# CS738: Advanced Compiler Optimizations

# Pointer Analysis

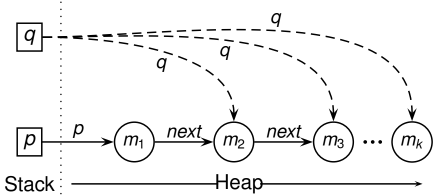
Amey Karkare
karkare@cse.iitk.ac.in

http://www.cse.iitk.ac.in/~karkare/cs738
Department of CSE, IIT Kanpur

# Why Pointer Analysis?

- Static analysis of pointers & references

| | |
|---|---|
| $S1.$ | $\ldots$ |
| $S2.$ | $q = p;$ |
| $S3.$ | *do* $\{$ |
| $S4.$ | $\quad q = q.next;$ |
| $S5.$ | $\}$ *while* $(\ldots)$ |
| $S6.$ | $\mathbf{p.data} = \mathbf{r1};$ |
| $S7.$ | $q.data = q.data + r2;$ |
| $S8.$ | $\mathbf{p.data} = \mathbf{r1};$ |
| $S9.$ | $r3 = p.data + r2;$ |
| $S10.$ | $\ldots$ |



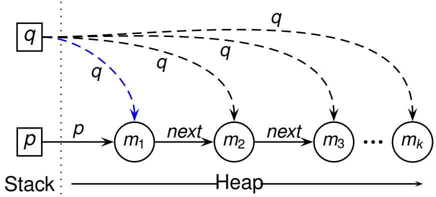

Superimposition of memory graphs after *do-while* loop

*p* and *q* are definitely not aliases statement S6 onwards.
Statement S8 **is** redundant.

# Why Pointer Analysis?

- Static analysis of pointers & references

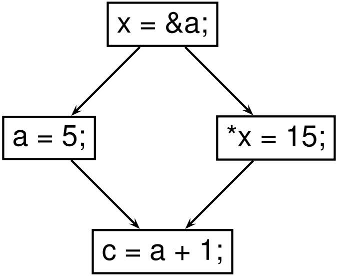
```
S1.   ...
S2.   q = p;
S3.   while (...) {
S4.       q = q.next;
S5.   }
S6.   p.data = r1;
S7.   q.data = q.data + r2;
S8.   p.data = r1;
S9.   r3 = p.data + r2;
S10.  ...
```




Superimposition of memory graphs after *while* loop

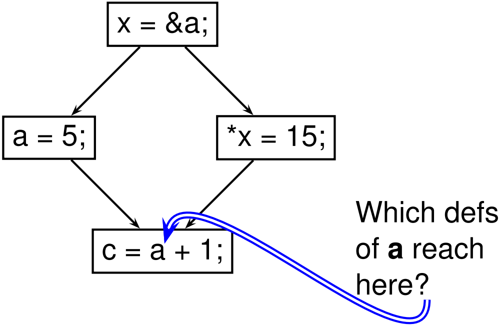
*p* and *q* may be aliases statement S6 onwards.
Statement S8 **is not** redundant.

# Why Pointer Analysis?




Reaching definitions analysis

# Why Pointer Analysis?




Reaching definitions analysis

# Flow Sensitivity in Data Flow Analysis

- Flow Sensitive Analysis

# Flow Sensitivity in Data Flow Analysis

- Flow Sensitive Analysis
  - Order of execution: Determined by the semantics of language

# Flow Sensitivity in Data Flow Analysis

- Flow Sensitive Analysis
  - Order of execution: Determined by the semantics of language
  - Point-specific information computed at each program point within a procedure

# Flow Sensitivity in Data Flow Analysis

- Flow Sensitive Analysis
  - Order of execution: Determined by the semantics of language
  - Point-specific information computed at each program point within a procedure
  - A statement can "override" information computed by a previous statement

# Flow Sensitivity in Data Flow Analysis

- Flow Sensitive Analysis
  - Order of execution: Determined by the semantics of language
  - Point-specific information computed at each program point within a procedure
  - A statement can “override” information computed by a previous statement
    - *Kill* component in the flow function

# Flow Sensitivity in Data Flow Analysis

- Flow Insensitive Analysis

# Flow Sensitivity in Data Flow Analysis

- Flow Insensitive Analysis
  - Order of execution: Statements are assumed to execute in any order

# Flow Sensitivity in Data Flow Analysis

- Flow Insensitive Analysis
  - Order of execution: Statements are assumed to execute in any order
  - As a result, all the program points in a procedure receive identical data flow information.

# Flow Sensitivity in Data Flow Analysis

- Flow Insensitive Analysis
  - Order of execution: Statements are assumed to execute in any order
  - As a result, all the program points in a procedure receive identical data flow information.
    - “Summary” for the procedure

# Flow Sensitivity in Data Flow Analysis

- Flow Insensitive Analysis
  - Order of execution: Statements are assumed to execute in any order
  - As a result, all the program points in a procedure receive identical data flow information.
    - “Summary” for the procedure
    - Safe approximation of flow-sensitive point-specific information for any point, for any given execution order

# Flow Sensitivity in Data Flow Analysis

- Flow Insensitive Analysis
  - Order of execution: Statements are assumed to execute in any order
  - As a result, all the program points in a procedure receive identical data flow information.
    - "Summary" for the procedure
    - Safe approximation of flow-sensitive point-specific information for any point, for any given execution order
  - A statement can not "override" information computed by another statement

# Flow Sensitivity in Data Flow Analysis

- Flow Insensitive Analysis
  - Order of execution: Statements are assumed to execute in any order
  - As a result, all the program points in a procedure receive identical data flow information.
    - "Summary" for the procedure
    - Safe approximation of flow-sensitive point-specific information for any point, for any given execution order
  - A statement can not "override" information computed by another statement
    - *NO* Kill component in the flow function

# Flow Sensitivity in Data Flow Analysis

- Flow Insensitive Analysis
  - Order of execution: Statements are assumed to execute in any order
  - As a result, all the program points in a procedure receive identical data flow information.
    - "Summary" for the procedure
    - Safe approximation of flow-sensitive point-specific information for any point, for any given execution order
  - A statement can not "override" information computed by another statement
    - *NO* Kill component in the flow function
    - If statement *s* kills some data flow information, there is an alternate path that excludes *s*

# Examples of Flow Insensitive Analyses

- Type checking, Type inferencing

# Examples of Flow Insensitive Analyses

- Type checking, Type inferencing
  - Compute/Verify type of a variable/expression

# Examples of Flow Insensitive Analyses

- Type checking, Type inferencing
  - Compute/Verify type of a variable/expression
- Address taken analysis

# Examples of Flow Insensitive Analyses

- Type checking, Type inferencing
  - Compute/Verify type of a variable/expression
- Address taken analysis
  - Which variables have their addresses taken?

# Examples of Flow Insensitive Analyses

- Type checking, Type inferencing
  - Compute/Verify type of a variable/expression
- Address taken analysis
  - Which variables have their addresses taken?
  - A very simple form of pointer analysis

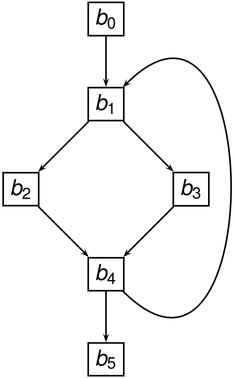
# Examples of Flow Insensitive Analyses

- Type checking, Type inferencing
  - Compute/Verify type of a variable/expression
- Address taken analysis
  - Which variables have their addresses taken?
  - A very simple form of pointer analysis
- Side effects analysis

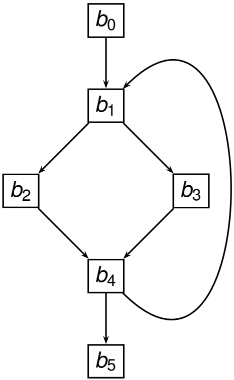
# Examples of Flow Insensitive Analyses

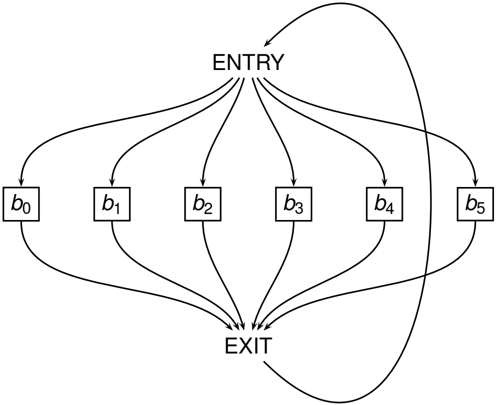
- Type checking, Type inferencing
  - Compute/Verify type of a variable/expression
- Address taken analysis
  - Which variables have their addresses taken?
  - A very simple form of pointer analysis
- Side effects analysis
  - Does a procedure modify address / global variable / reference parameter / . . . ?

# Realizing Flow Insensitivity

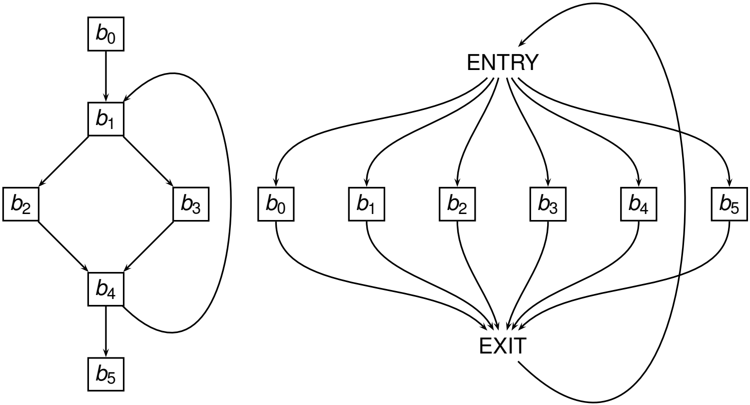


# Realizing Flow Insensitivity



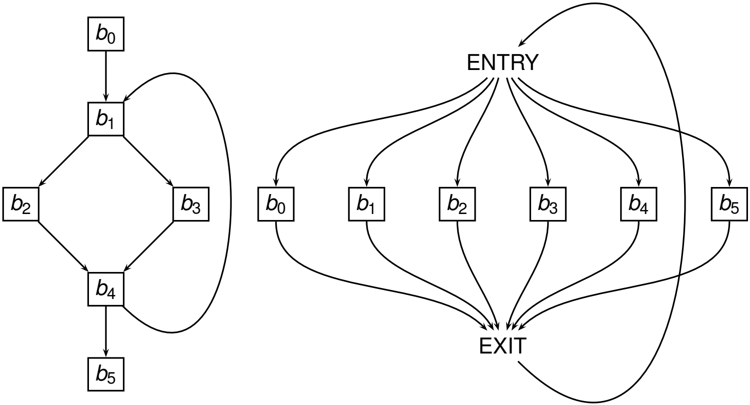



# Realizing Flow Insensitivity




*Allows arbitrary compositions of flow functions in any order* ⇒ *Flow insensitivity*

# Realizing Flow Insensitivity




*In practice, dependent constraints are collected in a global repository in one pass and solved independently*

# Alias Analysis vs. Points-to Analysis

| | Points-to Analysis | Alias Analysis |
|---|---|---|
| | x = &a | x = a |
| | x points-to a | x and a are aliases |

# Alias Analysis vs. Points-to Analysis

| | **Points-to Analysis** | **Alias Analysis** |
|---|---|---|
| | x = &a | x = a |
| | x points-to a | x and a are aliases |
| | $x \rightarrow a$ | $x \equiv a$ |

# Alias Analysis vs. Points-to Analysis

| | Points-to Analysis | Alias Analysis |
|---|---|---|
| | x = &a | x = a |
| | x points-to a | x and a are aliases |
| | $x \rightarrow a$ | $x \equiv a$ |
| Reflexive? | | |

# Alias Analysis vs. Points-to Analysis

| | **Points-to Analysis** | **Alias Analysis** |
|---|---|---|
| | x = &a | x = a |
| | x points-to a | x and a are aliases |
| | x $\rightarrow$ a | x $\equiv$ a |
| Reflexive? | No | Yes |

# Alias Analysis vs. Points-to Analysis

| | **Points-to Analysis** | **Alias Analysis** |
|---|---|---|
| | x = &a | x = a |
| | x points-to a | x and a are aliases |
| | $x \rightarrow a$ | $x \equiv a$ |
| Reflexive? | No | Yes |
| Symmetric? | | |

# Alias Analysis vs. Points-to Analysis

| | **Points-to Analysis** | **Alias Analysis** |
|---|---|---|
| | x = &a | x = a |
| | x points-to a | x and a are aliases |
| | $x \rightarrow a$ | $x \equiv a$ |
| Reflexive? | No | Yes |
| Symmetric? | No | Yes |

# Alias Analysis vs. Points-to Analysis

| | **Points-to Analysis** | **Alias Analysis** |
|---|---|---|
| | x = &a | x = a |
| | x points-to a | x and a are aliases |
| | x $\rightarrow$ a | x $\equiv$ a |
| Reflexive? | No | Yes |
| Symmetric? | No | Yes |
| Transitive? | | |

# Alias Analysis vs. Points-to Analysis

| | Points-to Analysis | Alias Analysis |
|---|---|---|
| | x = &a | x = a |
| | x points-to a | x and a are aliases |
| | $x \rightarrow a$ | $x \equiv a$ |
| Reflexive? | No | Yes |
| Symmetric? | No | Yes |
| Transitive? | No | |

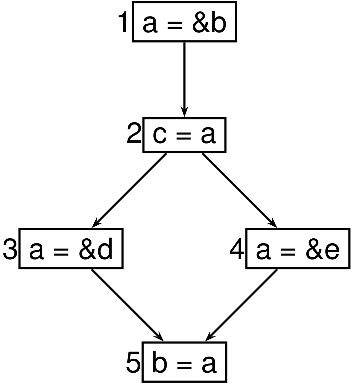
# Alias Analysis vs. Points-to Analysis

| | Points-to Analysis | Alias Analysis |
|---|---|---|
| | x = &a | x = a |
| | x points-to a | x and a are aliases |
| | $x \rightarrow a$ | $x \equiv a$ |
| Reflexive? | No | Yes |
| Symmetric? | No | Yes |
| Transitive? | No | Must alias: Yes,<br>May alias: No |

# Andersen's Flow Insensitive Points-to Analysis

- Subset based analysis

| Program | Constraints | Points-to Graph |
|---|---|---|

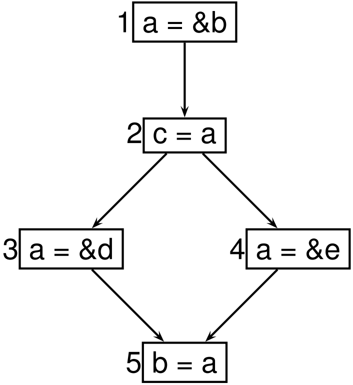



| # | **Constraint** |
|---|---|
| | |

# Andersen's Flow Insensitive Points-to Analysis

- Subset based analysis
- $P_{lhs} \supseteq P_{rhs}$

| Program | Constraints | Points-to Graph |
|---|---|---|

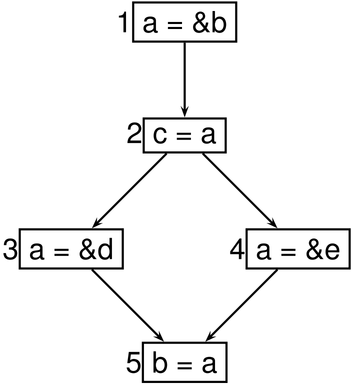



| # | **Constraint** |
|---|---|
| | |

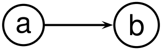
# Andersen's Flow Insensitive Points-to Analysis

- Subset based analysis
- $P_{lhs} \supseteq P_{rhs}$

| Program | Constraints | Points-to Graph |
|---|---|---|



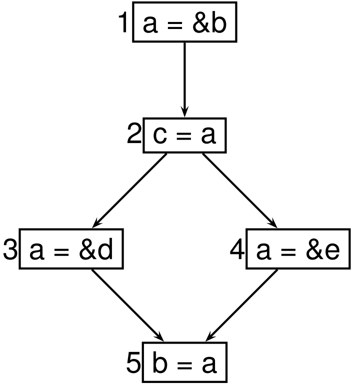

| # | Constraint |
|---|---|
| 1 | $P_a \supseteq \{b\}$ |



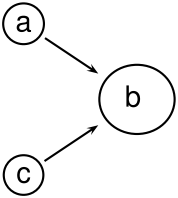
# Andersen's Flow Insensitive Points-to Analysis

- Subset based analysis
- $P_{lhs} \supseteq P_{rhs}$

| Program | Constraints | Points-to Graph |
|---|---|---|



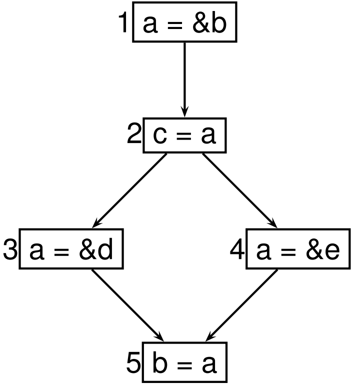

| # | **Constraint** |
|---|---|
| 1 | $P_a \supseteq \{b\}$ |
| 2 | $P_c \supseteq P_a$ |



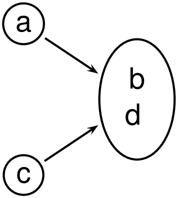
# Andersen's Flow Insensitive Points-to Analysis

- Subset based analysis
- $P_{lhs} \supseteq P_{rhs}$

| Program | Constraints | Points-to Graph |
|---|---|---|



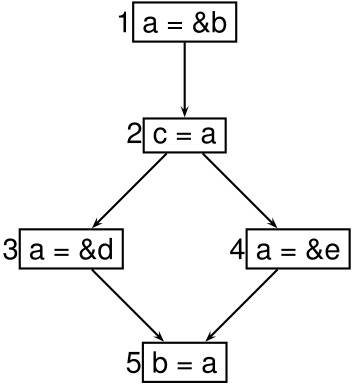

| # | **Constraint** |
|---|---|
| 1 | $P_a \supseteq \{b\}$ |
| 2 | $P_c \supseteq P_a$ |
| 3 | $P_a \supseteq \{d\}$ |



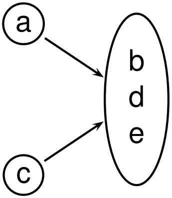
# Andersen's Flow Insensitive Points-to Analysis

- Subset based analysis
- $P_{lhs} \supseteq P_{rhs}$

| Program | Constraints | Points-to Graph |
|---|---|---|



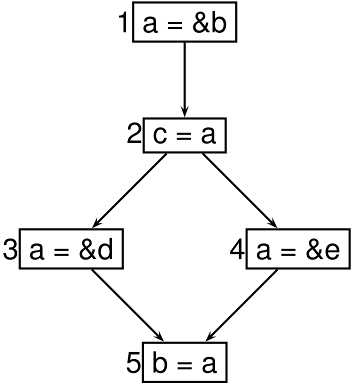

| # | Constraint |
|---|---|
| 1 | $P_a \supseteq \{b\}$ |
| 2 | $P_c \supseteq P_a$ |
| 3 | $P_a \supseteq \{d\}$ |
| 4 | $P_a \supseteq \{e\}$ |



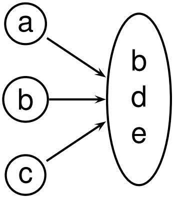
# Andersen's Flow Insensitive Points-to Analysis

- Subset based analysis
- $P_{lhs} \supseteq P_{rhs}$

| Program | Constraints | Points-to Graph |
|---|---|---|



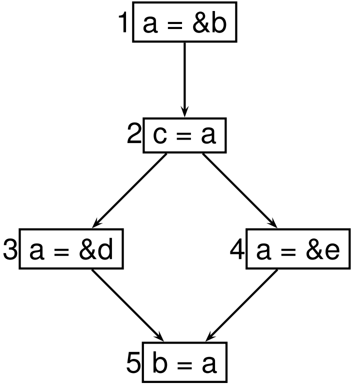

| # | **Constraint** |
|---|---|
| 1 | $P_a \supseteq \{b\}$ |
| 2 | $P_c \supseteq P_a$ |
| 3 | $P_a \supseteq \{d\}$ |
| 4 | $P_a \supseteq \{e\}$ |
| 5 | $P_b \supseteq P_a$ |



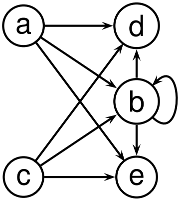
# Andersen's Flow Insensitive Points-to Analysis

- Subset based analysis
- $P_{lhs} \supseteq P_{rhs}$

| Program | Constraints | Points-to Graph |
|---|---|---|



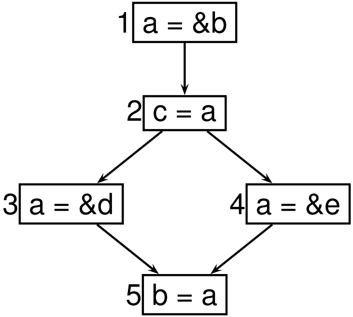

| # | **Constraint** |
|---|---|
| 1 | $P_a \supseteq \{b\}$ |
| 2 | $P_c \supseteq P_a$ |
| 3 | $P_a \supseteq \{d\}$ |
| 4 | $P_a \supseteq \{e\}$ |
| 5 | $P_b \supseteq P_a$ |



# Steensgaard's Flow Insensitive Points-to Analysis

- Equality based analysis: $P_{lhs} \equiv P_{rhs}$
- Only one Points-to successor at any time, merge (potential) multiple successors

| Program | Constraints | Points-to Graph |
|---|---|---|

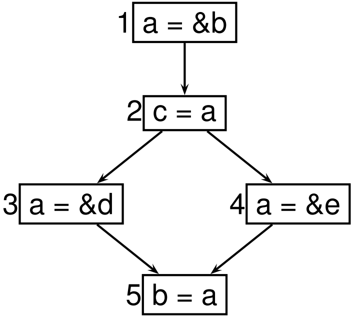



| # | **Constraint** |
|---|---|
| | |

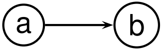
# Steensgaard's Flow Insensitive Points-to Analysis

- Equality based analysis: $P_{lhs} \equiv P_{rhs}$
- Only one Points-to successor at any time, merge (potential) multiple successors

| Program | Constraints | Points-to Graph |
|---|---|---|



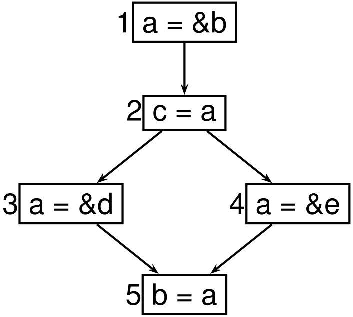

| # | Constraint |
|---|---|
| 1 | $P_a \supseteq \{b\}$ |



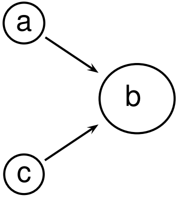
# Steensgaard's Flow Insensitive Points-to Analysis

- Equality based analysis: $P_{lhs} \equiv P_{rhs}$
- Only one Points-to successor at any time, merge (potential) multiple successors

| Program | Constraints | Points-to Graph |
|---|---|---|



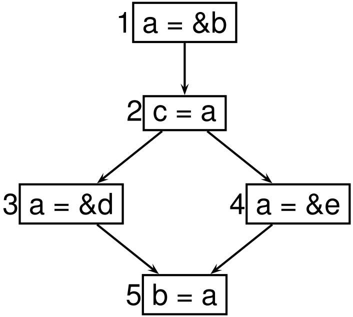

| # | Constraint |
|---|---|
| 1 | $P_a \supseteq \{b\}$ |
| 2 | MERGE($P_c, P_a$) |



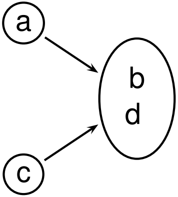
# Steensgaard's Flow Insensitive Points-to Analysis

- Equality based analysis: $P_{lhs} \equiv P_{rhs}$
- Only one Points-to successor at any time, merge (potential) multiple successors

| Program | Constraints | Points-to Graph |
|---|---|---|



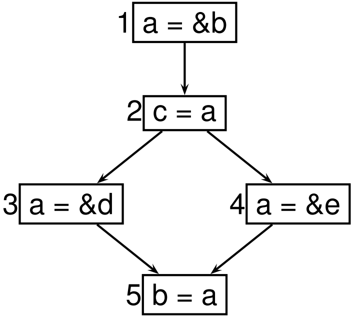

| # | **Constraint** |
|---|---|
| 1 | $P_a \supseteq \{b\}$ |
| 2 | MERGE($P_c, P_a$) |
| 3 | $P_a \supseteq \{d\}$ |



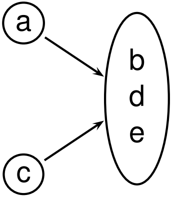
# Steensgaard's Flow Insensitive Points-to Analysis

- Equality based analysis: $P_{lhs} \equiv P_{rhs}$
- Only one Points-to successor at any time, merge (potential) multiple successors

| Program | Constraints | Points-to Graph |
|---|---|---|



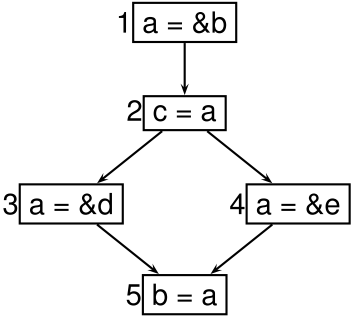

| # | Constraint |
|---|---|
| 1 | $P_a \supseteq \{b\}$ |
| 2 | MERGE($P_c, P_a$) |
| 3 | $P_a \supseteq \{d\}$ |
| 4 | $P_a \supseteq \{e\}$ |



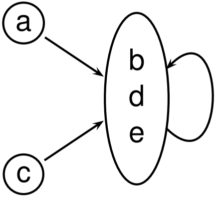
# Steensgaard's Flow Insensitive Points-to Analysis

- Equality based analysis: $P_{lhs} \equiv P_{rhs}$
- Only one Points-to successor at any time, merge (potential) multiple successors

| Program | Constraints | Points-to Graph |
|---|---|---|



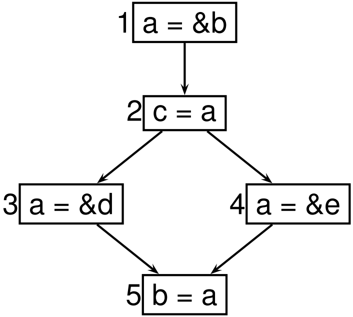

| # | **Constraint** |
|---|---|
| 1 | $P_a \supseteq \{b\}$ |
| 2 | MERGE($P_c, P_a$) |
| 3 | $P_a \supseteq \{d\}$ |
| 4 | $P_a \supseteq \{e\}$ |
| 5 | MERGE($P_b, P_a$) |



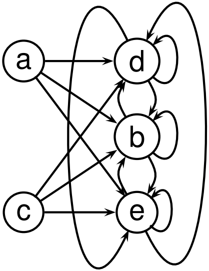
# Steensgaard's Flow Insensitive Points-to Analysis

- Equality based analysis: $P_{lhs} \equiv P_{rhs}$
- Only one Points-to successor at any time, merge (potential) multiple successors

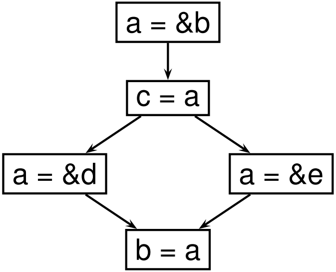
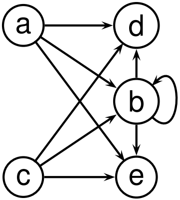
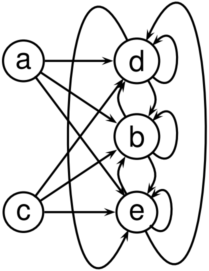
| Program | Constraints | Points-to Graph |
|---|---|---|




| # | **Constraint** |
|---|---|
| 1 | $P_a \supseteq \{b\}$ |
| 2 | MERGE($P_c$, $P_a$) |
| 3 | $P_a \supseteq \{d\}$ |
| 4 | $P_a \supseteq \{e\}$ |
| 5 | MERGE($P_b$, $P_a$) |



# Comparing Anderson's and Steensgaard's Analyses

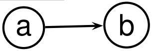
| Program | Subset based Points-to Graph | Equality based Points-to Graph |
| --- | --- | --- |
|  |  |  |

# Comparing Anderson's and Steensgaard's Analyses

```
a = &b;
```

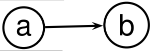
# Comparing Anderson's and Steensgaard's Analyses

```
a = &b;
```



# Comparing Anderson's and Steensgaard's Analyses

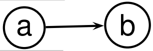
```
a = &b;
```



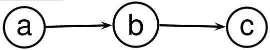

```
b = &c;
```

# Comparing Anderson's and Steensgaard's Analyses

a = &b;

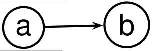



b = &c;

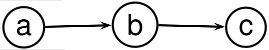


# Comparing Anderson's and Steensgaard's Analyses

```
a = &b;
```



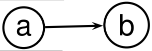

```
b = &c;
```



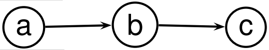

```
d = &e;
```

# Comparing Anderson's and Steensgaard's Analyses

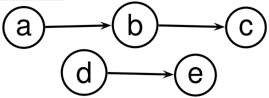
a = &b;




b = &c;

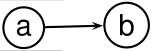



d = &e;

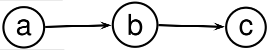


# Comparing Anderson's and Steensgaard's Analyses

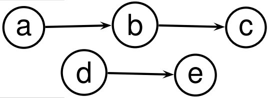
```
a = &b;
```




```
b = &c;
```



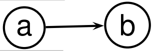

```
d = &e;
```



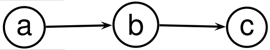

```
a = &d;
```

# Comparing Anderson's and Steensgaard's Analyses

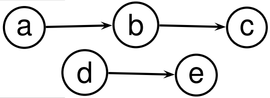
```
a = &b;
```




```
b = &c;
```

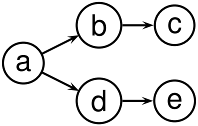



```
d = &e;
```

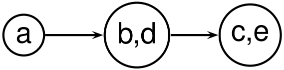



```
a = &d;
```

Subset based



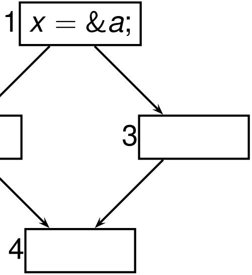

Equality based



# Pointer Indirection Constraints

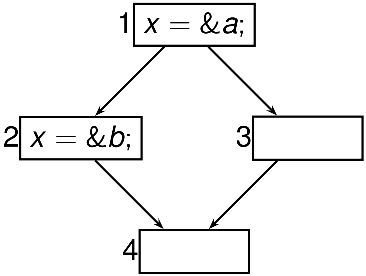
| Stmt | Subset based | Equality based |
|---|---|---|
| a = *b | $P_a \supseteq P_c, \forall c \in P_b$ | MERGE($P_a, P_c$), $\forall c \in P_b$ |
| *a = b | $P_c \supseteq P_b, \forall c \in P_a$ | MERGE($P_b, P_c$), $\forall c \in P_a$ |

# Must Points-to Analysis



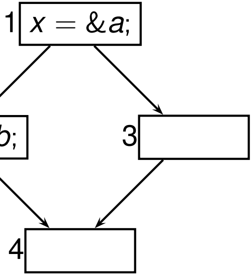

- *x definitely* points-to *a* at various points in the program
- $x \xrightarrow{D} a$

# May Points-to Analysis



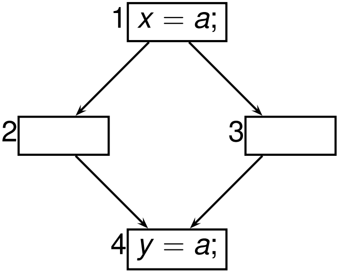

- At OUT of 2, *x* definitely points-to *b*
- At OUT of 3, *x* definitely points-to *a*
- At IN of 4, *x* *possibly* points-to *a* (or *b*)
  - $x \xrightarrow{P} a, x \xrightarrow{P} b$

# May Points-to Analysis



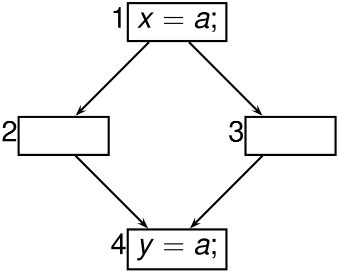

- At OUT of 2, $x$ definitely points-to $b$
- At OUT of 3, $x$ definitely points-to $a$
- At IN of 4, $x$ *possibly* points-to $a$ (or $b$)
  - $x \xrightarrow{P} \{a, b\}$

# Must Alias Analysis



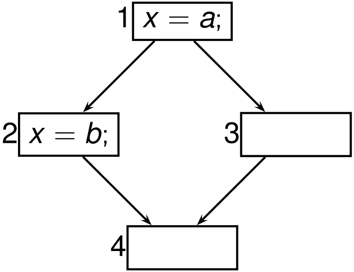

- $x$ and $a$ always refer to same memory location
- $x \stackrel{D}{\equiv} a$

# Must Alias Analysis



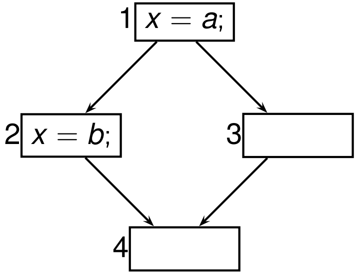

- $x$ and $a$ always refer to same memory location
- $x \stackrel{D}{\equiv} a$
- $x, y$ and $a$ refer to same location at OUT of 4.
- $x \stackrel{D}{\equiv} y \stackrel{D}{\equiv} a$

# May Alias Analysis



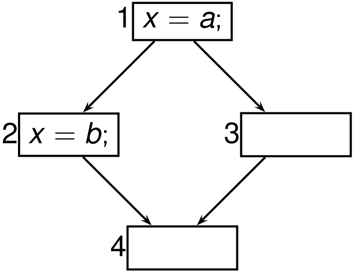

- At OUT of 2, $x$ and $b$ are must aliases
- At OUT of 3, $x$ and $a$ are must aliases
- At IN of 4, $x$ can *possibly* be aliased with either $a$ (or $b$)
  - $x \overset{P}{\equiv} a$, $x \overset{P}{\equiv} b$

# May Alias Analysis




- At OUT of 2, $x$ and $b$ are must aliases
- At OUT of 3, $x$ and $a$ are must aliases
- At IN of 4, $x$ can *possibly* be aliased with either $a$ (or $b$)
  - $(x, a), (x, b)$

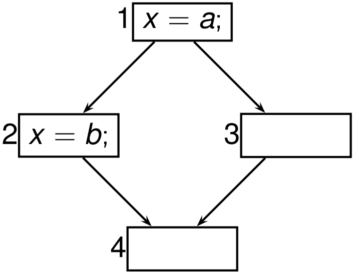
# May Alias Analysis




- At OUT of 2, *x* and *b* are must aliases
- At OUT of 3, *x* and *a* are must aliases
- At IN of 4, *x* can *possibly* be aliased with either *a* (or *b*)
  - $(x, a), (x, b)$

- If we say: $(x, a, b)$, Is it *Precise?*

# May Alias Analysis




- At OUT of 2, $x$ and $b$ are must aliases
- At OUT of 3, $x$ and $a$ are must aliases
- At IN of 4, $x$ can *possibly* be aliased with either $a$ (or $b$)
  - $(x, a), (x, b)$

- If we say: $(x, a, b)$, Is it *Precise? Safe?*

# Must Pointer Analysis

- Makes sense only for Flow Sensitive analysis

# Must Pointer Analysis

- Makes sense only for Flow Sensitive analysis
- Why?

# Must Pointer Analysis

- Makes sense only for Flow Sensitive analysis
- Why?
- Must analysis $\Rightarrow$ Flow sensitive analysis

# Must Pointer Analysis

- Makes sense only for Flow Sensitive analysis
- Why?
- Must analysis $\Rightarrow$ Flow sensitive analysis
- Flow insensitive analysis $\Rightarrow$ May analysis

# Must Pointer Analysis

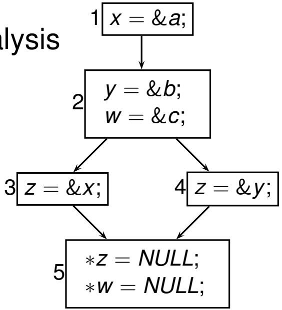
- Makes sense only for Flow Sensitive analysis
- Why?
- Must analysis $\Rightarrow$ Flow sensitive analysis
- Flow insensitive analysis $\Rightarrow$ May analysis
- Why?

# Updating Information: When Can We *Kill*?

- Never if flow insensitive analysis

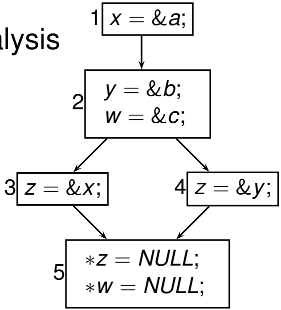
# Updating Information: When Can We *Kill*?

- Never if flow insensitive analysis
- For flow sensitive



# Updating Information: When Can We *Kill*?

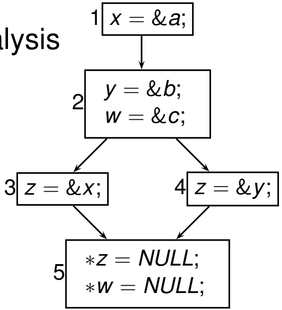
- Never if flow insensitive analysis
- For flow sensitive




- $x$, $y$ may or may not get modified in 5: *Weak* update

# Updating Information: When Can We *Kill*?

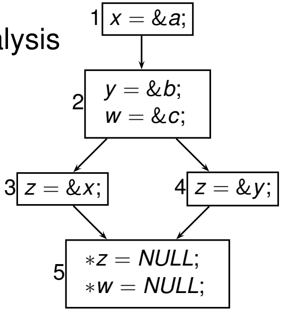
- Never if flow insensitive analysis
- For flow sensitive




- $x$, $y$ may or may not get modified in 5: *Weak* update
- $c$ definitely gets modified in 5: *Strong update*

# Updating Information: When Can We *Kill*?

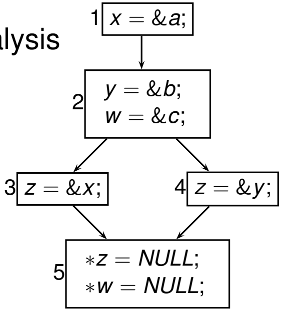
- Never if flow insensitive analysis
- For flow sensitive




- $x$, $y$ may or may not get modified in 5: *Weak* update
- $c$ definitely gets modified in 5: *Strong update*
- Must information is killed by Strong and Weak updates

# Updating Information: When Can We *Kill*?

- Never if flow insensitive analysis
- For flow sensitive




- $x$, $y$ may or may not get modified in 5: *Weak* update
- $c$ definitely gets modified in 5: *Strong update*
- Must information is killed by Strong and Weak updates
- May information is killed only by Strong updates

# Flow Functions for Points-to Analysis

- Basic statements for pointer manipulation

# Flow Functions for Points-to Analysis

- Basic statements for pointer manipulation
  - x = y

# Flow Functions for Points-to Analysis

- Basic statements for pointer manipulation
  - x = y
  - x = &y

# Flow Functions for Points-to Analysis

- Basic statements for pointer manipulation
  - x = y
  - x = &y
  - x = *y

# Flow Functions for Points-to Analysis

- Basic statements for pointer manipulation
  - x = y
  - x = &y
  - x = *y
  - *x = y

# Flow Functions for Points-to Analysis

- Basic statements for pointer manipulation
  - x = y
  - x = &y
  - x = *y
  - *x = y
- Other statements can be rewritten in terms of above

# Flow Functions for Points-to Analysis

- Basic statements for pointer manipulation
  - x = y
  - x = &y
  - x = *y
  - *x = y
- Other statements can be rewritten in terms of above
  - *x = *y $\Rightarrow$ t = *y, *x = t

# Flow Functions for Points-to Analysis

- Basic statements for pointer manipulation
  - x = y
  - x = &y
  - x = *y
  - *x = y
- Other statements can be rewritten in terms of above
  - *x = *y $\Rightarrow$ t = *y, *x = t
  - x = NULL $\Rightarrow$ treat NULL as a special variable

# Flow Functions for Points-to Analysis

- Basic statements for pointer manipulation
  - x = y
  - x = &y
  - x = *y
  - *x = y
- Other statements can be rewritten in terms of above
  - *x = *y $\Rightarrow$ t = *y, *x = t
  - x = NULL $\Rightarrow$ treat NULL as a special variable
- $OUT = IN - kill \cup gen$

# Flow Functions for Points-to Analysis

- Basic statements for pointer manipulation
  - x = y
  - x = &y
  - x = *y
  - *x = y
- Other statements can be rewritten in terms of above
  - *x = *y $\Rightarrow$ t = *y, *x = t
  - x = NULL $\Rightarrow$ treat NULL as a special variable
- $OUT = IN - kill \cup gen$
  - with a twist!

# Flow Function: x = y

$$\begin{aligned} \text{May}_{gen} &= \{x \rightarrow p \mid y \rightarrow p \in \text{May}_{IN}\} \\ \text{May}_{kill} &= \bigcup_{p \in Vars} \{x \rightarrow p\} \end{aligned}$$

# Flow Function: x = y

$$
\begin{aligned}
\text{May}_{gen} &= \{x \to p \mid y \to p \in \text{May}_{IN}\} \\
\text{May}_{kill} &= \bigcup_{p \in Vars} \{x \to p\}
\end{aligned}
$$

$$
\begin{aligned}
\text{Must}_{gen} &= \{x \to p \mid y \to p \in \text{Must}_{IN}\} \\
\text{Must}_{kill} &= \bigcup_{p \in Vars} \{x \to p\}
\end{aligned}
$$

# Flow Function: x = &y

$$\begin{aligned} \mathrm{May}_{gen} &= \{x \to y\} \\ \mathrm{May}_{kill} &= \bigcup_{p \in Vars} \{x \to p\} \end{aligned}$$

# Flow Function: x = &y

$$
\begin{aligned}
\text{May}_{gen} &= \{x \to y\} \\
\text{May}_{kill} &= \bigcup_{p \in Vars} \{x \to p\}
\end{aligned}
$$

$$
\begin{aligned}
\text{Must}_{gen} &= \{x \to y\} \\
\text{Must}_{kill} &= \bigcup_{p \in Vars} \{x \to p\}
\end{aligned}
$$

# Flow Function: x = *y

$$\begin{aligned} \text{May}_{gen} &= \{x \to p \mid y \to p' \in \text{May}_{IN} \text{ and } p' \to p \in \text{May}_{IN}\} \\ \text{May}_{kill} &= \bigcup_{p \in Vars} \{x \to p\} \end{aligned}$$

# Flow Function: x = *y

$$\begin{aligned}
May_{gen} &= \{x \rightarrow p \mid y \rightarrow p' \in May_{IN} \text{ and } p' \rightarrow p \in May_{IN}\} \\
May_{kill} &= \bigcup_{p \in Vars} \{x \rightarrow p\}
\end{aligned}$$

$$\begin{aligned}
Must_{gen} &= \{x \rightarrow p \mid y \rightarrow p' \in Must_{IN} \text{ and } p' \rightarrow p \in Must_{IN}\} \\
Must_{kill} &= \bigcup_{p \in Vars} \{x \rightarrow p\}
\end{aligned}$$

# Flow Function: *x = y

$$
\begin{aligned}
\text{May}_{gen} &= \{p \rightarrow p' \mid x \rightarrow p \in \text{May}_{IN}, y \rightarrow p' \in \text{May}_{IN}\} \\
\text{May}_{kill} &= \bigcup_{p' \in Vars} \{p \rightarrow p' \mid x \rightarrow p \in \text{Must}_{IN}\}
\end{aligned}
$$

# Flow Function: *x = y

$$\begin{aligned} May_{gen} &= \{p \to p' \mid x \to p \in May_{IN}, y \to p' \in May_{IN}\} \\ May_{kill} &= \bigcup_{p' \in Vars} \{p \to p' \mid x \to p \in Must_{IN}\} \end{aligned}$$

$$\begin{aligned} Must_{gen} &= \{p \to p' \mid x \to p \in Must_{IN}, y \to p' \in Must_{IN}\} \\ Must_{kill} &= \bigcup_{p' \in Vars} \{p \to p' \mid x \to p \in May_{IN}\} \end{aligned}$$

# Flow Function: *x = y

$$\text{May}_{gen} = \{p \to p' \mid x \to p \in \text{May}_{IN}, y \to p' \in \text{May}_{IN}\}$$

$$\text{May}_{kill} = \bigcup_{p' \in Vars} \{p \to p' \mid x \to p \in \text{Must}_{IN}\}$$

*Strong update!!*

$$\text{Must}_{gen} = \{p \to p' \mid x \to p \in \text{Must}_{IN}, y \to p' \in \text{Must}_{IN}\}$$

$$\text{Must}_{kill} = \bigcup_{p' \in Vars} \{p \to p' \mid x \to p \in \text{May}_{IN}\}$$

*Weak update!!*

# Summarizing Flow Functions

- May Points-To analysis

# Summarizing Flow Functions

- May Points-To analysis
  - A points-to pair should be removed only if it must be removed along all paths

# Summarizing Flow Functions

- May Points-To analysis
  - A points-to pair should be removed only if it must be removed along all paths
  - $\Rightarrow$ should remove only strong updates

# Summarizing Flow Functions

- May Points-To analysis
  - A points-to pair should be removed only if it must be removed along all paths
  - $\Rightarrow$ should remove only strong updates
  - $\Rightarrow$ should kill using Must Points-To information

# Summarizing Flow Functions

- May Points-To analysis
  - A points-to pair should be removed only if it must be removed along all paths
  - $\Rightarrow$ should remove only strong updates
  - $\Rightarrow$ should kill using Must Points-To information
- Must Points-To analysis

# Summarizing Flow Functions

- May Points-To analysis
  - A points-to pair should be removed only if it must be removed along all paths
  - $\Rightarrow$ should remove only strong updates
  - $\Rightarrow$ should kill using Must Points-To information
- Must Points-To analysis
  - A points-to pair should be removed if it can be removed along some path

# Summarizing Flow Functions

- May Points-To analysis
  - A points-to pair should be removed only if it must be removed along all paths
  - $\Rightarrow$ should remove only strong updates
  - $\Rightarrow$ should kill using Must Points-To information
- Must Points-To analysis
  - A points-to pair should be removed if it can be removed along some path
  - $\Rightarrow$ should remove all weak updates

# Summarizing Flow Functions

- May Points-To analysis
  - A points-to pair should be removed only if it must be removed along all paths
  - $\Rightarrow$ should remove only strong updates
  - $\Rightarrow$ should kill using Must Points-To information
- Must Points-To analysis
  - A points-to pair should be removed if it can be removed along some path
  - $\Rightarrow$ should remove all weak updates
  - $\Rightarrow$ should kill using May Points-To information

# Summarizing Flow Functions

- May Points-To analysis
  - A points-to pair should be removed only if it must be removed along all paths
  - $\Rightarrow$ should remove only strong updates
  - $\Rightarrow$ should kill using Must Points-To information
- Must Points-To analysis
  - A points-to pair should be removed if it can be removed along some path
  - $\Rightarrow$ should remove all weak updates
  - $\Rightarrow$ should kill using May Points-To information
- Must Points-To $\subseteq$ May Points-To

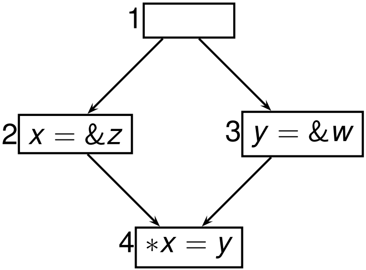
# Safe Approximations for May and Must Points-to

- A pointer variable

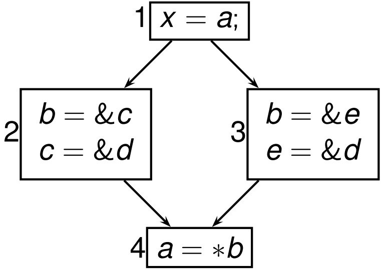
| | **May** | **Must** |
|---|---|---|
| **Points-to** | points to every possible location | points to nothing |
| **Alias** | aliased to every other pointer variable | only to itself |

# Non-Distributivity of Points-to Analysis

**May Information**

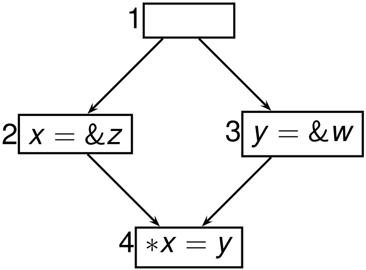



**Must Information**



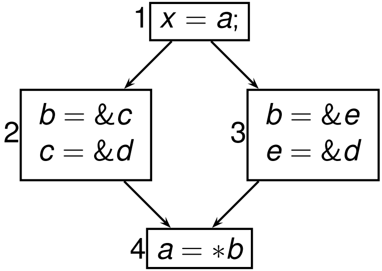
# Non-Distributivity of Points-to Analysis

**May Information**



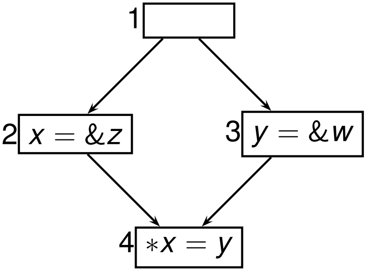

$z \rightarrow w$ is spurious

**Must Information**



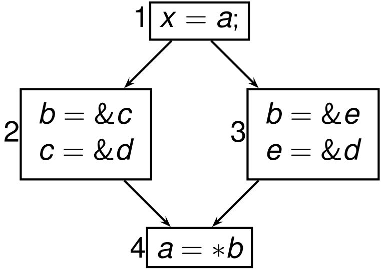
# Non-Distributivity of Points-to Analysis

**May Information**




$z \rightarrow w$ is spurious

**Must Information**




$a \rightarrow d$ is missing