
Key Idea #4: Aliasing is Required with Explicit Liveness

1. \( x = y \)
2. \( x.p = t \)
3. \( y = y.p \)
4. use \( y.q.d \)
Key Idea #4: Aliasing is Required with Explicit Liveness

```
1  x = y

{ x, y, y->p, y->p->q }

2  x.p = t

{ y, y->p, y->p->q }

3  y = y.p

{ y, y->q }

4  use y.q.d
```
Key Idea #4: Aliasing is Required with Explicit Liveness

1. \( x = y \)
   - \( \{ x, y, y \rightarrow p, y \rightarrow p \rightarrow q \} \)

2. \( x.p = t \)
   - \( \{ y, y \rightarrow p, y \rightarrow p \rightarrow q \} \)

3. \( y = y.p \)
   - \( \{ y, y \rightarrow q \} \)

4. use \( y.q.d \)
Key Idea #4: Aliasing is Required with Explicit Liveness

1. \( x = y \)

   \( \{ x, y, y \rightarrow p, y \rightarrow p \rightarrow q \} \)

2. \( x.p = t \)

   \( \{ y, y \rightarrow p, y \rightarrow p \rightarrow q \} \)

3. \( y = y.p \)

   \( \{ y, y \rightarrow q \} \)

4. \( \text{use } y.q.d \)
Key Idea #4: Aliasing is Required with Explicit Liveness

1. \( x = y \)
   \[ \{x, y, y \rightarrow p, y \rightarrow p \rightarrow q\} \]

2. \( x.p = t \)
   \[ \{y, y \rightarrow p, y \rightarrow p \rightarrow q\} \]

3. \( y = y.p \)
   \[ \{y, y \rightarrow q\} \]

4. Use \( y.q.d \)
Key Idea #4: Aliasing is Required with Explicit Liveness

1. \( x = y \) 

\[ \{ x, y, y \to p, y \to p \to q \} \]

2. \( x.p = t \) 

\[ \{ y, y \to p, y \to p \to q \} \]

3. \( y = y.p \) 

\[ \{ y, y \to q \} \]

4. use \( y.q.d \)

Explicit Liveness

Effect of Aliasing

\[ y \to p \equiv x \to p \]
\[ y \to p \to q \equiv x \to p \to q \]
Key Idea #4: Aliasing is Required with Explicit Liveness

1. \( x = y \)
   \[ \{ x, y, y \rightarrow p, y \rightarrow p \rightarrow q \} \]
   \[ \{ x, y, t, t \rightarrow q \} \]

2. \( x.p = t \)
   \[ \{ y, y \rightarrow p, y \rightarrow p \rightarrow q \} \]
   \[ \{ y, y \rightarrow p, y \rightarrow p \rightarrow q \} \]

3. \( y = y.p \)
   \[ \{ y, y \rightarrow q \} \]
   \[ \{ y, y \rightarrow q \} \]

Effect of Aliasing:
\( y \rightarrow p \equiv x \rightarrow p \)
\( y \rightarrow p \rightarrow q \equiv x \rightarrow p \rightarrow q \)
Key Idea #4: Aliasing is Required with Explicit Liveness

1. $x = y$

2. $x.p = t$

3. $y = y.p$

4. use $y.q.d$

Explicit Liveness

- $\{x, y, \{ y \to p, y \to p \to q \}\}$
- $\{y, y \to p, y \to p \to q\}$

Required Liveness

- $\{x, y, \{ t, t \to q \}\}$
- $\{y, y \to p, y \to p \to q\}$

Effect of Aliasing

- $y \to p \equiv x \to p$
- $y \to p \to q \equiv x \to p \to q$

Spurious

- $\{y, y \to p, y \to p \to q\}$

Missing

- $\{y, y \to q\}$
- $\{y, y \to q\}$

Sep 2017
Key Idea #4: Aliasing is Required with Explicit Liveness

1. $x = y$
   - **Explicit Liveness**: $\{x, y, y \rightarrow p, y \rightarrow p \rightarrow q\}$
   - **Required Liveness**: $\{x, y, t, t \rightarrow q\}$

2. $x.p = t$
   - **Spurious**: $y \rightarrow p \equiv x \rightarrow p$
   - **Missing**: $y \rightarrow p \rightarrow q \equiv x \rightarrow p \rightarrow q$

3. $y = y.p$
   - **Effect of Aliasing**: $y \rightarrow p \rightarrow q$

4. use $y.q.d$

The need of link alias closure of LHS
- Transferring liveness to RHS (soundness)
- Killing liveness (precision)

Link alias closure of RHS can be computed later for implicit liveness
Notation for Defining Flow Functions for Explicit Liveness

- Basic entities
  - Variables $u, v \in \mathbb{V}_{ar}$
  - Pointer variables $w, x, y, z \in P \subseteq \mathbb{V}_{ar}$
  - Pointer fields $f, g, h \in pF$
  - Non-pointer fields $a, b, c, d \in npF$

- Additional notation
  - Sequence of pointer fields $\sigma \in pF^*$ (could be $\epsilon$)
  - Access paths $\rho \in P \times pF^*$
    Example: $\{x, x \rightarrow f, x \rightarrow f \rightarrow g\}$
  - Summarized access paths rooted at $x$ or $x \rightarrow \sigma$ for a given $x$ and $\sigma$
    - $x \rightarrow \ast = \{x \rightarrow \sigma \mid \sigma \in pF^*\}$
    - $x \rightarrow \sigma \rightarrow \ast = \{x \rightarrow \sigma \rightarrow \sigma' \mid \sigma' \in pF^*\}$
Data Flow Equations for Explicit Liveness Analysis

\[ \ln_n = (\text{Out}_n - \text{Kill}_n(\text{Out}_n)) \cup \text{Gen}_n(\text{Out}_n) \]

\[ \text{Out}_n = \begin{cases} 
    \text{Bl} & \text{if } n \text{ is End} \\
    \bigcup_{s \in \text{succ}(n)} \text{In}_s & \text{otherwise}
\end{cases} \]
Let $A$ denote May Aliases at the exit of node $n$

<table>
<thead>
<tr>
<th>Statement $n$</th>
<th>$\text{Gen}_n(X)$</th>
<th>$\text{Kill}_n(X)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = y$</td>
<td>${y \rightarrow \sigma \mid x \rightarrow \sigma \in X}$</td>
<td>$x \rightarrow*$</td>
</tr>
<tr>
<td>$x = y.f$</td>
<td>${y \rightarrow f \rightarrow \sigma \mid x \rightarrow \sigma \in X}$</td>
<td>$x \rightarrow*$</td>
</tr>
<tr>
<td>$x.f = y$</td>
<td>$\left{ y \rightarrow \sigma \mid z \rightarrow f \rightarrow \sigma \in X, z \in A(x) \right}$</td>
<td>$\bigcup_{z \in \text{Must}(A)(x)} z \rightarrow f \rightarrow*$</td>
</tr>
<tr>
<td>$x = \text{new}$</td>
<td>$\emptyset$</td>
<td>$x \rightarrow*$</td>
</tr>
<tr>
<td>$x = \text{null}$</td>
<td>$\emptyset$</td>
<td>$x \rightarrow*$</td>
</tr>
<tr>
<td>other</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
Flow Functions for Explicit Liveness Analysis

Let $A$ denote May Aliases at the exit of node $n$

<table>
<thead>
<tr>
<th>Statement $n$</th>
<th>$\text{Gen}_n(X)$</th>
<th>$\text{Kill}_n(X)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = y$</td>
<td>${y \mapsto \sigma \mid x \mapsto \sigma \in X}$</td>
<td>$x \mapsto *$</td>
</tr>
<tr>
<td>$x = y.f$</td>
<td>${y \mapsto f \mapsto \sigma \mid x \mapsto \sigma \in X}$</td>
<td>$x \mapsto *$</td>
</tr>
<tr>
<td>$x.f = y$</td>
<td>${y \mapsto \sigma \mid \exists z \in \text{Must}(A) \ (x) \ (z \mapsto f \mapsto \sigma \in X, z \in A(x))}$</td>
<td>$\bigcup_{z \in \text{Must}(A) \ (x)} z \mapsto f \mapsto *$</td>
</tr>
<tr>
<td>$x = \text{new}$</td>
<td>$\emptyset$</td>
<td>$x \mapsto *$</td>
</tr>
<tr>
<td>$x = \text{null}$</td>
<td>$\emptyset$</td>
<td>$x \mapsto *$</td>
</tr>
<tr>
<td>other</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

May link aliasing for soundness
Flow Functions for Explicit Liveness Analysis

Let $A$ denote May Aliases at the exit of node $n$

<table>
<thead>
<tr>
<th>Statement $n$</th>
<th>$\text{Gen}_n(X)$</th>
<th>$\text{Kill}_n(X)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = y$</td>
<td>${y \rightarrow \sigma \mid x \rightarrow \sigma \in X}$</td>
<td>$x \rightarrow^*$</td>
</tr>
<tr>
<td>$x = y.f$</td>
<td>${y \rightarrow f \rightarrow \sigma \mid x \rightarrow \sigma \in X}$</td>
<td>$x \rightarrow^*$</td>
</tr>
<tr>
<td>$x.f = y$</td>
<td>${y \rightarrow \sigma \mid z \rightarrow f \rightarrow \sigma \in X, z \in A(x)}$</td>
<td>$\bigcup_{z \in \text{Must}(A)(x)} z \rightarrow f \rightarrow^*$</td>
</tr>
<tr>
<td>$x = \text{new}$</td>
<td>$\emptyset$</td>
<td>$x \rightarrow^*$</td>
</tr>
<tr>
<td>$x = \text{null}$</td>
<td>$\emptyset$</td>
<td>$x \rightarrow^*$</td>
</tr>
<tr>
<td>other</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

May link aliasing for soundness

Must link aliasing for precision
Flow Functions for Explicit Liveness Analysis

Let $A$ denote May Aliases at the exit of node $n$

<table>
<thead>
<tr>
<th>Statement $n$</th>
<th>$\text{Gen}_n(X)$</th>
<th>$\text{Kill}_n(X)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = y$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x.f = y$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x.f = y$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Why is $y \notin \text{Gen}_n(X)$ for $x.f = y$ when $x \notin X$?
  If $\not\exists x \in \text{Out}_n$, we can do dead code elimination

- Why is $y \notin \text{Gen}_n(X)$ for $x = y.f$ when $x \rightarrow \sigma \notin X$?
  If $\not\exists x \rightarrow \sigma \in \text{Out}_n$, we can do dead code elimination

- Why is $x \notin \text{Gen}_n(X)$ for $x.f = y$?
  - If $\not\exists x \rightarrow f \rightarrow \sigma \in \text{Out}_n$, we can do dead code elimination
  - If $\exists x \rightarrow f \rightarrow \sigma \in \text{Out}_n$, then $\exists x \in \text{Out}_n$
    It will not be killed, so no need of $x \in \text{Gen}_n$
Computing Explicit Liveness Using Sets of Access Paths

\[ x = x.n \]

\[ \{ x, x \rightarrow r \} \]

\[ \ldots = x.r.d \]
Computing Explicit Liveness Using Sets of Access Paths

\[ x = x.n \]
\[ \{ x, x \rightarrow r \} \]
\[ \{ x, x \rightarrow r \} \]
\[ \ldots = x.r.d \]
Computing Explicit Liveness Using Sets of Access Paths

Analysis

\[ x = x.n \]
\[ \{x, x \rightarrow r\} \]
\[ \{x, x \rightarrow r\} \]
\[ \ldots = x.r.d \]

\[ x \rightarrow n \] extended with \( r \)
Computing Explicit Liveness Using Sets of Access Paths

\[ x = x.n \]
\[ \{ x, x \rightarrow n, x \rightarrow n \rightarrow r \} \]
\[ \{ x, x \rightarrow r \} \]
\[ \{ x, x \rightarrow r \} \]
\[ \ldots = x.r.d \]

Diagram:

- Node \( x \)
- Node \( y \)
- Edges: \( n \rightarrow r \rightarrow r \)
- Path: \( x \rightarrow n \rightarrow r \rightarrow d \)
Anticipability of Heap References: An *All Paths* problem

\[ x = x.n \rightarrow \{ x, x \rightarrow n, x \rightarrow n \rightarrow r \} \]

\[ \{ x, x \rightarrow r \} \]

\[ \{ x, x \rightarrow r \} \]

\[ \ldots = x.r.d \]
Computing Explicit Liveness Using Sets of Access Paths

Anticipability of Heap References: An *All Paths* problem

\[
\{x, x \rightarrow n, x \rightarrow n \rightarrow r\} \cap \{x, x \rightarrow r\} \cap \{x, x \rightarrow n, x \rightarrow n \rightarrow r\}
\]

\[
\cdots = x.r.d
\]
Computing Explicit Liveness Using Sets of Access Paths

Anticipability of Heap References: An *All Paths* problem

\[
x = x.n
\]

\[
\{x, x \rightarrow n, x \rightarrow n \rightarrow r\}
\]

\[
\{x\}
\]

\[
\{x, x \rightarrow r\}
\]

\[
\ldots = x.r.d
\]

Sep 2017  IIT Bombay
Anticipability of Heap References: An \textit{All Paths} problem

\[
x = x.n
\]

\[
\{x\}
\]

\[
\{x, x \rightarrow r\}
\]

\[
... = x.r.d
\]
Computing Explicit Liveness Using Sets of Access Paths

Liveness of Heap References: An *Any Path* problem

Analysis

\[
\begin{align*}
{x, x \rightarrow n, x \rightarrow n \rightarrow r} \\
{x = x.n} \\
{x, x \rightarrow r} \\
\ldots = x.r.d
\end{align*}
\]
Liveness of Heap References: An *Any Path* problem

\[
\{x, x \rightarrow n, x \rightarrow n \rightarrow r\}
\]

\[
\{x, x \rightarrow r\} \cup \{x, x \rightarrow n, x \rightarrow n \rightarrow r\}
\]

\[
\{x, x \rightarrow r\}
\]

\[
\ldots = x.r.d
\]
Liveness of Heap References: An *Any Path* problem

\[ x \rightarrow x \text{ extended with } r, n, \text{ and } n \rightarrow r \]

\[ \{x, x \rightarrow r, x \rightarrow n, x \rightarrow n \rightarrow r\} \]

\[ \{x, x \rightarrow r\} \]

\[ \ldots = x.r.d \]
Computing Explicit Liveness Using Sets of Access Paths

Liveness of Heap References: An *Any Path* problem

\[
\begin{align*}
\text{Analysis} & : x = x.n \\
& \downarrow \{ x, x \rightarrow n, x \rightarrow n \rightarrow n, x \rightarrow n \rightarrow r, x \rightarrow n \rightarrow n \rightarrow r \} \\
& \downarrow \{ x, x \rightarrow r, x \rightarrow n, x \rightarrow n \rightarrow r \} \\
& \downarrow \{ x, x \rightarrow r \} \\
\ldots & = x.r.d
\end{align*}
\]
Computing Explicit Liveness Using Sets of Access Paths

Liveness of Heap References: An *Any Path* problem

\[ x = x.n \]

\[ \{ x, x \rightarrow n, x \rightarrow n \rightarrow r, x \rightarrow n \rightarrow n \rightarrow r, x \rightarrow n \rightarrow \cdots \rightarrow n \rightarrow r \} \]

\[ \{ x, x \rightarrow r, x \rightarrow n, x \rightarrow n \rightarrow r, x \rightarrow n \rightarrow \cdots \rightarrow n \rightarrow r \} \]

\[ \{ x, x \rightarrow r \} \]

\[ \ldots = x.r.d \]

*Infinite Number of Unbounded Access Paths*
Key Idea #5: Using Graphs as Data Flow Values

Finite Number of Bounded Structures
Key Idea #6: Include Program Point in Graphs

Different occurrences of n’s in an access path are Indistinguishable

\[ \{x, x \rightarrow n, x \rightarrow n \rightarrow n, x \rightarrow n \rightarrow n \rightarrow n, \ldots\} \]

Different occurrences of n’s in an access path are Distinct

\[ \{x, x \rightarrow n, x \rightarrow n \rightarrow n, x \rightarrow n \rightarrow n \rightarrow r\} \]
Key Idea #6: Include Program Point in Graphs

Different occurrences of n’s in an access path are **Indistinguishable**

Different occurrences of n’s in an access path are **Distinct**

(pattern of subsequent dereferences could be distinct)

\[
\{ x, x \rightarrow n, x \rightarrow n \rightarrow n, x \rightarrow n \rightarrow n \rightarrow n, \ldots \} 
\]
Key Idea #6: Include Program Point in Graphs

Different occurrences of n’s in an access path are **Indistinguishable**
*(pattern of subsequent dereferences remains same)*

\[ \{ x, x \rightarrow n, x \rightarrow n \rightarrow n, x \rightarrow n \rightarrow n \rightarrow n, \ldots \} \]

Different occurrences of n’s in an access path are **Distinct**
*(pattern of subsequent dereferences could be distinct)*

\[ \{ x, x \rightarrow n, x \rightarrow n \rightarrow n, x \rightarrow n \rightarrow n \rightarrow r \} \]
Key Idea #6: Include Program Point in Graphs

1. $x = x.n$

\[
\{ x, x \rightarrow n, x \rightarrow n \rightarrow n, x \rightarrow n \rightarrow n \rightarrow n, \ldots \} 
\]

**Different occurrences of n’s in an access path are indistinguishable**
*(pattern of subsequent dereferences remains same)*

Access Graph:

2. $x = x.n$

\[
\{ x, x \rightarrow n, x \rightarrow n \rightarrow n, x \rightarrow n \rightarrow n \rightarrow r \} 
\]

**Different occurrences of n’s in an access path are distinct**
*(pattern of subsequent dereferences could be distinct)*

Access Graph:
Inclusion of Program Point Facilitates Summarization

1. \( x = x.r \)
2. \( = x.n.d \)
3. \( x = x.r \)
4. \( = x.n.d \)
Inclusion of Program Point Facilitates Summarization

\[ x = x.\text{n.d} \]

\[ x = x.r \]

\[ G_4 \]

\[ x \rightarrow n \quad n_4 \]

\[ = x.\text{n.d} \]

\[ G_4 \]
Inclusion of Program Point Facilitates Summarization
Inclusion of Program Point Facilitates Summarization
Inclusion of Program Point Facilitates Summarization

\[ G_1 = G_2 \cup G_3 \]

\[ x = x.r \]

\[ x = x.n.d \]

\[ x = x.n.d \]

Analysis
Inclusion of Program Point Facilitates Summarization

Iteration #1

Analysis

1 \( x = x.n \)

2 \( \ldots = x.r.d \)
Inclusion of Program Point Facilitates Summarization

Iteration #1

Analysis

1 \[ x = x.n \]

2 \[ ... = x.r.d \]

\[ x \rightarrow r \rightarrow r_2 \]
Inclusion of Program Point Facilitates Summarization

Iteration #1

Analysis

1 $x = x.n$

2 $\ldots = x.r.d$

$x \rightarrow r \rightarrow r_2$

$x \rightarrow r \rightarrow r_2$
Inclusion of Program Point Facilitates Summarization

Iteration #1

1. \( x = x.n \)

2. \( \ldots = x.r.d \)

Diagram:

- \( x \rightarrow n \rightarrow n_1 \rightarrow r_2 \)
- \( x \rightarrow r \rightarrow r_2 \)
- \( x \rightarrow r \rightarrow r_2 \)
Inclusion of Program Point Facilitates Summarization

\[ x = x.n \]

\[ ... = x.r.d \]

Iteration #1

Analysis

\[ x \rightarrow n \rightarrow n_1 \rightarrow r \rightarrow r_2 \]

\[ x \rightarrow r \rightarrow r_2 \]

\[ x \rightarrow r \rightarrow r_2 \]
Inclusion of Program Point Facilitates Summarization

Iteration #2

1. \( x = x.n \)

2. \( \ldots = x.r.d \)

Analysis
Inclusion of Program Point Facilitates Summarization

Iteration #2

Analysis

1 \( x = x.n \)

2 \( \ldots = x.r.d \)
Inclusion of Program Point Facilitates Summarization

Iteration #2

Analysis

1 $x = x.n$

2 $... = x.r.d$

Sep 2017 IIT Bombay
Inclusion of Program Point Facilitates Summarization

Iteration #2

Analysis

1 \[ x = x.n \]

2 \[ \ldots = x.r.d \]
Inclusion of Program Point Facilitates Summarization

1. \( x = x \cdot n \)
2. \( ... = x \cdot r \cdot d \)

Iteration #3

\[ \bigcup_G \left( x \rightarrow n \rightarrow r \rightarrow n_1 \rightarrow r_2 \right) \]

Sep 2017 IIT Bombay
Inclusion of Program Point Facilitates Summarization

Iteration #3

Analysis

1. \( x = x.n \)

2. \( \ldots = x.r.d \)
Inclusion of Program Point Facilitates Summarization

Analysis

1 \[ x = x.n \]

2 \[ \ldots = x.r.d \]

Iteration #3

\[ n \rightarrow n \rightarrow r \rightarrow n \rightarrow x \rightarrow n_1 \rightarrow n_1 \rightarrow r_2 \rightarrow r_2 \rightarrow r_2 \]
Inclusion of Program Point Facilitates Summarization

Iteration #3

Analysis

1  \(x = x.n\)

2  \(\ldots = x.r.d\)
Access Graph and Memory Graph

Program Fragment

\[
x.l = y.r
\]

1

\[
\text{if } (x.l.n == y.r.n)
\]

2
Access Graph and Memory Graph

Program Fragment

\[ x.l = y.r \]

\[ \text{if } (x.l.n == y.r.n) \]

Memory Graph

1

2
Access Graph and Memory Graph

Program Fragment

\[ x.l = y.r \]

if \((x.l.n == y.r.n)\)

Memory Graph

Access Graphs
Access Graph and Memory Graph

Program Fragment

\[ x.l = y.r \]

\[ \text{if} \ (x.l.n == y.r.n) \]

Memory Graph

Access Graphs

- Memory Graph: Nodes represent locations and edges represent links (i.e. pointers).
Access Graph and Memory Graph

Program Fragment

\[ x.l = y.r \]

\[ \text{if} \ (x.l.n == y.r.n) \]

Memory Graph

Access Graphs

- **Memory Graph**: Nodes represent locations and edges represent links (i.e. pointers).

- **Access Graphs**: Nodes represent dereference of links at particular statements. Memory locations are implicit.
Lattice of Access Graphs

- Finite number of nodes in an access graph for a variable

- \( \sqcup \) induces a partial order on access graphs

  \[ \Rightarrow \] a finite (and hence complete) lattice

  \[ \Rightarrow \] All standard results of classical data flow analysis can be extended to this analysis.

  \[ \textit{Termination and boundedness, convergence on MFP, complexity etc.} \]
Access Graph Operations

- **Union.** $G \cup G'$
- **Path Removal**
  $G \ominus R$ removes those access paths in $G$ which have $\rho \in R$ as a prefix
- **Factorization** (/)
- **Extension**
Given statement $x.n = y$, what should be the result of transfer?

<table>
<thead>
<tr>
<th>Live AP</th>
<th>Memory Graph</th>
<th>Transfer</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \to n \to r$</td>
<td><img src="image" alt="Memory Graph" /></td>
<td>$y \to r$</td>
<td>$r$ (LHS is contained in the live access path)</td>
</tr>
<tr>
<td>$x \to n$</td>
<td><img src="image" alt="Memory Graph" /></td>
<td>$y$</td>
<td>$\epsilon$ (LHS is contained in the live access path)</td>
</tr>
<tr>
<td>$x$</td>
<td><img src="image" alt="Memory Graph" /></td>
<td>no transfer</td>
<td>?? (LHS is not contained in the live access path)</td>
</tr>
</tbody>
</table>
### Defining Factorization

Given statement \( x.n = y \), what should be the result of transfer?

<table>
<thead>
<tr>
<th>Live AP</th>
<th>Memory Graph</th>
<th>Transfer</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x \rightarrow n \rightarrow r )</td>
<td>![Memory Graph 1]</td>
<td>( y \rightarrow r )</td>
<td>( r ) (LHS is contained in the live access path)</td>
</tr>
<tr>
<td>( x \rightarrow n )</td>
<td>![Memory Graph 2]</td>
<td>( y )</td>
<td>( \epsilon ) (LHS is contained in the live access path)</td>
</tr>
<tr>
<td>( x )</td>
<td>![Memory Graph 3]</td>
<td>no transfer</td>
<td>?? (LHS is not contained in the live access path)</td>
</tr>
</tbody>
</table>

Quotient is empty
So no remainder
Semantics of Access Graph Operations

- \( P(G) \) is the set of all paths in graph \( G \)
- \( P(G, M) \) is the set of paths in \( G \) terminating on nodes in \( M \)
- \( S \) is the set of remainder graphs
- \( P(S) \) is the set of all paths in all remainder graphs in \( S \)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Access Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>Union</td>
<td>( G_3 = G_1 \sqcup G_2 ) \hspace{1cm} ( P(G_3) \supseteq P(G_1) \cup P(G_2) )</td>
</tr>
<tr>
<td>Path Removal</td>
<td>( G_2 = G_1 \ominus X ) \hspace{1cm} ( P(G_2) \supseteq P(G_1) - { \rho \rightarrow \sigma \mid \rho \in X, \rho \rightarrow \sigma \in P(G_1) } )</td>
</tr>
<tr>
<td>Factorization</td>
<td>( S = G_1/\rho ) \hspace{1cm} ( P(S) = { \sigma \mid \rho \rightarrow \sigma \in P(G_1) } )</td>
</tr>
<tr>
<td>Extension</td>
<td>( G_2 = (G_1, M)#\emptyset ) \hspace{1cm} ( P(G_2) = \emptyset )</td>
</tr>
<tr>
<td></td>
<td>( G_2 = (G_1, M)# S ) \hspace{1cm} ( P(G_2) \supseteq P(G_1) \cup { \rho \rightarrow \sigma \mid \rho \in P(G_1, M), \sigma \in P(S) } )</td>
</tr>
</tbody>
</table>
**Semantics of Access Graph Operations**

- $P(G)$ is the set of all paths in graph $G$
- $P(G, M)$ is the set of paths in $G$ terminating on nodes in $M$
- $S$ is the set of remainder graphs
- $P(S)$ is the set of all paths in all remainder graphs in $S$

<table>
<thead>
<tr>
<th>Operation</th>
<th>Access Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Union</strong></td>
<td>$G_3 = G_1 \uplus G_2$</td>
</tr>
<tr>
<td></td>
<td>$P(G_3) \supseteq P(G_1) \cup P(G_2)$</td>
</tr>
<tr>
<td><strong>Path Removal</strong></td>
<td>$G_2 = G_1 \ominus X$</td>
</tr>
<tr>
<td></td>
<td>$P(G_2) \supseteq P(G_1) - {\rho \rightarrow \sigma \mid \rho \in X, \rho \rightarrow \sigma \in P(G_1)}$</td>
</tr>
<tr>
<td><strong>Factorization</strong></td>
<td>$S = G_1/\rho$</td>
</tr>
<tr>
<td></td>
<td>$P(S) = {\sigma \mid \rho \rightarrow \sigma \in P(G_1)}$</td>
</tr>
<tr>
<td><strong>Extension</strong></td>
<td>$G_2 = (G_1, M) # \emptyset$</td>
</tr>
<tr>
<td></td>
<td>$P(G_2) = \emptyset$</td>
</tr>
<tr>
<td></td>
<td>$P(G_2) \supseteq P(G_1) \cup {\rho \rightarrow \sigma \mid \rho \in P(G_1, M), \sigma \in P(S)}$</td>
</tr>
</tbody>
</table>

$\sigma$ represents remainder
### Access Graph Operations: Examples

<table>
<thead>
<tr>
<th>Program</th>
<th>Access Graphs</th>
<th>Remainder Graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (x = x.l)</td>
<td><img src="#" alt="Graph 1" /></td>
<td><img src="#" alt="Remainder Graph 1" /></td>
</tr>
<tr>
<td>2 (y = x.r.d)</td>
<td><img src="#" alt="Graph 2" /></td>
<td><img src="#" alt="Remainder Graph 2" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Union</th>
<th>Path Removal</th>
<th>Factorisation</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Access Graph Operations: Examples

<table>
<thead>
<tr>
<th>Program</th>
<th>Access Graphs</th>
<th>Remainder Graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = x.!l )</td>
<td>( g_1 )</td>
<td>( rg_1 )</td>
</tr>
<tr>
<td>( y = x.!r.!d )</td>
<td>( g_2 )</td>
<td>( rg_2 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Union</th>
<th>Path Removal</th>
<th>Factorisation</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g_3 \cup g_4 = g_4 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( g_2 \cup g_4 = g_5 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( g_5 \cup g_4 = g_5 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( g_5 \cup g_6 = g_6 )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Access Graph Operations: Examples

<table>
<thead>
<tr>
<th>Program</th>
<th>Access Graphs</th>
<th>Remainder Graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = x.l )</td>
<td>( g_1 )</td>
<td>( rg_1 )</td>
</tr>
<tr>
<td>( y = x.r.d )</td>
<td>( g_4 )</td>
<td>( rg_2 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Union</th>
<th>Path Removal</th>
<th>Factorisation</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g_3 \uplus g_4 = g_4 )</td>
<td>( g_6 \ominus {x \rightarrow l} = g_2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( g_2 \uplus g_4 = g_5 )</td>
<td>( g_5 \ominus {x} = E_G )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( g_5 \uplus g_4 = g_5 )</td>
<td>( g_4 \ominus {x \rightarrow r} = g_4 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( g_5 \uplus g_6 = g_6 )</td>
<td>( g_4 \ominus {x \rightarrow l} = g_1 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Access Graph Operations: Examples

<table>
<thead>
<tr>
<th>Program</th>
<th>Access Graphs</th>
<th>Remainder Graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ( x = x.l )</td>
<td>( g_1 \Rightarrow x )</td>
<td>( rg_1 \Rightarrow r_2 )</td>
</tr>
<tr>
<td>2 ( y = x.r.d )</td>
<td>( g_4 \Rightarrow x \xrightarrow{r_1} r_2 )</td>
<td>( rg_2 \Rightarrow l_1 \rightarrow r_2 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Union</td>
<td>( g_3 \cup g_4 = g_4 )</td>
</tr>
<tr>
<td>Path Removal</td>
<td>( g_6 \ominus { x \xrightarrow{l} } = g_2 )</td>
</tr>
<tr>
<td>Factorisation</td>
<td>( g_2/x = { rg_1 } )</td>
</tr>
<tr>
<td>Extension</td>
<td>( g_2/x = \emptyset )</td>
</tr>
<tr>
<td>Union</td>
<td>( g_2 \cup g_4 = g_5 )</td>
</tr>
<tr>
<td>Path Removal</td>
<td>( g_5 \ominus { x } = \mathcal{E}_G )</td>
</tr>
<tr>
<td>Factorisation</td>
<td>( g_5/x = { rg_1, rg_2 } )</td>
</tr>
<tr>
<td>Extension</td>
<td>( g_5/x = \emptyset )</td>
</tr>
<tr>
<td>Union</td>
<td>( g_5 \cup g_4 = g_5 )</td>
</tr>
<tr>
<td>Path Removal</td>
<td>( g_4 \ominus { x \xrightarrow{r} } = g_4 )</td>
</tr>
<tr>
<td>Factorisation</td>
<td>( g_4/x = { \epsilon_{RG} } )</td>
</tr>
<tr>
<td>Extension</td>
<td>( g_4/x = \emptyset )</td>
</tr>
</tbody>
</table>
## Access Graph Operations: Examples

<table>
<thead>
<tr>
<th>Program</th>
<th>Access Graphs</th>
<th>Remainder Graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 [x = x \cdot l]</td>
<td><img src="chart1" alt="Graph G1" /></td>
<td><img src="chart2" alt="Graph RG1" /></td>
</tr>
<tr>
<td>2 [y = x \cdot r \cdot d]</td>
<td><img src="chart3" alt="Graph G4" /></td>
<td><img src="chart4" alt="Graph RG2" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Union</th>
<th>Path Removal</th>
<th>Factorisation</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>[g_3 \cup g_4 = g_4]</td>
<td>[g_6 \cup {x \mapsto l} = g_2]</td>
<td>[g_2/{x} = {rg_1}]</td>
<td>[(g_3, {l}) # {rg_1} = g_4]</td>
<td></td>
</tr>
<tr>
<td>[g_2 \cup g_4 = g_5]</td>
<td>[g_5 \cup {x} = \mathcal{E}_G]</td>
<td>[g_5/{x} = {rg_1, rg_2}]</td>
<td>[(g_3, {x, l}) # {rg_1, rg_2} = g_6]</td>
<td></td>
</tr>
<tr>
<td>[g_5 \cup g_4 = g_5]</td>
<td>[g_4 \cup {x \mapsto r} = g_4]</td>
<td>[g_5/x \mapsto r = {\epsilon_{RG}}]</td>
<td>[(g_3, {x, l}) # {rg_1, rg_2} = g_6]</td>
<td></td>
</tr>
<tr>
<td>[g_5 \cup g_6 = g_6]</td>
<td>[g_4 \cup {x \mapsto l} = g_1]</td>
<td>[g_4/x \mapsto r = \emptyset]</td>
<td>[(g_2, {r}) # {\epsilon_{RG}} = g_2]</td>
<td></td>
</tr>
</tbody>
</table>
Access Graph Operations: Examples

<table>
<thead>
<tr>
<th>Program</th>
<th>Access Graphs</th>
<th>Remainder Graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 [x = x.l]</td>
<td>[g_1 \Rightarrow x]</td>
<td>[rg_1 \Rightarrow r_2]</td>
</tr>
<tr>
<td>2 [y = x.r.d]</td>
<td>[g_4 \Rightarrow x \rightarrow l, r_2]</td>
<td>[rg_2 \Rightarrow l, r_2]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Union</th>
<th>Path Removal</th>
<th>Factorisation</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>[g_3 \cup g_4 = g_4]</td>
<td>[g_6 \oplus {x \rightarrow l} = g_2]</td>
<td>[g_2/x = {rg_1}]</td>
<td>((g_3, {l}) # {rg_1} = g_4)</td>
</tr>
<tr>
<td>[g_2 \cup g_4 = g_5]</td>
<td>[g_5 \oplus {x} = \mathcal{E}_G]</td>
<td>[g_5/x = {rg_1, rg_2}]</td>
<td>((g_3, {x, l}) # {rg_1, rg_2} = g_6)</td>
</tr>
<tr>
<td>[g_5 \cup g_4 = g_5]</td>
<td>[g_4 \oplus {x \rightarrow r} = g_4]</td>
<td>[g_5/x \rightarrow r = {\epsilon_{RG}}]</td>
<td>((g_3, {x, l}) # {\epsilon_{RG}} = g_6)</td>
</tr>
<tr>
<td>[g_5 \cup g_6 = g_6]</td>
<td>[g_4 \oplus {x \rightarrow l} = g_1]</td>
<td>[g_4/x \rightarrow r = \emptyset]</td>
<td>((g_2, {r}) # \emptyset = \mathcal{E}_G)</td>
</tr>
</tbody>
</table>

Remainder is empty

Quotient is empty
Data Flow Equations for Explicit Liveness Analysis: Access Graphs Version

\[ \text{In}_n = (\text{Out}_n \ominus \text{Kill}_n(\text{Out}_n)) \uplus \text{Gen}_n(\text{Out}_n) \]

\[ \text{Out}_n = \begin{cases} \text{Bl} & \text{if } n \text{ is } \text{End} \\ \uplus \bigcup_{s \in \text{succ}(n)} \text{In}_s & \text{otherwise} \end{cases} \]

- \( \text{In}_n, \text{Out}_n, \) and \( \text{Gen}_n \) are access graphs
- \( \text{Kill}_n \) is a set of access paths
Flow Functions for Explicit Liveness Analysis: Access Paths Version

Let $A$ denote May Aliases at the exit of node $n$

<table>
<thead>
<tr>
<th>Statement $n$</th>
<th>$\text{Gen}_n(X)$</th>
<th>$\text{Kill}_n(X)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = y$</td>
<td>${y \rightarrow \sigma \mid x \rightarrow \sigma \in X}$</td>
<td>$x \rightarrow^*$</td>
</tr>
<tr>
<td>$x = y.f$</td>
<td>${y \rightarrow f \rightarrow \sigma \mid x \rightarrow \sigma \in X}$</td>
<td>$x \rightarrow^*$</td>
</tr>
<tr>
<td>$x.f = y$</td>
<td>${y \rightarrow \sigma \mid z \rightarrow f \rightarrow \sigma \in X, z \in A(x)}$</td>
<td>$\bigcup_{z \in \text{Must}(A)(x)} z \rightarrow f \rightarrow^*$</td>
</tr>
<tr>
<td>$x = \text{new}$</td>
<td>$\emptyset$</td>
<td>$x \rightarrow^*$</td>
</tr>
<tr>
<td>$x = \text{null}$</td>
<td>$\emptyset$</td>
<td>$x \rightarrow^*$</td>
</tr>
<tr>
<td>other</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
**Flow Functions for Explicit Liveness Analysis: Access Paths Version**

Let $A$ denote May Aliases at the exit of node $n$

<table>
<thead>
<tr>
<th>Statement $n$</th>
<th>$\text{Gen}_n(X)$</th>
<th>$\text{Kill}_n(X)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = y$</td>
<td>${y \rightarrow \sigma \mid x \rightarrow \sigma \in X}$</td>
<td>$x \rightarrow \ast$</td>
</tr>
<tr>
<td>$x = y.f$</td>
<td>${y \rightarrow f \rightarrow \sigma \mid x \rightarrow \sigma \in X}$</td>
<td>$x \rightarrow \ast$</td>
</tr>
<tr>
<td>$x.f = y$</td>
<td>${y \rightarrow \sigma \mid z \rightarrow f \rightarrow \sigma \in X, z \in A(x)}$</td>
<td>$\bigcup_{z \in \text{Must}(A)(x)} z \rightarrow f \rightarrow \ast$</td>
</tr>
<tr>
<td>$x = \text{new}$</td>
<td>$\emptyset$</td>
<td>$x \rightarrow \ast$</td>
</tr>
<tr>
<td>$x = \text{null}$</td>
<td>$\emptyset$</td>
<td>$x \rightarrow \ast$</td>
</tr>
<tr>
<td>other</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

May link aliasing for soundness
Flow Functions for Explicit Liveness Analysis: Access Paths Version

Let $A$ denote May Aliases at the exit of node $n$

<table>
<thead>
<tr>
<th>Statement $n$</th>
<th>$\text{Gen}_n(X)$</th>
<th>$\text{Kill}_n(X)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = y$</td>
<td>${y \Rightarrow \sigma \mid x \Rightarrow \sigma \in X}$</td>
<td>$x \Rightarrow \ast$</td>
</tr>
<tr>
<td>$x = y.f$</td>
<td>${y \Rightarrow f \Rightarrow \sigma \mid x \Rightarrow \sigma \in X}$</td>
<td>$x \Rightarrow \ast$</td>
</tr>
<tr>
<td>$x.f = y$</td>
<td>${y \Rightarrow \sigma \mid z \Rightarrow f \Rightarrow \sigma \in X, z \in A(x)}$</td>
<td>$\bigcup_{z \in \text{Must}(A)(x)} z \Rightarrow f \Rightarrow \ast$</td>
</tr>
<tr>
<td>$x = \text{new}$</td>
<td>$\emptyset$</td>
<td>$x \Rightarrow \ast$</td>
</tr>
<tr>
<td>$x = \text{null}$</td>
<td>$\emptyset$</td>
<td>$\text{null} \Rightarrow \ast$</td>
</tr>
<tr>
<td>other</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

May link aliasing for soundness

Must link aliasing for precision
Flow Functions for Explicit Liveness Analysis: Access Graphs Version

- $A$ denotes May Aliases at the exit of node $n$
- $mkGraph(\rho)$ creates an access graph for access path $\rho$

<table>
<thead>
<tr>
<th>Statement $n$</th>
<th>$Gen_n(X)$</th>
<th>$Kill_n(X)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = y$</td>
<td>$mkGraph(y)#(X/x)$</td>
<td>${x}$</td>
</tr>
<tr>
<td>$x = y.f$</td>
<td>$mkGraph(y\rightarrow f)#(X/x)$</td>
<td>${x}$</td>
</tr>
<tr>
<td>$x.f = y$</td>
<td>$mkGraph(y)#\left(\bigcup_{z \in A(x)} (X/(z\rightarrow f))\right)$</td>
<td>${z\rightarrow f \mid z \in Must(A)(x)}$</td>
</tr>
<tr>
<td>$x = new$</td>
<td>$\emptyset$</td>
<td>${x}$</td>
</tr>
<tr>
<td>$x = null$</td>
<td>$\emptyset$</td>
<td>${x}$</td>
</tr>
<tr>
<td>other</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
Liveness Analysis of Example Program: 1st Iteration

1. \( w = x \)

2. while (x.data < max)

3. \( x = x.rptr \)

4. \( y = x.lptr \)

5. \( z = \text{New class of } z \)

6. \( y = y.lptr \)

7. \( z.sum = x.data + y.data \)
Liveness Analysis of Example Program: 2nd Iteration

1. \( w = x \)

2. \( \text{while } (x.\text{data} < \text{max}) \)

3. \( x = x.\text{rptr} \)

4. \( y = x.\text{lptr} \)

5. \( z = \text{New class_of_z} \)

6. \( y = y.\text{lptr} \)

7. \( z.\text{sum} = x.\text{data} + y.\text{data} \)
Liveness Analysis of Example Program: 3rd Iteration

1. \( w = x \)

2. while \((x\.data < max)\)

3. \( x = x\.rptr \)

4. \( y = x\.lptr \)

5. \( z = \text{New class_of_z} \)

6. \( y = y\.lptr \)

7. \( z\.sum = x\.data + y\.data \)
Liveness Analysis of Example Program: 4th Iteration

1. \( w = x \)

2. \( \text{while } (x\text{.data} < \text{max}) \)

3. \( x = x\text{.rptr} \)

4. \( y = x\text{.lptr} \)

5. \( z = \text{New class_of_z} \)

6. \( y = y\text{.lptr} \)

7. \( z\text{.sum} = x\text{.data} + y\text{.data} \)
**Tutorial Problem for Explicit Liveness (1)**

Construct access graphs at the entry of block 1 for the following programs:

**A**

1. \( x = x.n \)
   - 2. \( x = x.n \)
   - 3. Use \( x.r.d \)

**B**

1. \( x = x.n \)
   - 2. \( x = x.n \)
   - 3. Use \( x.r.d \)

**C**

1. \( x = x.n \)
   - 2. \( x = x.n \)
   - 3. Use \( x.r.d \)

**D**

1. \( x = x.n \)
   - 2. \( x = x.n \)
   - 3. Use \( x.r.d \)
   - 4. 

**E**

1. \( x = x.n \)
   - 2. \( x = x.n \)
   - 3. Use \( x.r.d \)
   - 4. Use \( x.r.d \)

**F**

1. \( x = x.n \)
   - 2. \( x = x.n \)
   - 3. \( x = x.l \)
   - 4. 
   - 5. Use \( x.r.d \)
Tutorial Problem for Explicit Liveness (1)

Construct access graphs at the entry of block 1 for the following programs:

A

1. $x = x.n$
2. $x = x.n$
3. Use $x.r.d$

B

1. $x = x.n$
2. $x = x.n$
3. Use $x.r.d$

C

1. $x = x.n$
2. $x = x.n$
3. Use $x.r.d$

D

1. $x = x.n$
2. $x = x.n$
3. Use $x.r.d$

E

1. $x = x.n$
2. $x = x.n$
3. Use $x.r.d$

F

1. $x = x.n$
2. $x = x.n$
3. $x = x.l$
4. Use $x.r.d$
5. Use $x.r.d$

Why are the access graphs for programs B and D identical?
Construct access graphs at the entry of block 1 for the following programs:

A
1. $x = x.n$
2. $x = x.n$
3. Use $x.r.d$

B
1. 
2. $x = x.n$

C
1. 
2. $x = x.n$
Use $x.r.d$

D
1. 
2. $x = x.n$
3. Use $x.r.d$

The final magic!!

Rotate each picture anti-clockwise by $90^\circ$ and compare it with its access graph.
Tutorial Problem for Explicit Liveness (1)

Construct access graphs at the entry of block 1 for the following programs

A
1 \( x = x.n \)
2 \( x = x.n \)
3 Use \( x.r.d \)

B
1

C
1
2 \( x = x.n \)
Use \( x.r.d \)

D
1
2 \( x = x.n \)
3 Use \( x.r.d \)

The final magic!!

Rotate each picture anti-clockwise by 90° and compare it with its access graph

The structure of access graph of variable \( x \) is identical to the control flow structure between pointer assignments of \( x \)
Tutorial Problem for Explicit Liveness (2)

- Unfortunately the student who constructed these access graphs forgot to attach statement numbers as subscripts to node labels and has misplaced the programs which gave rise to these graphs.

- Please help her by constructing CFGs for which these access graphs represent explicit liveness at some program point in the CFGs.
Tutorial Problem for Explicit Liveness (3)

- Compute explicit liveness for the program.
- Are the following access paths live at node 1?
  Show the corresponding execution sequence of statements

P1 : y → m → l
P2 : y → l → n → m
P3 : y → l → n → l
P4 : y → n → l → n
Which Access Paths Can be Nullified?

- Consider extensions of accessible paths for nullification.

Let $\rho$ be accessible at $p$ (i.e. available or anticipable) for each reference field $f$ of the object pointed to by $\rho$ if $\rho \rightarrow f$ is not live at $p$ then

$$\text{Insert } \rho \rightarrow f = \text{null at } p \text{ subject to profitability}$$

- For simple access paths, $\rho$ is empty and $f$ is the root variable name.
Which Access Paths Can be Nullified?

Can be safely dereferenced

- Consider extensions of accessible paths for nullification.

Let $\rho$ be accessible at $p$ (i.e. available or anticipable) for each reference field $f$ of the object pointed to by $\rho$

  If $\rho \rightarrow f$ is not live at $p$ then

  Insert $\rho \rightarrow f = \text{null}$ at $p$ subject to profitability

- For simple access paths, $\rho$ is empty and $f$ is the root variable name.
Which Access Paths Can be Nullified?

- Consider extensions of accessible paths for nullification.

Let $\rho$ be accessible at $p$ (i.e., available or anticipable) for each reference field $f$ of the object pointed to by $\rho$ if $\rho \rightarrow f$ is not live at $p$ then

Insert $\rho \rightarrow f = \text{null}$ at $p$ subject to profitability

- For simple access paths, $\rho$ is empty and $f$ is the root variable name.
Which Access Paths Can be Nullified?

- Consider extensions of accessible paths for nullification.

Let $\rho$ be accessible at $p$ (i.e. available or anticipable) for each reference field $f$ of the object pointed to by $\rho$.

If $\rho \rightarrow f$ is not live at $p$ then

Insert $\rho \rightarrow f = \text{null}$ at $p$ subject to profitability.

- For simple access paths, $\rho$ is empty and $f$ is the root variable name.

Cannot be hoisted and is not redefined at $p$. 

Can be safely dereferenced

Consider link aliases at $p$.
Availability and Anticipability Analyses

- $\rho$ is available at program point $p$ if the target of each prefix of $\rho$ is guaranteed to be created along every control flow path reaching $p$.  

- $\rho$ is anticipable at program point $p$ if the target of each prefix of $\rho$ is guaranteed to be dereferenced along every control flow path starting at $p$.  

Availability and Anticipability Analyses

• \( \rho \) is \textbf{available} at program point \( p \) if the target of each prefix of \( \rho \) is guaranteed to be created along every control flow path reaching \( p \).

• \( \rho \) is \textbf{anticipable} at program point \( p \) if the target of each prefix of \( \rho \) is guaranteed to be dereferenced along every control flow path starting at \( p \).

• Finiteness.
  
  ▶ An anticipable (available) access path must be anticipable (available) along every paths. Thus unbounded paths arising out of loops cannot be anticipable (available).
  
  ▶ Due to “every control flow path nature”, computation of anticipable and available access paths uses \( \cap \) as the confluence. Thus the sets are bounded.

  \( \Rightarrow \) No need of access graphs.
Availability Analysis of Example Program

1. \( w = x \)
2. \( \text{while } (x.\text{data} < \text{max}) \)
3. \( x = x.\text{rptr} \)
4. \( y = x.\text{lptr} \)
5. \( z = \text{New class_of_z} \)
6. \( y = y.\text{lptr} \)
7. \( z.\text{sum} = x.\text{data} + y.\text{data} \)
Anticipability Analysis of Example Program

1. `w = x
   `{x}`

2. `while (x.data < max)
   `{x}`

3. `x = x.rptr
   `{x, x->rptr }`

4. `y = x.lptr
   `{x, y, y->lptr }`

5. `z = New class_of_x
   `{x, y, y->lptr, z}`

6. `y = y.lptr
   `{x, y, z}`

7. `z.sum = x.data + y.data`
   `∅`
Live and Accessible Paths

1. \( w = x \)

2. while \( (x.data < \text{max}) \)

3. \( x = x.rptr \)

4. \( y = x.lptr \)

5. \( z = \text{New class_of_z} \)

6. \( y = y.lptr \)

7. \( z.sum = x.data + y.data \)
Creating null Assignments from Live and Accessible Paths

1. \( w = x \)
   \( w = \text{null} \)

2. \( \text{while (} x \text{.data < max) } \)
   \( x \text{.rptr} = x \text{.lptr.rptr} = \text{null} \)
   \( x \text{.lptr.lptr.lptr} = x \text{.lptr.lptr.rptr} = \text{null} \)

3. \( x = x \text{.rptr} \)
   \( x \text{.lptr} = \text{null} \)

4. \( y = x \text{.lptr} \)
   \( x \text{.lptr} = y \text{.rptr} = \text{null} \)
   \( y \text{.lptr.lptr} = y \text{.lptr.rptr} = \text{null} \)

5. \( z = \text{New class of } z \)
   \( z \text{.lptr} = z \text{.rptr} = \text{null} \)

6. \( y = y \text{.lptr} \)
   \( y \text{.lptr} = y \text{.rptr} = \text{null} \)

7. \( z \text{.sum} = x \text{.data} + y \text{.data} \)
   \( x = y = z = \text{null} \)
The Resulting Program

1. \( w = x \)
2. \( w = \text{null} \)
3. \( \text{while} \ (x.\text{data} < \text{max}) \)
   \[ \{ \]
   \( x.\text{lptr} = \text{null} \)
   \( x = x.\text{rptr} \)
   \[ \} \]
4. \( y = x.\text{lptr} \)
5. \( z = \text{New class of z} \)
6. \( y = y.\text{lptr} \)
7. \( z.\text{sum} = x.\text{data} + y.\text{data} \)
8. \( x = y = z = \text{null} \)
Overapproximation Caused by Our Summarization

- The program allocates $x \rightarrow p$ in one iteration and uses it in the next
Overapproximation Caused by Our Summarization

- The program allocates $x \rightarrow p$ in one iteration and uses it in the next.
Overapproximation Caused by Our Summarization

1. $t = x.p$
   - $t \Rightarrow x \Rightarrow p_1 \Rightarrow p_3$

2. $t.p = \text{new}$
   - $t \Rightarrow x \Rightarrow p_3 \Rightarrow p_1$
   - $x \Rightarrow p_3 \Rightarrow p_1$

3. $x = x.p$
   - $x \Rightarrow p_3 \Rightarrow p_1$

4. $x...$
   - $x \Rightarrow p_3 \Rightarrow p_1$

- The program allocates $x \rightarrow p$ in one iteration and uses it in the next.
- Only $x \rightarrow p \rightarrow p$ is live at Out$_2$.
Overapproximation Caused by Our Summarization

1. $t = x.p$
   - $x \rightarrow p_1$
   - $x \rightarrow p_3 \rightarrow t$

2. $t.p = \text{new}$
   - $x \rightarrow p_3 \rightarrow p_1$

3. $x = x.p$
   - $x \rightarrow p_3$
   - $x \rightarrow p_1$

4. $x \ldots$

Out$_1$

- The program allocates $x \rightarrow p$ in one iteration and uses it in the next
- **Only** $x \rightarrow p \rightarrow p$ is live at Out$_2$
Overapproximation Caused by Our Summarization

1. \( t = x.p \)

2. \( t.p = \text{new} \)

3. \( x = x.p \)

4. \( x \ldots \)

- The program allocates \( x \rightarrow p \) in one iteration and uses it in the next.
- Only \( x \rightarrow p \rightarrow p \) is live at \( \text{Out}_2 \).
- \( x \rightarrow p \rightarrow p \) is live at \( \text{Out}_2 \).
- \( x \rightarrow p \rightarrow p \rightarrow p \) is dead at \( \text{Out}_2 \).
- First \( p \) used in statement 3.
- Second \( p \) used in statement 4.
- Third \( p \) is reallocated.
Overapproximation Caused by Our Summarization

- The program allocates $x \rightarrow p$ in one iteration and uses it in the next
- Only $x \rightarrow p \rightarrow p$ is live at $Out_2$
- $x \rightarrow p \rightarrow p$ is live at $Out_2$
- $x \rightarrow p \rightarrow p \rightarrow p$ is dead at $Out_2$
- First $p$ used in statement 3
- Second $p$ used in statement 4
- Third $p$ is reallocated
Non-Distributivity of Explicit Liveness Analysis

1. $x.n = \text{null}$
2. $x = x.n$
3. $x.n.n = \text{null}$
4. $x = x.r$
5. $x.n.r = \text{null}$
6. $x = x.n$
7. use $x.n.d$
8. use $x.r.d$
Non-Distributivity of Explicit Liveness Analysis

1. $x.n = \text{null}$

2. $x = x.n$

3. $x.n.n = \text{null}$

4. $x = x.r$

5. $x.n.r = \text{null}$

6. $x = x.n$

7. use $x.n.d$

8. use $x.r.d$
Non-Distributivity of Explicit Liveness Analysis

1. \( x.n = \text{null} \)

2. \( x = x.n \)

3. \( x.n.n = \text{null} \)

4. \( x = x.r \)

5. \( x.n.r = \text{null} \)

6. \( x = x.n \)

7. use \( x.n.d \)

8. use \( x.r.d \)

Sep 2017
Non-Distributivity of Explicit Liveness Analysis

1. $x.n = \text{null}$

2. $x = x.n$

3. $x.n.n = \text{null}$

4. $x = x.r$

5. $x.n.r = \text{null}$

6. $x = x.n$

7. use $x.n.d$

8. use $x.r.d$
Non-Distributivity of Explicit Liveness Analysis

1. \( x.n = \text{null} \)

2. \( x = x.n \)
   - \( n_3 \)
   - \( x \)
   - \( n_6 \) \( r_8 \)

3. \( x.n.n = \text{null} \)

4. \( x = x.r \)

5. \( x.n.r = \text{null} \)
   - \( r_8 \)
   - \( x \)
   - \( n_6 \)
   - \( n_7 \)

6. \( x = x.n \)
   - \( x \)
   - \( n_7 \)

7. \( \text{use } x.n.d \)

8. \( \text{use } x.r.d \)

Sep 2017 IIT Bombay
Non-Distributivity of Explicit Liveness Analysis

1. \( x.n = \text{null} \)

2. \( x = x.n \)

3. \( x.n.n = \text{null} \)

4. \( x = x.r \)

5. \( x.n.r = \text{null} \)

6. \( x = x.n \)

7. \( \text{use } x.n.d \)

8. \( \text{use } x.r.d \)

\( x \rightarrow n_7 \)

\( x \rightarrow r_8 \)

\( x \rightarrow n_6 \rightarrow r_8 \)

\( x \rightarrow n_6 \rightarrow n_7 \)

\( x \rightarrow n_5 \rightarrow n_6 \rightarrow n_7 \)

\( x \rightarrow n_5 \rightarrow n_6 \rightarrow n_7 \)

\( x \rightarrow n_5 \rightarrow n_6 \rightarrow n_7 \)

\( x \rightarrow n_5 \rightarrow n_6 \rightarrow n_7 \)

\( x \rightarrow n_5 \rightarrow n_6 \rightarrow n_7 \)

\( x \rightarrow n_5 \rightarrow n_6 \rightarrow n_7 \)
Non-Distributivity of Explicit Liveness Analysis

1. \( x.n = \text{null} \)
2. \( x = x.n \)
3. \( x.n.n = \text{null} \)
4. \( x = x.r \)
5. \( x.n.r = \text{null} \)
6. \( x = x.n \)
7. use \( x.n.d \)
8. use \( x.r.d \)

\( x \rightarrow n_2 \)
\( n_3 \rightarrow n_2 \)
\( n_3 \rightarrow n_6 \)
\( n_6 \rightarrow r_8 \)
\( x \rightarrow n_6 \)
\( r_8 \rightarrow n_6 \)
\( n_6 \rightarrow r_8 \)
\( x \rightarrow n_7 \)
\( n_7 \rightarrow n_6 \)
\( n_7 \rightarrow n_3 \)
\( x \rightarrow r_8 \)
\( r_8 \rightarrow n_7 \)
\( n_7 \rightarrow n_3 \)

Sep 2017 IIT Bombay
Non-Distributivity of Explicit Liveness Analysis

1. $x.n = \text{null}$
2. $x = x.n$
3. $x.n.n = \text{null}$
4. $x = x.r$
5. $x.n.r = \text{null}$
6. $x = x.n$
7. use $x.n.d$
8. use $x.r.d$

Sep 2017
Non-Distributivity of Explicit Liveness Analysis

1. $x.n = \text{null}$
2. $x = x.n$
3. $x.n.n = \text{null}$
4. $x = x.r$
5. $x.n.r = \text{null}$
6. $x = x.n$
7. use $x.n.d$
8. use $x.r.d$
Non-Distributivity of Explicit Liveness Analysis

1. $x.n = \text{null}$

2. $x = x.n$

3. $x.n.n = \text{null}$

4. $x = x.n$

5. $x.n.r = \text{null}$

6. $x = x.n$

7. Use $x.n.d$

8. Use $x.r.d$

Remove $x \rightarrow n \rightarrow *$ due to the assignment in node 1

$f_1(\text{Out}_2 \cup \text{Out}_4)$
Non-Distributivity of Explicit Liveness Analysis

1. $x.n = \text{null}$

2. $x = x.n$

3. $x.n.n = \text{null}$

4. $x.n.r = \text{null}$

5. $x.n.r = \text{null}$

6. $x = x.n$

7. use $x.n.d$

8. use $x.r.d$

Remove $x \rightarrow n \rightarrow *$ due to the assignment in node 1.

$f_1(In_2 \cup In_4)$
Non-Distributivity of Explicit Liveness Analysis

1. \( x.n = \text{null} \)
2. \( x = x.n \)
3. \( x.n.n = \text{null} \)
4. \( x.n.r = \text{null} \)
5. \( x.n.n = \text{null} \)
6. \( x = x.n \)
7. use \( x.n.d \)
8. use \( x.r.d \)

Remove \( x \rightarrow n \rightarrow * \) due to the assignment in node 1.

\[ f_1(\text{In}_2 \cup \text{In}_4) \]

\[ f_1(\text{In}_2) \cup f_1(\text{In}_4) \]
Non-Distributivity of Explicit Liveness Analysis

1. $x.n = \text{null}$

2. $x = x.n$

3. $x.n.n = \text{null}$

4. $x.n.r = \text{null}$

5. $x.n.r = \text{null}$

6. $x = x.n$

7. use $x.n.d$

8. use $x.r.d$

remove $x \rightarrow n \rightarrow \ast$ due to the assignment in node 1

$f_1(\text{ln}_2 \cup \text{ln}_4)$

$f_1(\text{ln}_2) \cup f_1(\text{ln}_4)$

Out$_1$

Sep 2017 IIT Bombay
Non-Distributivity of Explicit Liveness Analysis

1. \( x.n = \text{null} \)

2. \( x = x.n \)

3. \( x.n.n = \text{null} \)

4. \( x.n.r = \text{null} \)

5. \( f_1(Ln_2 \cup Ln_4) \sqsubset f_1(Ln_2) \cup f_1(Ln_4) \)

Access path \( x \rightarrow r \rightarrow n \rightarrow r \) (shown in blue color) is a spurious access path that arises due to \( \cup \) and is not removed by the assignment in node 1.

7. \( \text{use } x.n.d \)

8. \( \text{use } x.r.d \)
Issues Not Covered

- Precision of information
  - Cyclic Data Structures
  - Eliminating Redundant null Assignments

- Properties of Data Flow Analysis:
  Monotonicity, Boundedness, Complexity

- Interprocedural Analysis

- Extensions for C/C++

- Formulation for functional languages

- Issues that need to be researched: Good alias analysis of heap
BTW, What is Static Analysis of Heap?
BTW, What is Static Analysis of Heap?

Abstract, Bounded, Single Instance

Concrete, Unbounded, Infinitely Many

Static

Program Code

Dynamic

Program Execution
BTW, What is Static Analysis of Heap?

- Static: Abstract, Bounded, Single Instance
- Dynamic: Concrete, Unbounded, Infinitely Many

Static: Program Code

Dynamic: Program Execution → Heap Memory
BTW, What is Static Analysis of Heap?

Abstract, Bounded, Single Instance

Concrete, Unbounded, Infinitely Many

Static

Program Code

Summary Heap Data

Dynamic

Program Execution

Heap Memory

Heap Memory

Heap Memory
BTW, What is Static Analysis of Heap?

Abstract, Bounded, Single Instance

Concrete, Unbounded, Infinitely Many

Static

Program Code

Summary Heap Data

Dynamic

Profiling

Program Execution

Heap Memory
BTW, What is Static Analysis of Heap?

Abstract, Bounded, Single Instance

Concrete, Unbounded, Infinitely Many

Static

Program Code → Static Analysis → Summary Heap Data

Dynamic

Program Execution → Heap Memory

Summary

Heap Data
Conclusions

• Unbounded information can be summarized using interesting insights

  ▶ Contrary to popular perception, heap structure is not arbitrary

  *Heap manipulations consist of repeating patterns which bear a close resemblance to program structure*

  Analysis of heap data is possible despite the fact that the mappings between access expressions and l-values keep changing