Pointer Analysis

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

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

Dec 2019
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.*
An Outline of Pointer Analysis Coverage

- The larger perspective
- IR for Points-to Analysis
- Flow-Insensitive Points-to Analysis
- Flow-Sensitive Points-to Analysis
Code Optimization In Presence of Pointers (1)

**Program**

1. q = p;
2. while (…) {
3.     q = q→next;
4. }
5. p→data = r1;
6. print (q→data);
7. p→data = r2;

**Memory graph at statement 5**

![Memory graph diagram]

- Is p→data live at the exit of line 5? Can we delete line 5?
Code Optimization In Presence of Pointers (1)

Program

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

Memory graph at statement 5

• Is p→data live at the exit of line 5? Can we delete line 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 (1)

**Program**

1. `q = p;`
2. `do {
   3.     q = q→next;
   4. } while (…)
5. `p→data = r1;`
6. `print (q→data);`
7. `p→data = r2;`

---

**Memory graph at statement 5**

- Is `p→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 (2)

Original Program

\[
\begin{align*}
    a &= 5 \\
    x &= \&a \\
    b &= *x
\end{align*}
\]
Code Optimization In Presence of Pointers (2)

Original Program  Constant Propagation
without aliasing

\( a = 5 \)
\( x = &a \)
\( b = *x \)

\( a = 5 \)
\( x = &a \)
\( b = *x \)
Code Optimization In Presence of Pointers (2)

Original Program  \[ a = 5 \]
\[ x = \&a \]
\[ b = *x \]

Constant Propagation without aliasing  \[ a = 5 \]
\[ x = \&a \]
\[ b = *x \]

Constant Propagation with aliasing  \[ a = 5 \]
\[ x = \&a \]
\[ b = 5 \]
Code Optimization In Presence of Pointers (3)

\begin{align*}
\text{main} & \quad a = 5 \\
\text{f} & \quad p = g; \\
\text{main} & \quad f(); \\
\text{main} & \quad p(); \\
\text{main} & \quad b = *x \\
\end{align*}

\begin{align*}
\text{g} & \quad x = &a; \\
\text{h} & \quad x = &c; \\
\end{align*}
Code Optimization In Presence of Pointers (3)

```
main

a = 5

f();

p();

b = *x

f

p = g;

f();

p();

b = *x

Dec 2019 IIT Bombay
```
Code Optimization In Presence of Pointers (3)

```
f
p = g;

main
a = 5
f();
p();

b = *x

f

Dec 2019 IIT Bombay
```
Code Optimization In Presence of Pointers (3)

```
f
  p = g;
  f();
  p();
  b = 5

main
  a = 5
  f();
  p();
  b = 5

h

Dec 2019 IIT Bombay
```
Pointer Analysis

• Answers the following questions for indirect accesses:
  ▶ Which data is read? \( x = *y \)
  ▶ Which data is written? \( *x = y \)
  ▶ Which procedure is called? \( p() \) or \( x \to f() \)

• Enables precise data flow and interprocedural control flow analysis

• Computationally intensive analyses are ineffective when supplied with imprecise points-to information,
  (e.g., model checking, interprocedural analyses)

• Needs to scale to large programs
The World of Pointer Analysis

Alias Analysis

- Alias analysis of reference parameters, fields of unions, array indices
- Alias analysis of data pointers

Pointer Analysis

- Points-to analysis of data and function pointers
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
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
  ◦ 2019 😞
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?
So what should we expect? To quote Hind [PASTE 2001]
The Engineering of Pointer Analysis

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: Precision versus Scalability

• Ideally, an analysis should be
  ○ Sound
  ○ Precise
  ○ Scalable
**Pointer Analysis: Precision versus Scalability**

- Ideally, an analysis should be
  - Sound
  - Precise
  - Scalable

**Common belief**
- Precision and scalability cannot be achieved together for exhaustive analysis

**Common Practice**
- Trade off precision using approximations
• Ideally, an analysis should be
  ○ Sound
  ○ Precise
  ○ Scalable

• The main factors enhancing the precision of an exhaustive (as against a demand-driven) analysis are
  ○ Flow sensitivity
  ○ Context sensitivity
  ○ Field sensitivity
Demand-Driven Analysis Vs. Exhaustive Analysis

- **Exhaustive.** Compute all possible information
- **Demand-Driven.** Compute only the requested information (by a client)

Different from incremental analysis which also computes only some information but it updates the earlier computed solution
Flow Sensitivity Vs. Flow Insensitivity

Flow Sensitive

Flow Insensitive

Start

End
Flow Sensitivity Vs. Flow Insensitivity

**Flow Sensitive**

Start

0 \( f_0 \)

1 \( f_1 \)

2 \( f_2 \) 

3 \( f_3 \)

\( \cdots \)

\( \vdots \)

\( i \) \( f_i \)

\( \cdots \)

\( m \) \( f_m \)

End

**Flow Insensitive**

Assumption: Statements can be executed in any order
Flow Sensitivity Vs. Flow Insensitivity

Flow Sensitive

0 \( f_0 \)

1 \( f_1 \)

2 \( f_2 \)

3 \( f_3 \)

\( i \) \( f_i \)

\( m \) \( f_m \)

Flow Insensitive

Start

0 \( f_0 \)

1 \( f_1 \)

2 \( f_2 \)

3 \( f_3 \)

\( i \) \( f_i \)

\( m \) \( f_m \)

End

Dec 2019
Flow Sensitivity Vs. Flow Insensitivity

Flow-insensitive analysis is less precise than a flow-sensitive analysis

Flow-insensitive points-to information
\{ x \rightarrow a, x \rightarrow b, \\
y \rightarrow a, y \rightarrow b \}\n
Flow-sensitive points-to information
\{ x \rightarrow b, \\
y \rightarrow a \}\n
Start
\[ x = \& a; \]

\[ y = x; \]

\[ x = \& b; \]

End

Dec 2019 IIT Bombay
Context Sensitivity Vs. Context Insensitivity

\[ a = \& b \]
\[ c_i \]
\[ C_i \]
\[ R_i \]
\[ End_s \]

\[ c = \& d \]
\[ c_j \]
\[ C_j \]
\[ R_j \]
\[ End_t \]

Dec 2019 IIT Bombay
Context Sensitivity Vs. Context Insensitivity

\[ a = \& b \]
\[ c_i \]
\[ C_i \]
\[ R_i \]
\[ End_s \]
\[ Start_s \]
\[ c_j \]
\[ C_j \]
\[ R_j \]
\[ End_t \]
\[ Start_t \]

Dec 2019
Context Sensitivity Vs. Context Insensitivity

\[ \text{Start}_s \]
\[ a = \& b \]
\[ c_i \]
\[ C_i \]
\[ R_i \]
\[ \text{End}_s \]

\[ \text{Start}_r \]
\[ \text{End}_r \]
\[ f_r \]
\[ c_j \]
\[ C_j \]
\[ R_j \]

\[ \text{Start}_t \]
\[ c = \& d \]
\[ c_j \]

Dec 2019 IIT Bombay
Context Sensitivity Vs. Context Insensitivity

Start_s

\[ a = \& b \]

C_i

R_i

End_s

Start_r

\[ f_r \]

End_r

Start_t

\[ c = \& d \]

C_j

R_j

End_t
Context Sensitivity Vs. Context Insensitivity

\[ \text{Start}_s \]
\[ a = \& b \]
\[ C_i \]
\[ R_i \]
\[ \text{End}_s \]

\[ \text{Start}_r \]
\[ \text{End}_r \]
\[ f_r \]

\[ \text{Start}_t \]
\[ c = \& d \]
\[ C_j \]
\[ R_j \]
\[ \text{End}_t \]

Dec 2019 IIT Bombay
Context Sensitivity Vs. Context Insensitivity
Context Sensitivity Vs. Context Insensitivity

\[
\begin{align*}
\text{Start}_s & \quad a = \& b \\
& \quad C_i \\
& \quad R_i \\
& \quad \text{End}_s \\
& \quad c_i \\
\text{Start}_r & \quad \text{End}_r \\
& \quad f_r \\
\text{Start}_t & \quad c = \& d \\
& \quad C_j \\
& \quad R_j \\
& \quad \text{End}_t \\
& \quad c_j
\end{align*}
\]
Context Sensitivity Vs. Context Insensitivity

\[ a = \& b \]

\[ c_i \]

\[ r \]

\[ C_i \]

\[ R_i \]

\[ End_s \]

\[ f_r \]

\[ c_j \]

\[ f_r \]

\[ C_j \]

\[ R_j \]

\[ End_t \]
Context Sensitivity Vs. Context Insensitivity

\[ \text{Start}_s \rightarrow a = \& b \rightarrow C_i \rightarrow R_i \rightarrow \text{End}_s \]

\[ \text{Start}_r \rightarrow c = \& d \rightarrow C_j \rightarrow R_j \rightarrow \text{End}_r \]

Dec 2019 IIT Bombay
Context Sensitivity Vs. Context Insensitivity

Start_s

a = &b

C_i

R_i

End_s

Start_t

c = &d

C_j

R_j

End_t

Start_r

f_r

a → b → c → d

Dec 2019 IIT Bombay
Context Sensitivity Vs. Context Insensitivity

Context-insensitive analysis is less precise than a context-sensitive analysis.
## Field Sensitivity Vs. Field Insensitivity

<table>
<thead>
<tr>
<th>Program</th>
<th>Field-sensitive points-to graph</th>
<th>Field-insensitive points-to graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x \rightarrow f = &amp; y )</td>
<td><img src="image" alt="Field-sensitive graph" /></td>
<td></td>
</tr>
<tr>
<td>( x \rightarrow g = &amp; z )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( w = x \rightarrow f )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Field Sensitivity Vs. Field Insensitivity

<table>
<thead>
<tr>
<th>Program</th>
<th>Field-sensitive points-to graph</th>
<th>Field-insensitive points-to graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \rightarrow f = &amp;y$</td>
<td>![Field-sensitive graph]</td>
<td>![Field-insensitive graph]</td>
</tr>
<tr>
<td>$x \rightarrow g = &amp;z$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$w = x \rightarrow f$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Field Sensitivity Vs. Field Insensitivity

<table>
<thead>
<tr>
<th>Program</th>
<th>Field-sensitive points-to graph</th>
<th>Field-insensitive points-to graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \rightarrow f = &amp; y$</td>
<td><img src="image" alt="Field-sensitive graph" /></td>
<td><img src="image" alt="Field-insensitive graph" /></td>
</tr>
<tr>
<td>$x \rightarrow g = &amp; z$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$w = x \rightarrow f$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Field-insensitive analysis is less precise than a field-sensitive analysis.
Pointer analysis is a fertile ground for research because the factors that enhance the precision of points-to analysis (flow, context, and field sensitivity), hamper scalability.
Pointer analysis is a fertile ground for research because the factors that enhance the precision of points-to analysis (flow, context, and field sensitivity), hamper scalability.
Pointer Analysis is a fertile ground for research because the factors that enhance the precision of points-to analysis (flow, context, and field sensitivity), hamper scalability.

Data structures: BDDs, probabilistic

Methods: Parallel, demand, randomized

Flow Sensitivity Increases

Context Sensitivity Increases

FI ⊆ FI_{SSA} ⊆ FS ≤ FS\text{NoKill}
Pointer analysis is a fertile ground for research because the factors that enhance the precision of points-to analysis (flow, context, and field sensitivity), hamper scalability.

Data structures: BDDs, probabilistic

Methods: Parallel, demand, randomized

Refinement: Level-wise, bootstrapping

Flow Sensitivity Increases

Context Sensitivity Increases
Pointer analysis is a fertile ground for research because the factors that enhance the precision of points-to analysis (flow, context, and field sensitivity), hamper scalability.
Pointer analysis is a fertile ground for research because the factors that enhance the precision of points-to analysis (flow, context, and field sensitivity), hamper scalability.

- Refinement: Level-wise, bootstrapping
- Methods: Parallel, demand, randomized
- Data structures: BDDs, probabilistic

That's the corner we are trying to occupy :)
An Outline of Pointer Analysis Coverage

- The larger perspective
- IR for Points-to Analysis  
  Next Topic
- Flow-Insensitive Points-to Analysis
- Flow-Sensitive Points-to Analysis
### Pointer Statements

<table>
<thead>
<tr>
<th>Pointer assignments</th>
<th>Use pointers in expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addr</td>
<td>$x = &amp;y$</td>
</tr>
<tr>
<td>Copy</td>
<td>$x = y$</td>
</tr>
<tr>
<td>Load</td>
<td>$x = *y$</td>
</tr>
<tr>
<td></td>
<td>$x = y \rightarrow n$</td>
</tr>
<tr>
<td>Store</td>
<td>$*x = y$</td>
</tr>
<tr>
<td></td>
<td>$x \rightarrow n = y$</td>
</tr>
</tbody>
</table>

- Field accesses such as $x.n$ are treated as new compile time names
- Containment of $x.n$ within $x$ is recorded in terms of offsets
- Heap will be introduced later
What Does a Use Statement Represent? (1)

Consider the declaration: int a, *x, **y;

<table>
<thead>
<tr>
<th>Source</th>
<th>3-Address representation</th>
<th>Our modelling</th>
</tr>
</thead>
<tbody>
<tr>
<td>*x = a</td>
<td>*x = a</td>
<td>Use x</td>
</tr>
<tr>
<td>a = *x</td>
<td>a = *x</td>
<td>Use x</td>
</tr>
<tr>
<td>if (x == NULL)</td>
<td>if (x == NULL)</td>
<td>Use x</td>
</tr>
<tr>
<td>if (*x == 5)</td>
<td>if (*x == 5)</td>
<td>Use x</td>
</tr>
<tr>
<td>if (*y == NULL)</td>
<td>t = *y</td>
<td>t = *y</td>
</tr>
<tr>
<td></td>
<td>if (t == NULL)</td>
<td>Use t</td>
</tr>
<tr>
<td>(** y = a)</td>
<td>t = *y</td>
<td>t = *y</td>
</tr>
<tr>
<td></td>
<td>*t = a</td>
<td>Use t</td>
</tr>
</tbody>
</table>

We retain only the pointers
What Does a Use Statement Represent? (2)

Consider the declaration:

```c
struct s {
    struct s *n;
    int m;
} a, b, *x;
```

<table>
<thead>
<tr>
<th>Source</th>
<th>3-Address representation</th>
<th>Our modelling</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a.n = &amp;b</code></td>
<td><code>a.n = &amp;b</code></td>
<td><code>a.n = &amp;b</code></td>
</tr>
<tr>
<td>if (<code>x \rightarrow n == NULL</code>)</td>
<td><code>t = x \rightarrow n</code> if (<code>t == NULL</code>)</td>
<td><code>t = x \rightarrow n</code> Use <code>t</code></td>
</tr>
<tr>
<td>if (<code>a.n == NULL</code>)</td>
<td><code>t = a.n</code> if (<code>t == NULL</code>)</td>
<td><code>t = a.n</code> Use <code>t</code></td>
</tr>
</tbody>
</table>

*We retain only the pointers*
An Outline of Pointer Analysis Coverage

- The larger perspective
- IR for Points-to Analysis
- **Flow-Insensitive Points-to Analysis**
- Flow-Sensitive Points-to Analysis
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.

_The control flow graph is a complete graph (except for the Start and End nodes)_
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?
Notation for Andersen’s and Steensgaard’s Points-to Analysis

- $P_{x.f}$ denotes the set of pointees of pointer variable $x$ along field $f$
  - $P_{x.*}$ (concisely written as $P_x$) denotes the set of pointees of $x$
  - If $x$ is a structure, $P_x$ is the set of pointees of all fields of $x$

- $\text{Unify}(x, y)$ 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

- We use $x.f$ if the pointees of field $f$ of $x$ are to be 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} ) ( \forall z \in P_x. \text{Unify}(y, z) )</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>( \forall z \in P_x. P_z \supseteq P_y )</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>
</table>
| $x = \& y$ | $P_x \supseteq \{y\}$ | $P_x \supseteq \{y\}$  
$\forall z \in P_x. \ Unify(y, z)$ |
| $x = y$   | $P_x \supseteq P_y$      | $UnifyPTS(x, y)$            |
| $x = *y$  | $P_x \supseteq P_z. \forall z \in P_y$ | $\forall z \in P_y. \ UnifyPTS(x, z)$ |
| $*x = y$  | $\forall z \in P_x. P_z \supseteq P_y$ | $\forall z \in P_x. \ UnifyPTS(y, z)$ |

Points-to graph before the assignment:

- **Andersen’s Points-to Sets**:
  - $x$ to $p$
  - $p$ to $q$
  - $q$ to $r$

- **Steensgaard’s Points-to Sets**:
  - $x$ to $p$
  - $p$ to $q$
  - $q$ to $r$

- **Statement**: $x = \& y$
- **Points-to graph before the assignment**:
  - $x$ to $p$
  - $p$ to $q$
  - $q$ to $r$

- **Statement**: $x = y$
- **Points-to graph before the assignment**:
  - $x$ to $p$
  - $p$ to $q$
  - $q$ to $r$

- **Statement**: $x = *y$
- **Points-to graph before the assignment**:
  - $x$ to $p$
  - $p$ to $q$
  - $q$ to $r$

- **Statement**: $*x = y$
- **Points-to graph before the assignment**:
  - $x$ to $p$
  - $p$ to $q$
  - $q$ to $r$
## 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>$P_x \supseteq {y}$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>$P_x \supseteq P_y$</td>
<td>$\forall z \in P_x. \text{Unify}(y, z)$</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>$\forall z \in P_x. P_z \supseteq P_y$</td>
<td>$\forall z \in P_x. \text{UnifyPTS}(y, z)$</td>
</tr>
</tbody>
</table>

Points-to graph before the assignment

Andersen’s graph after the assignment

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>$P_x \supseteq {y}$, $\forall z \in P_x. \text{Unify}(y, z)$</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>$\forall z \in P_x. P_z \supseteq P_y$</td>
<td>$\forall z \in P_x. \text{UnifyPTS}(y, z)$</td>
</tr>
</tbody>
</table>

Points-to graph before the assignment

Andersen’s graph after the assignment

Steensgaard’s graph after the assignment

Inclusion

Equality
### 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}) (\forall z \in P_x. \text{Unify}(y, z))</td>
</tr>
<tr>
<td>(\boxed{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>(\forall z \in P_x. P_z \supseteq P_y)</td>
<td>(\forall z \in P_x. \text{UnifyPTS}(y, z))</td>
</tr>
</tbody>
</table>

Points-to graph before the assignment:

- Red points: \(x, p, q, r\)
- Blue points: \(y, a, b, c\)
## 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>$P_x \supseteq {y}$</td>
</tr>
<tr>
<td>x = y</td>
<td>$P_x \supseteq P_y$</td>
<td>$\forall z \in P_x. \text{Unify}(y, z)$</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>$\forall z \in P_x. P_z \supseteq P_y$</td>
<td>$\forall z \in P_x. \text{UnifyPTS}(y, z)$</td>
</tr>
</tbody>
</table>

Points-to graph before the assignment

Andersen’s graph after the assignment

Inclusion

Dec 2019
## 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>$P_x \supseteq {y}$ \ $orall z \in P_x. \text{Unify}(y, z)$</td>
</tr>
<tr>
<td>$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$</td>
<td>$\forall z \in P_x. P_z \supseteq P_y$</td>
<td>$\forall z \in P_x. \text{UnifyPTS}(y, z)$</td>
</tr>
</tbody>
</table>

Points-to graph before the assignment

Andersen’s graph after the assignment

Steensgaard’s graph after the assignment
## 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})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\forall z \in P_x. \text{Unify}(y, z))</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>(\forall z \in P_x. P_z \supseteq P_y)</td>
<td>(\forall z \in P_x. \text{UnifyPTS}(y, z))</td>
</tr>
</tbody>
</table>

Points-to graph before the assignment:

- \(x \rightarrow p \rightarrow q \rightarrow r\)
- \(y \rightarrow a \rightarrow b \rightarrow c\)
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>$P_x \supseteq {y}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\forall z \in P_x. \text{Unify}(y, z)$</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>$\forall z \in P_x. P_z \supseteq P_y$</td>
<td>$\forall z \in P_x. \text{UnifyPTS}(y, z)$</td>
</tr>
</tbody>
</table>

Points-to graph before the assignment

Andersen’s graph after the assignment

Inclusion

Dec 2019
**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} ) ( \forall z \in P_x. \text{Unify}(y, z) )</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>( \forall z \in P_x. P_z \supseteq P_y )</td>
<td>( \forall z \in P_x. \text{UnifyPTS}(y, z) )</td>
</tr>
</tbody>
</table>

Points-to graph before the assignment:

Andersen’s graph after the assignment:

Steensgaard’s graph after the assignment:

Dec 2019
## 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>$P_x \supseteq {y}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\forall z \in P_x. \text{Unify}(y, z)$</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>$\forall z \in P_x. P_z \supseteq P_y$</td>
<td>$\forall z \in P_x. \text{UnifyPTS}(y, z)$</td>
</tr>
</tbody>
</table>

Points-to graph before the assignment:

- $x \rightarrow p \rightarrow q \rightarrow r$
- $y \rightarrow a \rightarrow b \rightarrow c$
### 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 } ) ( \forall z \in P_x. \text{Unify}(y, z) )</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 \equiv y)</td>
<td>( \forall z \in P_x. P_z \supseteq P_y )</td>
<td>( \forall z \in P_x. \text{UnifyPTS}(y, z) )</td>
</tr>
</tbody>
</table>

**Points-to graph before the assignment**

\[ x \rightarrow p \rightarrow q \rightarrow r \]

\[ y \rightarrow a \rightarrow b \rightarrow c \]

**Andersen’s graph after the assignment**

\[ x \rightarrow p \rightarrow q \rightarrow r \]

\[ y \rightarrow a \rightarrow b \rightarrow c \]

**Inclusion**

\[ x \rightarrow p \rightarrow q \rightarrow r \]

\[ y \rightarrow a \rightarrow b \rightarrow c \]
# 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>$P_x \supseteq {y}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\forall z \in P_x. \text{Unify}(y, z)$</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>$\forall z \in P_x. P_z \supseteq P_y$</td>
<td>$\forall z \in P_x. \text{UnifyPTS}(y, z)$</td>
</tr>
</tbody>
</table>

Points-to graph before the assignment

Andersen’s graph after the assignment

Steensgaard’s graph after the assignment
## 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>$P_x \supseteq {y}$ $\forall z \in P_x. \text{Unify}(y, z)$</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>$\forall z \in P_x. P_z \supseteq P_y$</td>
<td>$\forall z \in P_x. \text{UnifyPTS}(y, z)$</td>
</tr>
</tbody>
</table>

**Points-to graph before the assignment**

- $x \rightarrow p \rightarrow q \rightarrow r$
- $y \rightarrow a \rightarrow b \rightarrow c$
Example of Inclusion Based (aka Andersen’s) Points-to Analysis

Program

1. \( x = \& a \)
2. \( y = \& b \)
3. \( x \rightarrow n = y \)
4. \( y = x \)
5. \( x \rightarrow n = \& c \)
6. \( x = \& d \)

Type declarations

```c
struct s {
    struct s *n;
    int m;
}
*x, *y, a, b, c, d;
```
Example of Inclusion Based (aka Andersen’s) Points-to Analysis

Program

1. $x = &a$
2. $y = &b$
3. $x \rightarrow n = y$
4. $y = x$
5. $x \rightarrow n = &c$
6. $x = &d$

<table>
<thead>
<tr>
<th>Node</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$P_x \supseteq {a}$</td>
</tr>
<tr>
<td>2</td>
<td>$P_y \supseteq {b}$</td>
</tr>
<tr>
<td>3</td>
<td>$\forall z \in P_x, P_{z,n} \supseteq P_y$</td>
</tr>
<tr>
<td>4</td>
<td>$P_y \supseteq P_x$</td>
</tr>
<tr>
<td>5</td>
<td>$\forall z \in P_x, P_{z,n} \supseteq {c}$</td>
</tr>
<tr>
<td>6</td>
<td>$P_x \supseteq {d}$</td>
</tr>
</tbody>
</table>
Example of Inclusion Based (aka Andersen’s) Points-to Analysis

Program

1. \( x = \&a \)
2. \( y = \&b \)
3. \( x \rightarrow n = y \)
4. \( y = x \)
5. \( x \rightarrow n = \&c \)
6. \( x = \&d \)

Points-to Graph

Node | Constraint
--- | ---
1 | \( P_x \supseteq \{a\} \)
2 | \( P_y \supseteq \{b\} \)
3 | \( \forall z \in P_x, P_{z,n} \supseteq P_y \)
4 | \( P_y \supseteq P_x \)
5 | \( \forall z \in P_x, P_{z,n} \supseteq \{c\} \)
6 | \( P_x \supseteq \{d\} \)
Example of Inclusion Based (aka Andersen’s) Points-to Analysis

Program

1. \( x = \& a \)
2. \( y = \& b \)
3. \( x \rightarrow n = y \)
4. \( y = x \)
5. \( x \rightarrow n = \& c \)
6. \( x = \& d \)

Points-to Graph

Node | Constraint
--- | ---
1 | \( P_x \supseteq \{ a \} \)
2 | \( P_y \supseteq \{ b \} \)
3 | \( \forall z \in P_x, P_{z,n} \supseteq P_y \)
4 | \( P_y \supseteq P_x \)
5 | \( \forall z \in P_x, P_{z,n} \supseteq \{ c \} \)
6 | \( P_x \supseteq \{ d \} \)
Example of Inclusion Based (aka Andersen’s) Points-to Analysis

Program

1. \(x = \& a\)
2. \(y = \& b\)
3. \(x \rightarrow n = y\)
4. \(y = x\)
5. \(x \rightarrow n = \& c\)
6. \(x = \& d\)

Points-to Graph

```
\(x\)
\(a\)
\(y\)
\(n\)
\(b\)
```

Node | Constraint
--- | ---
1 | \(P_x \supseteq \{a\}\)
2 | \(P_y \supseteq \{b\}\)
3 | \(\forall z \in P_x, P_{z,n} \supseteq P_y\)
4 | \(P_y \supseteq P_x\)
5 | \(\forall z \in P_x, P_{z,n} \supseteq \{c\}\)
6 | \(P_x \supseteq \{d\}\)
Example of Inclusion Based (aka Andersen’s) Points-to Analysis

Program

1. \( x = \&a \)
2. \( y = \&b \)
3. \( x \rightarrow n = y \)
4. \( y = x \)
5. \( x \rightarrow n = \&c \)
6. \( x = \&d \)

Points-to Graph

Points-to Graph

<table>
<thead>
<tr>
<th>Node</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( P_x \supseteq {a} )</td>
</tr>
<tr>
<td>2</td>
<td>( P_y \supseteq {b} )</td>
</tr>
<tr>
<td>3</td>
<td>( \forall z \in P_x, P_{z,n} \supseteq P_y )</td>
</tr>
<tr>
<td>4</td>
<td>( P_y \supseteq P_x )</td>
</tr>
<tr>
<td>5</td>
<td>( \forall z \in P_x, P_{z,n} \supseteq {c} )</td>
</tr>
<tr>
<td>6</td>
<td>( P_x \supseteq {d} )</td>
</tr>
</tbody>
</table>
### Example of Inclusion Based (aka Andersen’s) Points-to Analysis

#### Program

1. $x = \&a$
2. $y = \&b$
3. $x \rightarrow n = y$
4. $y = x$
5. $x \rightarrow n = \&c$
6. $x = \&d$

#### Points-to Graph

- $x \rightarrow a$
- $y \rightarrow b$
- $n \rightarrow c$

#### Points-to Analysis

<table>
<thead>
<tr>
<th>Node</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$P_x \supseteq {a}$</td>
</tr>
<tr>
<td>2</td>
<td>$P_y \supseteq {b}$</td>
</tr>
<tr>
<td>3</td>
<td>$\forall z \in P_x, P_z \cup n \supseteq P_y$</td>
</tr>
<tr>
<td>4</td>
<td>$P_y \supseteq P_x$</td>
</tr>
<tr>
<td>5</td>
<td>$\forall z \in P_x, P_z \cup n \supseteq {c}$</td>
</tr>
<tr>
<td>6</td>
<td>$P_x \supseteq {d}$</td>
</tr>
</tbody>
</table>
Example of Inclusion Based (aka Andersen’s) Points-to Analysis

Program

<table>
<thead>
<tr>
<th>Node</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$P_x \supseteq {a}$</td>
</tr>
<tr>
<td>2</td>
<td>$P_y \supseteq {b}$</td>
</tr>
<tr>
<td>3</td>
<td>$\forall z \in P_x, P_{z.n} \supseteq P_y$</td>
</tr>
<tr>
<td>4</td>
<td>$P_y \supseteq P_x$</td>
</tr>
<tr>
<td>5</td>
<td>$\forall z \in P_x, P_{z.n} \supseteq {c}$</td>
</tr>
<tr>
<td>6</td>
<td>$P_x \supseteq {d}$</td>
</tr>
</tbody>
</table>

Points-to Graph

Dec 2019
Example of Inclusion Based (aka Andersen’s) Points-to Analysis

Program

1. \( x = \& a \)
2. \( y = \& b \)
3. \( n = y \rightarrow x \)
4. \( y = x \)
5. \( n = \& c \rightarrow x \)
6. \( x = \& d \)

Points-to Graph

<table>
<thead>
<tr>
<th>Node</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( P_x \supseteq { a } )</td>
</tr>
<tr>
<td>2</td>
<td>( P_y \supseteq { b } )</td>
</tr>
<tr>
<td>3</td>
<td>( \forall z \in P_x, P_{z,n} \supseteq P_y )</td>
</tr>
<tr>
<td>4</td>
<td>( P_y \supseteq P_x )</td>
</tr>
<tr>
<td>5</td>
<td>( \forall z \in P_x, P_{z,n} \supseteq { c } )</td>
</tr>
<tr>
<td>6</td>
<td>( P_x \supseteq { d } )</td>
</tr>
</tbody>
</table>

• Since \( P_x \) has changed, constraints 3, 4, and 5 need to be processed again
• Order of processing the sets influences the efficiency of this fixed point computation significantly
• A plethora of heuristics have been proposed
Example of Inclusion Based (aka Andersen’s) Points-to Analysis

Program

1. \( x = &a \)
2. \( y = &b \)
3. \( x \rightarrow n = y \)
4. \( y = x \)
5. \( x \rightarrow n = &c \)
6. \( x = &d \)

Points-to Graph

<table>
<thead>
<tr>
<th>Node</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( P_x \supseteq { a } )</td>
</tr>
<tr>
<td>2</td>
<td>( P_y \supseteq { b } )</td>
</tr>
<tr>
<td>3</td>
<td>( \forall z \in P_x, P_{z.n} \supseteq P_y )</td>
</tr>
<tr>
<td>4</td>
<td>( P_y \supseteq P_x )</td>
</tr>
<tr>
<td>5</td>
<td>( \forall z \in P_x, P_{z.n} \supseteq { c } )</td>
</tr>
<tr>
<td>6</td>
<td>( P_x \supseteq { d } )</td>
</tr>
</tbody>
</table>

- Since \( P_x \) has changed, constraints 3, 4, and 5 need to be processed again
- Order of processing the sets influences the efficiency of this fixed point computation significantly
- A plethora of heuristics have been proposed
Example of Inclusion Based (aka Andersen’s) Points-to Analysis

Program

1. \( x = \&a \)
2. \( y = \&b \)
3. \( x \rightarrow n = y \)
4. \( y = x \)
5. \( x \rightarrow n = \&c \)
6. \( x = \&d \)

Points-to Graph

Node | Constraint
---|---
1 | \( P_x \supseteq \{a\} \)
2 | \( P_y \supseteq \{b\} \)
3 | \( \forall z \in P_x, P_{z.n} \supseteq P_y \)
4 | \( P_y \supseteq P_x \)
5 | \( \forall z \in P_x, P_{z.n} \supseteq \{c\} \)
6 | \( P_x \supseteq \{d\} \)

- Actual graph after statement 6 (red box on the right) is much simpler with many edges killed.
- \( y \) does not point to \( d \) any time in the execution.

Dec 2019

IIT Bombay
Example of Inclusion Based (aka Andersen’s) Points-to Analysis

Program

1. \( x = \&a \)
2. \( y = \&b \)
3. \( x \rightarrow n = y \)
4. \( y = x \)
5. \( x \rightarrow n = \&c \)
6. \( x = \&d \)

Points-to Graph

- A union of all graphs at each program point
- \( y \) does not point to \( d \) any time in the execution

<table>
<thead>
<tr>
<th>Node</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( P_x \supseteq {a} )</td>
</tr>
<tr>
<td>2</td>
<td>( P_y \supseteq {b} )</td>
</tr>
<tr>
<td>3</td>
<td>( \forall z \in P_x, P_{z.n} \supseteq P_y )</td>
</tr>
<tr>
<td>4</td>
<td>( P_y \supseteq P_x )</td>
</tr>
<tr>
<td>5</td>
<td>( \forall z \in P_x, P_{z.n} \supseteq {c} )</td>
</tr>
<tr>
<td>6</td>
<td>( P_x \supseteq {d} )</td>
</tr>
</tbody>
</table>
Example of Equality Based (aka Steensgaard’s) Points-to Analysis

Program

1. $x = \&a$
2. $y = \&b$
3. $x \rightarrow n = y$
4. $y = x$
5. $x \rightarrow n = \&c$
6. $x = \&d$
Example of Equality Based (aka Steensgaard’s) Points-to Analysis

Program

<table>
<thead>
<tr>
<th>Node</th>
<th>Constraint</th>
</tr>
</thead>
</table>
| 1    | $P_x \supseteq \{a\}$  
     | $\forall z \in P_x, \text{Unify}(a, z)$  |
| 2    | $P_y \supseteq \{b\}$  
     | $\forall z \in P_y, \text{Unify}(b, z)$  |
| 3    | $\forall z \in P_x, \text{UnifyPTS}(y, z.n)$  |
| 4    | $\text{UnifyPTS}(x, y)$  |
| 5    | $\forall z \in P_x, P_{z,n} \supseteq \{c\}$  
     | $\forall w \in P_{z,n}, \text{Unify}(w, c)$  |
| 6    | $P_x \supseteq \{d\}$  
     | $\forall z \in P_x, \text{Unify}(d, z)$  |
Example of Equality Based (aka Steensgaard's) Points-to Analysis

Program:

1. \( x = \&a \)
2. \( y = \&b \)
3. \( x \rightarrow n = y \)
4. \( y = x \)
5. \( x \rightarrow n = \&c \)
6. \( x = \&d \)

Node | Constraint
--- | ---
1 | \( P_x \supseteq \{a\} \) \\
   | \( \forall z \in P_x, \text{Unify}(a, z) \)
2 | \( P_y \supseteq \{b\} \) \\
   | \( \forall z \in P_y, \text{Unify}(b, z) \)
3 | \( \forall z \in P_x, \text{UnifyPTS}(y, z.n) \)
4 | \( \text{UnifyPTS}(x, y) \)
5 | \( \forall z \in P_x, P_{z.n} \supseteq \{c\} \) \\
   | \( \forall w \in P_{z.n}, \text{Unify}(w, c) \)
6 | \( P_x \supseteq \{d\} \) \\
   | \( \forall z \in P_x, \text{Unify}(d, z) \)

Points-to Graph:

- Node 1 connected to \( x \)
- Node 2 connected to \( a \)
- Node 4 connected to \( y \)
- Node 5 connected to \( c \)
Example of Equality Based (aka Steensgaard’s) Points-to Analysis

Program

1. \( x = \&a \)
2. \( y = \&b \)
3. \( x \rightarrow n = y \)
4. \( y = x \)
5. \( x \rightarrow n = \&c \)
6. \( x = \&d \)

Node | Constraint
--- | ---
1 | \( P_x \supseteq \{a\} \)
   | \( \forall z \in P_x, \text{Unify}(a, z) \)
2 | \( P_y \supseteq \{b\} \)
   | \( \forall z \in P_y, \text{Unify}(b, z) \)
3 | \( \forall z \in P_x, \text{UnifyPTS}(y, z, n) \)
4 | \( \text{UnifyPTS}(x, y) \)
5 | \( \forall z \in P_x, P_{z, n} \supseteq \{c\} \)
   | \( \forall w \in P_{z, n}, \text{Unify}(w, c) \)
6 | \( P_x \supseteq \{d\} \)
   | \( \forall z \in P_x, \text{Unify}(d, z) \)

Points-to Graph

\( x \) \( a \)
\( y \) \( b \)
Example of Equality Based (aka Steensgaard’s) Points-to Analysis

Program

1. \[ x = &a \]
2. \[ y = &b \]
3. \[ x \rightarrow n = y \]
4. \[ y = x \]
5. \[ x \rightarrow n = &c \]
6. \[ x = &d \]

Node Constraint

<table>
<thead>
<tr>
<th>Node</th>
<th>Constraint</th>
</tr>
</thead>
</table>
| 1    | \[ P_x \supseteq \{a\} \] 
\[ \forall z \in P_x, \text{Unify}(a, z) \] |
| 2    | \[ P_y \supseteq \{b\} \] 
\[ \forall z \in P_y, \text{Unify}(b, z) \] |
| 3    | \[ \forall z \in P_x, \text{UnifyPTS}(y, z.n) \] |
| 4    | \[ \text{UnifyPTS}(x, y) \] |
| 5    | \[ \forall z \in P_x, P_{z.n} \supseteq \{c\} \] 
\[ \forall w \in P_{z.n}, \text{Unify}(w, c) \] |
| 6    | \[ P_x \supseteq \{d\} \] 
\[ \forall z \in P_x, \text{Unify}(d, z) \] |

Points-to Graph

- \[ x \rightarrow a \]
- \[ y \rightarrow b \]
Example of Equality Based (aka Steensgaard’s) Points-to Analysis

<table>
<thead>
<tr>
<th>Node</th>
<th>Constraint</th>
</tr>
</thead>
</table>
| 1    | $P_x \supseteq \{a\}$  
      | $\forall z \in P_x, \text{Unify}(a, z)$ |
| 2    | $P_y \supseteq \{b\}$  
      | $\forall z \in P_y, \text{Unify}(b, z)$ |
| 3    | $\forall z \in P_x, \text{UnifyPTS}(y, z.n)$ |
| 4    | $\text{UnifyPTS}(x, y)$ |
| 5    | $\forall z \in P_x, P_{z.n} \supseteq \{c\}$  
      | $\forall w \in P_{z.n}, \text{Unify}(w, c)$ |
| 6    | $P_x \supseteq \{d\}$  
      | $\forall z \in P_x, \text{Unify}(d, z)$ |

Points-to Graph:

```
  x ---- a ---- n
     |      |      |
     v      v      v
   y ---- b ---- n
```

Dec 2019 IIT Bombay
Example of Equality Based (aka Steensgaard’s) Points-to Analysis

Program

<table>
<thead>
<tr>
<th>Node</th>
<th>Constraint</th>
</tr>
</thead>
</table>
| 1    | \( P_x \supseteq \{ a \} \)  
\( \forall z \in P_x, Unify(a, z) \) |
| 2    | \( P_y \supseteq \{ b \} \)  
\( \forall z \in P_y, Unify(b, z) \) |
| 3    | \( \forall z \in P_x, UnifyPTS(y, z.n) \) |
| 4    | \( UnifyPTS(x, y) \) |
| 5    | \( \forall z \in P_x, P_{z.n} \supseteq \{ c \} \)  
\( \forall w \in P_{z.n}, Unify(w, c) \) |
| 6    | \( P_x \supseteq \{ d \} \)  
\( \forall z \in P_x, Unify(d, z) \) |

Points-to Graph

\( x \rightarrow a \)  
\( y \rightarrow b \)  
\( x \rightarrow n = y \)  
\( y = x \)  
\( x \rightarrow n = \& c \)  
\( x = \& d \)  

Dec 2019
Example of Equality Based (aka Steensgaard’s) Points-to Analysis

Program

![Program Diagram](image)

Node | Constraint |
--- | --- |
1 | $P_x \supseteq \{a\}$  
   $\forall z \in P_x, \text{Unify}(a, z)$ |
2 | $P_y \supseteq \{b\}$  
   $\forall z \in P_y, \text{Unify}(b, z)$ |
3 | $\forall z \in P_x, \text{UnifyPTS}(y, z, n)$ |
4 | $\text{UnifyPTS}(x, y)$ |
5 | $\forall z \in P_x, P_{z.n} \supseteq \{c\}$  
   $\forall w \in P_{z.n}, \text{Unify}(w, c)$ |
6 | $P_x \supseteq \{d\}$  
   $\forall z \in P_x, \text{Unify}(d, z)$ |

Points-to Graph
Example of Equality Based (aka Steensgaard's) Points-to Analysis

Program

1. $x = \& a$
2. $y = \& b$
3. $x \rightarrow n = y$
4. $y = x$
5. $x \rightarrow n = \& c$
6. $x = \& d$

Node | Constraint
--- | ---
1 | $P_x \supseteq \{a\}$
   | $\forall z \in P_x, \text{Unify}(a, z)$
2 | $P_y \supseteq \{b\}$
   | $\forall z \in P_y, \text{Unify}(b, z)$
3 | $\forall z \in P_x, \text{UnifyPTS}(y, z, n)$
4 | $\text{UnifyPTS}(x, y)$
5 | $\forall z \in P_x, P_{z.n} \supseteq \{c\}$
   | $\forall w \in P_{z.n}, \text{Unify}(w, c)$
6 | $P_x \supseteq \{d\}$
   | $\forall z \in P_x, \text{Unify}(d, z)$

Points-to Graph

No further change

Dec 2019
Example of Equality Based (aka Steensgaard’s) Points-to Analysis

Program

1. \( x = \& a \)
2. \( y = \& b \)
3. \( x \rightarrow n = y \)
4. \( y = x \)
5. \( x \rightarrow n = \& c \)
6. \( x = \& d \)

Node | Constraint
--- | ---
1 | \( P_x \supseteq \{ a \} \)
   \( \forall z \in P_x, \text{Unify}(a, z) \)
2 | \( P_y \supseteq \{ b \} \)
   \( \forall z \in P_y, \text{Unify}(b, z) \)
3 | \( \forall z \in P_x, \text{UnifyPTS}(y, z.n) \)
4 | \( \text{UnifyPTS}(x, y) \)
5 | \( \forall z \in P_x, P_{z.n} \supseteq \{ c \} \)
   \( \forall w \in P_{z.n}, \text{Unify}(w, c) \)
6 | \( P_x \supseteq \{ d \} \)
   \( \forall z \in P_x, \text{Unify}(d, z) \)

Points-to Graph

Red edges represent field \( n \) in the full blown up graph. It has far more edges than in Andersen’s graph.

Far more efficient but far less precise.
Comparing Equality and Inclusion Based Analyses

- 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

- 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.n = &amp;d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b.n = &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="Diagram for Andersen's approach" /></td>
<td><img src="image2" alt="Diagram for Steensgaard's approach" /></td>
</tr>
<tr>
<td>a = &amp;c</td>
<td><img src="image1" alt="Diagram for Andersen's approach" /></td>
<td><img src="image2" alt="Diagram for Steensgaard's approach" /></td>
</tr>
<tr>
<td>b.n = &amp;d</td>
<td><img src="image1" alt="Diagram for Andersen's approach" /></td>
<td><img src="image2" alt="Diagram for Steensgaard's approach" /></td>
</tr>
<tr>
<td>b.n = &amp;c</td>
<td><img src="image1" alt="Diagram for Andersen's approach" /></td>
<td><img src="image2" alt="Diagram for Steensgaard's approach" /></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.n = &amp;d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b.n = &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></td>
<td><img src="image1" alt="Diagram" /></td>
<td><img src="image2" alt="Diagram" /></td>
</tr>
<tr>
<td><code>a = &amp;c</code></td>
<td><img src="image3" alt="Diagram" /></td>
<td></td>
</tr>
<tr>
<td><code>b.n = &amp;d</code></td>
<td><img src="image4" alt="Diagram" /></td>
<td></td>
</tr>
<tr>
<td><code>b.n = &amp;c</code></td>
<td><img src="image5" alt="Diagram" /></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></td>
<td>![Diagram 1]</td>
<td>![Diagram 2]</td>
</tr>
<tr>
<td><code>a = &amp;c</code></td>
<td>![Diagram 3]</td>
<td>![Diagram 4]</td>
</tr>
<tr>
<td><code>b.n = &amp;d</code></td>
<td>![Diagram 5]</td>
<td>![Diagram 6]</td>
</tr>
<tr>
<td><code>b.n = &amp;c</code></td>
<td>![Diagram 7]</td>
<td>![Diagram 8]</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>![Diagram of Andersen's approach]</td>
<td>![Diagram of Steensgaard's approach]</td>
</tr>
<tr>
<td>a = &amp;c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b.n = &amp;d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b.n = &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>$b.n = &amp; d$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$b.n = &amp; c$</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>![Diagram A]</td>
<td>![Diagram B]</td>
</tr>
<tr>
<td>a = &amp;c</td>
<td>![Diagram A]</td>
<td>![Diagram B]</td>
</tr>
<tr>
<td>b.n = &amp;d</td>
<td>![Diagram A]</td>
<td>![Diagram B]</td>
</tr>
<tr>
<td>b.n = &amp;c</td>
<td>![Diagram A]</td>
<td>![Diagram B]</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="image" alt="Andersen's Diagram" /></td>
<td><img src="image" alt="Steensgaard's Diagram" /></td>
</tr>
<tr>
<td>(a = &amp; c)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(b.n = &amp; d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(b.n = &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

```c
struct s {
    struct s *f;
    int n;
} *x, *y, u, v;
```

```
x = &u
y = &v
```

```
y → f = x
y = &u
```

```
use v.f
use x
```
Inclusion Based (aka Andersen’s) Points-to Analysis: Example 2

```
struct s {
    struct s *f;
    int n;
} *x, *y, u, v;
```

Constraints on Points-to Sets

\[ P_x \supseteq \{u\} \]
\[ P_y \supseteq \{v\} \]

Andersen’s Points-to Graph

- \(x\) “points-to” \(u\)
- \(y\) “points-to” \(v\)
Inclusion Based (aka Andersen’s) Points-to Analysis: Example 2

```c
struct s {
    struct s *f;
    int n;
};

*x, *y, u, v;
```

1. \(x = \& u\)
2. \(y = \& v\)
3. \(y \rightarrow f = x\)
4. \(\text{use } v.f\)
   \(\text{use } x\)

- The \(f\) field of pointees of \(y\) should point to pointees of \(x\) also.
- The \(f\) field of \(v\) should point to \(u\) also.

Andersen’s Points-to Graph

 Constraints on Points-to Sets

\[
P_x \supseteq \{u\} \\
P_y \supseteq \{v\} \\
\forall w \in P_y, P_{w.f} \supseteq P_x
\]
Inclusion Based (aka Andersen’s) Points-to Analysis: Example 2

```
struct s {
    struct s *f;
    int n;
}
*x, *y, u, v;
```

1. \(x = &u\)
2. \(y = &v\)
3. \(y \rightarrow f = x\)
4. use \(v.f\)
   use \(x\)

- The \(f\) field of pointees of \(y\) should point to pointees of \(x\) also
- The \(f\) field of \(v\) should point to \(u\) also

Constraints on Points-to Sets

\[P_x \supseteq \{u\}\]
\[P_y \supseteq \{v\}\]
\[\forall w \in P_y, P_{w.f} \supseteq P_x\]

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

```
struct s {
    struct s *f;
    int n;
} *x, *y, u, v;
```

1. \( x = &u \)
2. \( y \rightarrow f = x \)
3. \( y = &u \)
4. use \( v.f \)
   use \( x \)

Constraints on Points-to Sets

\[
\begin{align*}
P_x &\supseteq \{u\} \\
P_y &\supseteq \{v\} \\
\forall w \in P_y, \ P_{w.f} &\supseteq P_x \\
P_y &\supseteq \{u\}
\end{align*}
\]

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

```c
struct s {
    struct s *f;
    int n;
} *x, *y, u, v;
```

```
x = &u
y = &v

y → f = x
```

- y should point to u also

```
use v.f
use x
```

Constraints on Points-to Sets

\[
P_x \supseteq \{u\}
P_y \supseteq \{v\}
\]

\[
\forall w \in P_y, \quad P_{w.f} \supseteq P_x
\]

\[
P_y \supseteq \{u\}
\]

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

struct s {
    struct s *f;
    int n;
} *x, *y, u, v;

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

\[ y \to f = x \]
\[ y = &u \]
\[ \text{use } v.f \]
\[ \text{use } x \]

Constraints on Points-to Sets

\[ P_x \supseteq \{u\} \]
\[ P_y \supseteq \{v\} \]
\[ \forall w \in P_y, \ P_{w.f} \supseteq P_x \]
\[ P_y \supseteq \{u\} \]

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

```c
struct s {
    struct s *f;
    int n;
} *x, *y, u, v;
```

Constraints on Points-to Sets

\[ P_x \supseteq \{ u \} \]
\[ P_y \supseteq \{ v \} \]
\[ \forall w \in P_y, P_{w.f} \supseteq P_x \]
\[ P_y \supseteq \{ u \} \]

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

```c
struct s {
    struct s *f;
    int n;
}
*x, *y, u, v;
```

Constraints on Points-to Sets

$$P_x \supseteq \{u\}$$

$$P_y \supseteq \{v\}$$

$$\forall w \in P_y, P_{w.f} \supseteq P_x$$

$$P_y \supseteq \{u\}$$

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

```c
struct s {
    struct s *f;
    int n;
} *x, *y, u, v;
```

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

Andersen’s Points-to Graph

Dec 2019
Equality Based (aka Steensgaard’s) Points-to Analysis: Example 2

```
struct s {
    struct s *f;
    int n;
} *x, *y, u, v;
```

1. \( x = \& u \)
2. \( y \rightarrow f = x \)
3. \( y = \& u \)
4. `use y.f`
   `use x`

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

**Andersen’s Points-to Graph**

**Effective additional constraints**

```
Unify(u, v)
/* pointees of y */
```
**Equality Based (aka Steensgaard's) Points-to Analysis: Example 2**

```
struct s {
    struct s *f;
    int n;
} *x, *y, u, v;
```

1. \( x = &u \)
2. \( y = &v \)
3. \( y \rightarrow f = x \)
4. \( y = &u \)

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

Steengaard's Points-to Graph:

- Effective additional constraints
  - \( \text{Unify}(u, v) \) /* pointees of \( y \) */
  - \( \Rightarrow u, v \) are equivalent
Equality Based (aka Steensgaard’s) Points-to Analysis: Example 2

```c
struct s {
    struct s *f;
    int n;
} *x, *y, u, v;
```

1. \( x = &u \)
2. \( y = &v \)
3. \( y \rightarrow f = x \)
4. \( y = &u \)

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

Pointees of all equivalent locations become equivalent

```
use v.f
use x
```

Use Steengaard’s Points-to Graph

Effective additional constraints

\[
\text{Unify}(u, v)
\]

/* pointees of y */

\( \Rightarrow u, v \) are equivalent
Equality Based (aka Steensgaard's) Points-to Analysis: Example 2

```c
struct s {
    struct s *f;
    int n;
}
*x, *y, u, v;
```

1. \( x = &u \)
2. \( y = &v \)
3. \( y \rightarrow f = x \)
4. \( y = &u \)

- Treat all pointees of a pointer as "equivalent" locations
- Transitive closure

```
use v.f
use x
```

Effective additional constraints

\[
\text{Unify}(u, v)
\]

\(/^{*}\) pointees of \( y \)/

\( \Rightarrow u, v \) are equivalent

Steengaard's Points-to Graph
## Tutorial Problem for Flow-Insensitive Pointer Analysis

<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></td>
<td></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>
Tutorial Problem for Flow-Insensitive Pointer Analysis

Program

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

Inclusion based

Equality based
Tutorial Problem for Flow-Insensitive Pointer Analysis

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**

<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>u</td>
<td>q</td>
</tr>
<tr>
<td>t = &amp;p</td>
<td>q</td>
<td></td>
</tr>
<tr>
<td>u = p</td>
<td>p</td>
<td>s</td>
</tr>
<tr>
<td>*t = r</td>
<td>r</td>
<td></td>
</tr>
</tbody>
</table>
Tutorial Problem for Flow-Insensitive Pointer Analysis

Program

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

Inclusion based

Equality based

\[ u \quad q \]
\[ t \quad p \]
\[ r \quad s \]
Tutorial Problem for Flow-Insensitive Pointer Analysis

Program

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

Inclusion based

Equality based
**Tutorial Problem for Flow-Insensitive Pointer Analysis**

<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>

Graph:

```
  u -> q
  t -> p
  r -> s
```

Dec 2019
Tutorial Problem for Flow-Insensitive Pointer Analysis

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

<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="" alt="Inclusion based graph" /></td>
<td><img src="" 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>
Tutorial Problem for Flow-Insensitive Pointer Analysis

Program

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

Inclusion based

Equality based

Dec 2019 IIT Bombay
Tutorial Problem for Flow-Insensitive Pointer Analysis

Program

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

Inclusion based

Equality based

Dec 2019
Tutorial Problem for Flow-Insensitive Pointer Analysis

Program

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

Inclusion based

Equality based
Tutorial Problem for Flow-Insensitive Pointer Analysis

Program

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

Inclusion based

Equality based
Tutorial Problem for Flow-Insensitive Pointer Analysis

Program

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

Inclusion based

Equality based
Tutorial Problem for Flow-Insensitive Pointer Analysis

Program

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

Inclusion based

Equality based
Tutorial Problem for Flow-Insensitive Pointer Analysis

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

Program

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

Inclusion based

Equality based
An Outline of Pointer Analysis Coverage

- The larger perspective
- IR for Points-to Analysis
- Flow-Insensitive Points-to Analysis
- Flow-Sensitive Points-to Analysis

Next Topic
Must Points-to Information

1. \( x = \&a \)

2

3

4

\( a \)

\( x \)

\( b \)
Must Points-to Information
May Points-to Information

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

Diagram showing pointers and assignments:
- $x$ points to $a$
- $x$ points to $b$
May Points-to Information

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

Diagram:

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

Audiocaption: May Points-to Information

Diagram:

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

Dec 2019
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

- **Weak update**: Modification of $x$ or $y$ due to $*z$ in block 5
  
  Only Gen, No Kill

```
1. $x = &a$

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

3. $z = &x$

4. $z = &y$

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

- **Weak update**: Modification of \( x \) or \( y \) due to \(*z\) in block 5
  
  Only Gen, No Kill

- **Strong update**: Modification of \( c \) due to \(*w\) in block 5
  
  Both Gen and Kill
Strong and Weak Updates

- **Weak update**: Modification of $x$ or $y$ due to $\ast z$ in block 5
  Only Gen, No Kill

- **Strong update**: Modification of $c$ due to $\ast w$ in block 5
  Both Gen and Kill

- **How is this concept related to May/Must nature of information?**
May and Must Analysis for Killing Points-to Information (1)

MFP of May Points-to Analysis

MFP of Must Points-to Analysis

```
1: a = &b
2: c = &a
3: c = &a
4: *c = &e
5: 
```
May and Must Analysis for Killing Points-to Information (1)

MFP of May Points-to Analysis

• \((a, b)\) should be in \(\text{MayIn}_5\)
  Holds along path 1-3-4

• \((a, b)\) should not be killed in node 4

• Possible if pointee set of \(c\) is \(\emptyset\)

• However, \(\text{MayIn}_4\) contains \((c, a)\)

MFP of Must Points-to Analysis

\[
\begin{align*}
1: & \quad a = \& b \\
2: & \quad c = \& a \\
3: & \quad \ast c = \& e \\
4: & \quad \ast c = \& e \\
5: & \quad \\
\end{align*}
\]
May and Must Analysis for Killing Points-to Information (1)

**MFP of May Points-to Analysis**
- \((a, b)\) should be in \(\text{MayIn}_5\)
  - Holds along path 1-3-4
- \((a, b)\) should not be killed in node 4
- Possible if pointee set of \(c\) is \(\emptyset\)
- However, \(\text{MayIn}_4\) contains \((c, a)\)

**MFP of Must Points-to Analysis**
- \((a, b)\) should not be in \(\text{MustIn}_5\)
  - Does not hold along path 1-2-4
- \((a, b)\) should be killed in node 4
- Possible if pointee set of \(c\) is \(\{a\}\)
- However, the pointee set of \(c\) is \(\emptyset\) in \(\text{MustIn}_4\)
May and Must Analysis for Killing Points-to Information (1)

**MFP of May Points-to Analysis**

- \((a, b)\) should be in \(\text{MayIn}_5\)
  - Holds along path 1-3-4
- \((a, b)\) should not be killed in node 4
- Possible if pointee set of \(c\) is \(\emptyset\) (Use \(\text{MustIn}_4\))
- However, \(\text{MayIn}_4\) contains \((c, a)\) (Use \(\text{MustIn}_4\))

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 \(\text{MustIn}_4\)

**MFP of Must Points-to Analysis**

- \((a, b)\) should not be in \(\text{MustIn}_5\)
  - Does not hold along path 1-2-4
- \((a, b)\) should be killed in node 4
- Possible if pointee set of \(c\) is \(\{a\}\) (Use \(\text{MayIn}_4\))
- However, the pointee set of \(c\) is \(\emptyset\) in \(\text{MustIn}_4\) (Use \(\text{MayIn}_4\))
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

  ⇒ 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

  ⇒ should use May Points-to information
Distinguishing Between Strong and Weak Updates

Indirect assignment
\[ n: \; \*x = \ldots \]

pointer \( x \) has a single pointee

Every path reaching \( n \) has a definition of \( x \)

Strong update

pointer \( x \) has multiple pointees

Some path reaching \( n \) does not have a definition of \( x \)

Weak update
Distinguishing Between Strong and Weak Updates

Can be eliminated if definition-free paths are eliminated

Indirect assignment

$n$: $*x = \ldots$

pointer $x$ has multiple pointees

pointer $x$ has a single pointee

Every path reaching $n$ has a definition of $x$

Some path reaching $n$ does not have a definition of $x$

Strong update

Weak update

Dec 2019 IIT Bombay
Distinguishing Between Strong and Weak Updates

Indirect assignment

\[ n: \ast x = \ldots \]

pointer \( x \) has a single pointee

pointer \( x \) has multiple pointees

Every path reaching \( n \) has a definition of \( x \)

Strong update

Weak update
Discovering Must Points-to Information from May Points-to Information

\[ a = &b \\
 b = &e \]

\[ c = &a \]

\[ a = &d \]

\[ c = &d \]

\[ a = &e \]
Discovering Must Points-to Information from May Points-to Information

1. \(a = \&b\)
   \(b = \&e\)

2. \(c = \&a\)

3. BI. every pointer points to "?"
   Assume that \(e\) is a scalar

4.
Discovering Must Points-to Information from May Points-to Information

- BI. every pointer points to “?”

Assume that e is a scalar

```
a = &b  
b = &e  
c = &a
```

Dec 2019
Discovering Must Points-to Information from May Points-to Information

- BI. every pointer points to “?”
  Assume that \( e \) is a scalar
- Perform usual may points-to analysis

\[
\begin{align*}
1 & \quad a = \&b \\
   & \quad b = \&e \\
2 & \quad c = \&a \\
3 & \quad \text{Box} \\
4 & \quad \text{Box}
\end{align*}
\]
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 “?”
   Assume that \( e \) is a scalar
Discovering Must Points-to Information from May Points-to Information

1. Assume that $e$ is a scalar.

2. Perform usual may points-to analysis.

3. $c = \&a$

4. Every pointer points to "?"
Discovering Must Points-to Information from May Points-to Information

1. $a \equiv &b$
   $b \equiv &e$

2. $c \equiv &a$

3. ?

4. ?

- $B1$. every pointer points to “?”
  Assume that $e$ is a scalar
- 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 "?"
  Assume that $e$ is a scalar
- 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
Discovering Must Points-to Information from May Points-to Information

The use of “?” to derive Must is valid under the following conditions:

If there is a definition free path from Start to node $i$ for pointer $x$, then $(x, ?)$ must reach $In_i$ during the very first visit to node $i$ in the analysis.

Conversely, if there is no definition free path from Start to node $i$ for pointer $x$, then $(x, ?)$ must not reach $In_i$ during the very first visit to node $i$ in the analysis.
Relevant Algebraic Operations on Relations (1)

- Let $P \subseteq V$ be the set of pointer variables
- May-points-to information: $\mathcal{A} = \langle 2^{P \times V}, \supseteq \rangle$
- Standard algebraic operations on points-to relations
  Given relation $R \subseteq P \times V$ and $X \subseteq P$,
  - Relation application $R \times X = \{ v \mid u \in X \land (u, v) \in R \}$
  - Relation restriction $(R|_X) R|_X = \{ (u, v) \in R \mid u \in X \}$
• Let $P \subseteq V$ be the set of pointer variables

• May-points-to information: $A = \langle 2^{P \times V}, \supseteq \rangle$

• Standard algebraic operations on points-to relations
  Given relation $R \subseteq P \times V$ and $X \subseteq P$,
  
  o Relation \textit{application} $R \times X = \{v \mid u \in X \land (u, v) \in R\}$
    (Find out the pointees of the pointers contained in $X$)
  o Relation \textit{restriction} $(R|_X) \times X = \{(u, v) \mid (u, v) \in R \land u \in X\}$
Relevant Algebraic Operations on Relations (1)

• Let $P \subseteq V$ be the set of pointer variables

• May-points-to information: $\mathcal{A} = \langle 2^{P \times V}, \supseteq \rangle$

• Standard algebraic operations on points-to relations
  
  Given relation $R \subseteq P \times V$ and $X \subseteq P$,
  
  - Relation *application* $R \times X = \{ v \mid u \in X \land (u, v) \in R \}$
    (Find out the pointees of the pointers contained in $X$)
  
  - Relation *restriction* $(R\mid_X) R\mid_X = \{ (u, v) \in R \mid u \in X \}$
    (Restrict the relation only to the pointers contained in $X$ by removing points-to information of other pointers)
Let

\[ V = \{a, b, c, d, e, f, g, ?\} \]
\[ P = \{a, b, c, d, e\} \]
\[ R = \{(a, b), (a, c), (b, d), (c, e), (c, g), (d, a), (e, ?)\} \]
\[ X = \{a, c\} \]

Then,

\[ R \times X = \{v \mid u \in X \land (u, v) \in R\} \]
\[ R \mid_X = \{(u, v) \in R \mid u \in X\} \]
Relevant Algebraic Operations on Relations (2)

Let

\[ V = \{a, b, c, d, e, f, g, ?\} \]
\[ P = \{a, b, c, d, e\} \]
\[ R = \{(a, b), (a, c), (b, d), (c, e), (c, g), (d, a), (e, ?)\} \]
\[ X = \{a, c\} \]

Then,

\[ R \times X = \{v \mid u \in X \land (u, v) \in R\} \]
\[ = \{b, c, e, g\} \]
\[ R\vert_X = \{(u, v) \in R \mid u \in X\} \]
Let

\[ V = \{ a, b, c, d, e, f, g, ? \} \]
\[ P = \{ a, b, c, d, e \} \]
\[ R = \{ (a, b), (a, c), (b, d), (c, e), (c, g), (d, a), (e, ?) \} \]
\[ X = \{ a, c \} \]

Then,

\[ R \times X = \{ v \mid u \in X \land (u, v) \in R \} \]
\[ = \{ b, c, e, g \} \]
\[ R|_X = \{ (u, v) \in R \mid u \in X \} \]
\[ = \{ (a, b), (a, c), (c, e), (c, g) \} \]
Points-to Analysis Data Flow Equations

\[
\begin{align*}
Pin_n &= \begin{cases} 
V \times \{?\} & \text{n is } \text{Start}_p \\
\bigcup_{p \in \text{pred}(n)} Pout_p & \text{otherwise}
\end{cases} \\
\text{Pout}_n &= \left( Pin_n - \left( \text{Kill}_n \times V \right) \right) \cup \left( \text{Def}_n \times \text{Pointee}_n \right)
\end{align*}
\]

- *Pin/Pout*: sets of may points-to pairs
- *Kill*_n, *Def*_n, and *Pointee*_n are defined in terms of *Pin*_n
Points-to Analysis Data Flow Equations

\[ Pin_n = \begin{cases} 
V \times \{?\} & n \text{ is } \text{Start}_p \\
\bigcup_{p \in \text{pred}(n)} Pout_p & \text{otherwise}
\end{cases} \]

\[ Pout_n = \left( Pin_n - \left( \text{Kill}_n \times V \right) \right) \cup \left( \text{Def}_n \times \text{Pointee}_n \right) \]

- \( Pin/Pout \): sets of may points-to pairs
- \( \text{Kill}_n, \text{Def}_n, \) and \( \text{Pointee}_n \) are defined in terms of \( Pin_n \)

Pointers whose points-to relations should be removed for strong update.
Points-to Analysis Data Flow Equations

\[ Pin_n = \begin{cases} 
V \times \{?\} & n \text{ is Start}_p \\
\bigcup_{p \in \text{pred}(n)} Pout_p & \text{otherwise}
\end{cases} \]

\[ Pout_n = (Pin_n - (Kill_n \times V)) \cup (Def_n \times Pointee_n) \]

- \( Pin/Pout\): sets of may points-to pairs
- \( Kill_n, Def_n, \) and \( Pointee_n \) are defined in terms of \( Pin_n \)

Pointers that are defined (i.e., pointers in which addresses are stored)
Points-to Analysis Data Flow Equations

\[ Pin_n = \begin{cases} 
  V \times \{?\} & \text{if } n \text{ is } \text{Start}_p \\
  \bigcup_{p \in \text{pred}(n)} Pout_p & \text{otherwise}
\end{cases} \]

\[ Pout_n = \left( Pin_n - \left( Kill_n \times V \right) \right) \cup \left( Def_n \times \text{Pointee}_n \right) \]

- \(Pin/Pout\): sets of may points-to pairs
- \(Kill_n, \ Def_n,\) and \(\text{Pointee}_n\) are defined in terms of \(Pin_n\)
Points-to Analysis Data Flow Equations

\[
\begin{align*}
Pin_n &= \begin{cases} 
  V \times \{?\} & \text{if } n \text{ is } \text{Start}_p \\
  \bigcup_{p \in \text{pred}(n)} Pout_p & \text{otherwise}
\end{cases} \\

Pout_n &= \left( Pin_n - \left( \text{Kill}_n \times V \right) \right) \cup \left( \text{Def}_n \times \text{Pointee}_n \right)
\end{align*}
\]

- \textit{Pin/Pout}: sets of may points-to pairs
- \textit{Kill}_n, \textit{Def}_n, and \textit{Pointee}_n are defined in terms of \textit{Pin}_n
Extractor Functions for Points-to Analysis

Values defined in terms of $Pin_n$ (denoted $P$)

<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>
Extractor Functions for Points-to Analysis

Values defined in terms of $Pin_n$ (denoted $P$)

<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 $Pin_n$ (denoted $P$)

<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>

Pointees (i.e. locations whose addresses are stored)
Extractor Functions for Points-to Analysis

Values defined in terms of $Pin_n$ (denoted $P$)

<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>

Pointers whose points-to relations should be removed for strong update
Extractor Functions for Points-to Analysis

Values defined in terms of $Pin_n$ (denoted $P$)

<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 $Pin_n$ (denoted $P$)

<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>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$*x = y$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>other</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
Extractor Functions for Points-to Analysis

Values defined in terms of $Pin_n$ (denoted $P$)

<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>$P{y}$</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>

Pointees of $y$ in $Pin_n$ are the targets of defined pointers
## Extractor Functions for Points-to Analysis

Values defined in terms of $Pin_n$ (denoted $P$)

<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>$P{y}$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$P(P{y} \cap P)$</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 those pointees of $y$ in $Pin_n$ which are pointers
Extractor Functions for Points-to Analysis

Values defined in terms of $Pin_n$ (denoted $P$)

<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>$P{y}$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$P(P{y} \cap P)$</td>
</tr>
<tr>
<td>$*x = y$</td>
<td>$P{x} \cap P$</td>
<td>$Must(P){x} \cap P$</td>
<td>$P{y}$</td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pointees of $x$ in $Pin_n$ receive new addresses
## Extractor Functions for Points-to Analysis

Values defined in terms of $Pin_n$:

<table>
<thead>
<tr>
<th></th>
<th>$Def_n$</th>
<th>$Kill_n$</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>$P{x} \cap P$</td>
<td>$Must(P){x} \cap P$</td>
</tr>
<tr>
<td>other</td>
<td></td>
<td>$P{y}$</td>
</tr>
</tbody>
</table>

**Strong update using must-points-to information computed from $Pin_n$**

$$Must(R) = \bigcup_{z \in P} \{z\} \times \begin{cases} \{w\} & R\{z\} = \{w\} \land w \neq ? \\ \emptyset & \text{otherwise} \end{cases}$$
Extractor Functions for Points-to Analysis

Values defined in terms of $Pin_n$:

<table>
<thead>
<tr>
<th>Operation</th>
<th>$Def_n$</th>
<th>$Kiln_n$</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>$P{x} \cap P$</td>
<td>$\text{Must}(P){x} \cap P$</td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Strong update using must-points-to information computed from $Pin_n$:

$$Must(R) = \bigcup_{z \in P} \{z\} \times \begin{cases} \{w\} & R\{z\} = \{w\} \land w \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$

Find out must-pointees of all pointers.
Extractor Functions for Points-to Analysis

Values defined in terms of $Pin_n$:

<table>
<thead>
<tr>
<th></th>
<th>$Def_n$</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>${x}$</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>$P{x} \cap P$</td>
<td>$\text{Must}(P){x} \cap P$</td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Strong update using must-points-to information computed from $Pin_n$.

$$Must(R) = \bigcup_{z \in P} \{z\} \times \begin{cases} \{w\} & \text{if } z \text{ has a single pointee } w \text{ in must-points-to relation} \\ \emptyset & \text{otherwise} \end{cases}$$

$$R\{z\} = \{w\} \land w \neq ?$$
Extractor Functions for Points-to Analysis

Values defined in terms of $Pin_n$:

<table>
<thead>
<tr>
<th></th>
<th>$Def_n$</th>
<th>$Kill_n$</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>$P{x} \cap P$</td>
<td>$\mathbf{Must(P){x} \cap P}$</td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Strong update using must-points-to information computed from $Pin_n$.

$Must(R) = \bigcup_{z \in P} \{z\} \times \begin{cases} \{w\} & R\{z\} = \{w\} \land w \neq ? \\ \emptyset & \text{otherwise} \end{cases}$

$z$ has no pointee in must-points-to relation.
Extractor Functions for Points-to Analysis

Values defined in terms of $Pin_n$ (denoted $P$)

<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>$P{y}$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$P({y} \cap P)$</td>
</tr>
<tr>
<td>$*x = y$</td>
<td>$P{x} \cap P$</td>
<td>$Must(P){x} \cap P$</td>
<td>$P{y}$</td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$Must(R) = \bigcup_{z \in P} \{z\} \times \begin{cases} \{w\} & R\{z\} = \{w\} \land w \neq ? \\ \emptyset & \text{otherwise} \end{cases}$
Extractor Functions for Points-to Analysis

Values defined in terms of $Pin_n$ (denoted $P$)

<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>$P{y}$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$P(P{y} \cap P)$</td>
</tr>
<tr>
<td>$*x = y$</td>
<td>$P{x} \cap P$</td>
<td>$Must(P){x} \cap P$</td>
<td>$P{y}$</td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pointees of $y$ in $Pin_n$ are the targets of defined pointers

$$Must(R) = \bigcup_{z \in P} \{z\} \times \begin{cases} \{w\} & R\{z\} = \{w\} \land w \neq ? \\ \emptyset & \text{otherwise} \end{cases}$$
Extractor Functions for Points-to Analysis

Values defined in terms of $Pin_n$ (denoted $P$)

<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>$P{y}$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$P(P{y} \cap P)$</td>
</tr>
<tr>
<td>$*x = y$</td>
<td>$P{x} \cap P$</td>
<td>$Must(P){x} \cap P$</td>
<td>$P{y}$</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 $Pin_n$ (denoted $P$)

<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>$P{y}$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$P(P{y} \cap P)$</td>
</tr>
<tr>
<td>$*x = y$</td>
<td>$P{x} \cap P$</td>
<td>$Must(P){x} \cap P$</td>
<td>$P{y}$</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 $Pin_n$ (denoted $P$)

<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>$P{y}$</td>
</tr>
<tr>
<td>$x = *y$</td>
<td>${x}$</td>
<td>${x}$</td>
<td>$P(P{y} \cap P)$</td>
</tr>
<tr>
<td>$*x = y$</td>
<td>$P{x} \cap P$</td>
<td>$Must(P){x} \cap P$</td>
<td>$P{y}$</td>
</tr>
<tr>
<td>other</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

$$Must(R) = \bigcup_{z \in P} \{z\} \times \begin{cases} \{w\} & R\{z\} = \{w\} \land w \neq ? \\ \emptyset & \text{otherwise} \end{cases}$$
An Example of Flow-Sensitive May Points-to Analysis

```c
int w;
int *u, *v, *x;
int **y, **z;
```

```
y = &v
z = &u
x = &w
```

```
*Z = X
z = y
```

```
Use u
```

Dec 2019 IIT Bombay
An Example of Flow-Sensitive May Points-to Analysis

```c
int w;
int *u, *v, *x;
int **y, **z;

y = &v
z = &u
x = &w

*z = x
z = y

Use u
```
An Example of Flow-Sensitive May Points-to Analysis

```
int w;
int *u, *v, *x;
int **y, **z;
```

```
y = &v
z = &u
x = &w
```

```
*y = x
z = y
```

```
Use u
```

Dec 2019
An Example of Flow-Sensitive May Points-to Analysis

```
int w;
int *u, *v, *x;
int **y, **z;
```

```
y = &v
z = &u
x = &w

*z = x
```

Dec 2019 IIT Bombay
An Example of Flow-Sensitive May Points-to Analysis

```c
int w;
int *u, *v, *x;
int **y, **z;

y = &v
z = &u
x = &w

*z = x
z = y

Use u
```

**Strong Update**
An Example of Flow-Sensitive May Points-to Analysis

```c
int w;
int *u, *v, *x;
int **y, **z;

y = &v
z = &u
x = &w

*z = x
n2

z = y
n3

Use u
n4
```

Dec 2019
An Example of Flow-Sensitive May Points-to Analysis

```c
int w;
int *u, *v, *x;
int **y, **z;
```

1. \( y = \& v \)
2. \( z = \& u \)
3. \( x = \& w \)
4. Use \( u \)
An Example of Flow-Sensitive May Points-to Analysis

```c
int w;
int *u, *v, *x;
int **y, **z;
```

Diagram:

1. $y = &v$
2. $z = &u$
3. $x = &w$
4. $*z = x$
5. $z = y$

Dec 2019 IIT Bombay
Tutorial Problem for Flow-Sensitive Pointer Analysis

```c
int a, b, c, *p, *q, *r;
int **y, ***x;
```

Diagram:

1. \( x = \& y \)
2. \( y = \& r \)
3. \( q = \& c \)
4. \( *x = \& p \)
5. \( p = \& a \)
6. \( *x = \& q \)
7. \( *y = \& b \)
# Solution of Tutorial Problem

<table>
<thead>
<tr>
<th>Pin(_n)</th>
<th>Pout(_n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 {(p, ?), (q, ?), (r, ?), (x, ?), (y, ?)}</td>
<td>{(p, ?), (q, ?), (r, ?), (x, y), (y, ?)}</td>
</tr>
<tr>
<td>2 {(p, ?), (q, ?), (r, ?), (x, y), (y, ?)}</td>
<td>{(p, ?), (q, ?), (r, ?), (x, y), (y, r)}</td>
</tr>
<tr>
<td>3 {(p, ?), (q, ?), (r, ?), (x, y), (y, r)}</td>
<td>{(p, ?), (q, c), (r, ?), (x, y), (y, r)}</td>
</tr>
<tr>
<td>4 {(p, ?), (q, c), (r, ?), (x, y), (y, r)}</td>
<td>{(p, ?), (q, c), (r, ?), (x, y), (y, p)}</td>
</tr>
<tr>
<td>5 {(p, ?), (q, ?), (r, ?), (x, y), (y, r)}</td>
<td>{(p, a), (q, ?), (r, ?), (x, y), (y, r)}</td>
</tr>
<tr>
<td>6 {(p, a), (q, ?), (r, ?), (x, y), (y, r)}</td>
<td>{(p, a), (q, ?), (r, ?), (x, y), (y, q)}</td>
</tr>
<tr>
<td>7 {(p, ?), (p, a), (q, ?), (q, c), (r, ?), (x, y), (y, p)(y, q)}</td>
<td>{(p, ?), (p, a), (p, b), (q, ?), (q, c), (q, b), (r, ?), (x, y), (y, p)(y, q)}</td>
</tr>
</tbody>
</table>
Extractor Functions in the Presence of Structures (1)

- We extend pointer to use field names as follows:
  - pointer $x$ is represented by $(x, \ast)$, and
  - pointer field $f$ of structure variable $x$ is represented by $(x, f)$
  - points-to information is of the form $((x, f) y)$

- For simplicity, we
  - separate LHS and RHS assuming that
  - only legal, type-correct pointer expressions are used in a statement

- From LHS, we extract $Def$ and $Kill$ as the sets of $(x, \ast)$ or $(a, f)$
  ($x$ is a pointer variable and $a$ is a structure variable)

- From RHS, we extract $Pointee$ as the sets of variables $x$
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 of pointer expression: Usually based on the length of pointer expression
Program

1. \( x = \text{malloc}(\ldots) \)

2. \( y = x \)

3. \( y \to f = \text{malloc}(\ldots) \)

4. \( y = y \to f \)

5. \( \text{assert } (y \neq \text{NULL}) \)
Allocation Site Based Abstraction of Points-to Graph

Program

1. $x = \text{malloc}(\ldots)$
2. $y = x$
3. $y \rightarrow f = \text{malloc}(\ldots)$
4. $y = y \rightarrow f$
5. \text{assert} (y \neq \text{NULL})

Memory graph representing multiple executions

Program

1. $x = \text{malloc}(\ldots)$
2. $y = x$
3. $y \rightarrow f = \text{malloc}(\ldots)$
4. $y = y \rightarrow f$
5. \text{assert} (y \neq \text{NULL})
Allocation Site Based Abstraction of Points-to Graph

Program

1. \( x = \text{malloc}(\ldots) \)
2. \( y = x \)
3. \( y \rightarrow f = \text{malloc}(\ldots) \)
4. \( y = y \rightarrow f \)
5. \( \text{assert } (y \neq NULL) \)

Memory graph representing multiple executions

Allocation-site based points-to graph
Allocation Site Based Abstraction of Points-to Graph

Program

1 \( x = malloc(\ldots) \)

2 \( y = x \)

3 \( y \rightarrow f = malloc(\ldots) \)

4 \( y = y \rightarrow f \)

5 \( \text{assert } (y \neq \text{NULL}) \)

Memory graph representing multiple executions

Allocation-site based points-to graph

Dec 2019

IIT Bombay
Allocation Site Based Abstraction of Points-to Graph

Program

1. \( x = \text{malloc}(\ldots) \)
2. \( y = x \)
3. \( y \to f = \text{malloc}(\ldots) \)
4. \( y = y \to f \)
5. \( \text{assert } (y \neq \text{NULL}) \)

Memory graph representing multiple executions

Allocation-site based points-to graph
Extractor Functions in the Presence of Structures (2)

<table>
<thead>
<tr>
<th>LHS</th>
<th>Def$_n$</th>
<th>Kill$_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>${(x,*)}$</td>
<td>${(x,*)}$</td>
</tr>
<tr>
<td>$*x$</td>
<td>${(z,<em>) \mid z \in A{(x,</em>)}}$</td>
<td>${(z,<em>) \mid z \in \text{Must}(A){(x,</em>)}}$</td>
</tr>
<tr>
<td>$x \rightarrow f$</td>
<td>${(z,f) \mid z \in A{(x,*)}}$</td>
<td>${(z,f) \mid z \in \text{Must}(A){(x,*)}}$</td>
</tr>
<tr>
<td>$x.f$</td>
<td>${(x,f)}$</td>
<td>${(x,f)}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RHS</th>
<th>Pointee$_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$&amp;y$</td>
<td>${y}$</td>
</tr>
<tr>
<td>$y$</td>
<td>${z \mid z \in A{(y,*)}}$</td>
</tr>
<tr>
<td>$*y$</td>
<td>${z \mid z \in A{(w,<em>)}, w \in A{(y,</em>)}}$</td>
</tr>
<tr>
<td>$y \rightarrow f$</td>
<td>${z \mid z \in A{(w,f)}, w \in A{(y,*)}}$</td>
</tr>
<tr>
<td>$y.f$</td>
<td>${z \mid z \in A{(y,f)}}$</td>
</tr>
</tbody>
</table>
An Example of Flow-Sensitive May Points-to Analysis

1. \( x = \&u \)
   \( y = \&v \)

2. \( y \rightarrow f = x \)

3. \( y = \&u \)

4. \( \text{use } v.f \)
   \( \text{use } x \)

Type Information

```c
struct s {
    struct s *f;
    int n;
};
*x, *y, u, v;
```

Andersen’s Points-to Graph

Steensgaard’s Points-to Graph
An Example of Flow-Sensitive May Points-to Analysis

1. \( x = \& u \)
   \( y = \& v \)

2. \( y \rightarrow f = x \)

3. \( y = \& u \)

4. use \( v.f \)
   use \( x \)

Type Information

```
struct s {
    struct s *f;
    int n;
}
*x, *y, u, v;
```

Andersen's Points-to Graph

Steensgaard's Points-to Graph
An Example of Flow-Sensitive May Points-to Analysis

Type Information

```c
struct s {
    struct s *f;
    int n;
}
*x, *y, u, v;
```

Andersen's Points-to Graph

Steensgaard's Points-to Graph
An Example of Flow-Sensitive May Points-to Analysis

```c
struct s {
    struct s *f;
    int n;
}

*x, *y, u, v;
```

Type Information

Andersen’s Points-to Graph

Steensgaard’s Points-to Graph
An Example of Flow-Sensitive May Points-to Analysis

Type Information

```
struct s {
    struct s *f;
    int n;
}
```

*`x, *y, u, v;`

Andersen's Points-to Graph

Steensgaard's Points-to Graph

Dec 2019 IIT Bombay
An Example of Flow-Sensitive May Points-to Analysis

Type Information

```c
struct s {
    struct s *f;
    int n;
} *x, *y, u, v;
```

Andersen's Points-to Graph

Steensgaard's Points-to Graph
An Example of Flow-Sensitive May Points-to Analysis

Type Information

```c
struct s {
    struct s *f;
    int n;
};
*x, *y, u, v;
```

Andersen’s Points-to Graph

Steensgaard’s Points-to Graph
An Example of Flow-Sensitive May Points-to Analysis

Type Information

```c
struct s {
    struct s *f;
    int n;
} *x, *y, u, v;
```

Andersen's Points-to Graph

Steensgaard's Points-to Graph
Non-Distributivity of Points-to Analysis

May Points-to

\[ n_1 \quad n_2 \xrightarrow{x = \&z} n_3 \xrightarrow{y = \&w} n_4 \xrightarrow{*x = y} \]

Must Points-to

\[ n_1 \quad n_2 \xrightarrow{b = \&c} n_3 \xrightarrow{c = \&d} n_4 \xrightarrow{a = *b} \]

Dec 2019
Non-Distributivity of Points-to Analysis

May Points-to

Must Points-to

$z \rightarrow w$ is spurious
Non-Distributivity of Points-to Analysis

May Points-to

\[ n_1 \]
\[ x = \& z \]
\[ n_2 \]
\[ *x = y \]
\[ n_4 \]
\[ n_3 \]
\[ y = \& w \]

\[ z \rightarrow w \] is spurious

Must Points-to

\[ n_1 \]
\[ b = \& c \]
\[ n_2 \]
\[ c = \& d \]
\[ n_3 \]
\[ b = \& e \]
\[ e = \& d \]
\[ n_4 \]
\[ a = *b \]

\[ a \rightarrow d \] is missing