CS738: Advanced Compiler Optimizations

Pointer Analysis

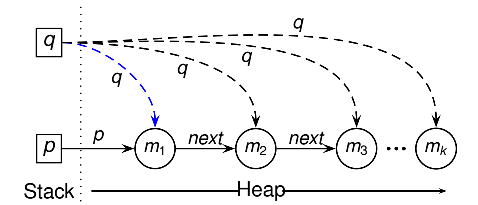
Amey Karkare
karkare@cse.iitk.ac.in

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

Why Pointer Analysis?

- Static analysis of pointers & references

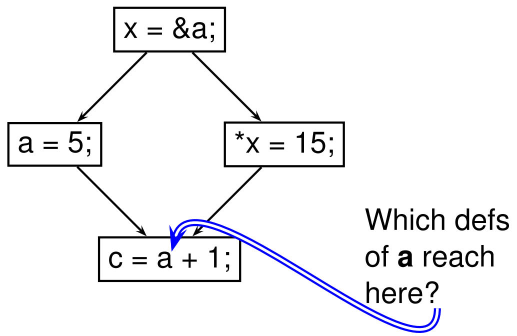
```
S1.   ...
S2.   q = p;
S3.   while (...) {
S4.       q = q.next;
S5.   }
S6.   p.data = r1;
S7.   q.data = q.data + r2;
S8.   p.data = r1;
S9.   r3 = p.data + r2;
S10.  ...
```



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

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

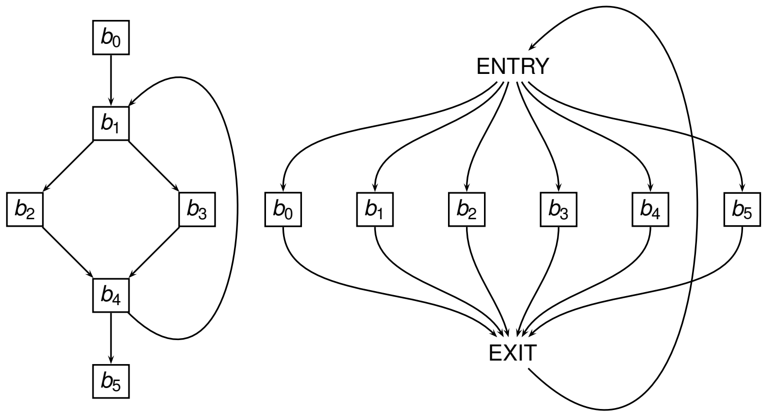
Flow Sensitivity in Data Flow Analysis

- Flow Insensitive Analysis
  - 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
  - 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



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

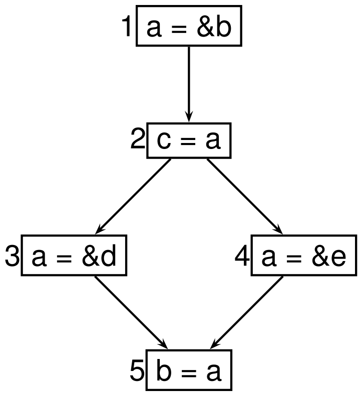
Alias Analysis vs. Points-to Analysis

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

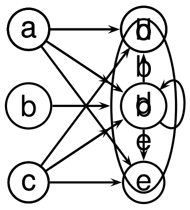
Andersen's Flow Insensitive Points-to Analysis

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

Program	Constraints	Points-to Graph



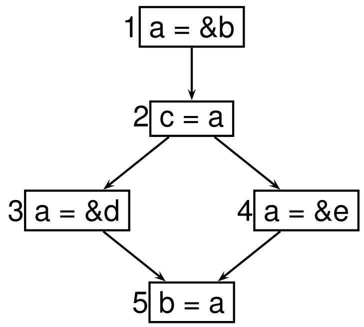
#	Constraint
1	$P_a \supseteq \{b\}$
2	$P_c \supseteq P_a$
3	$P_a \supseteq \{d\}$
4	$P_a \supseteq \{e\}$
5	$P_b \supseteq P_a$



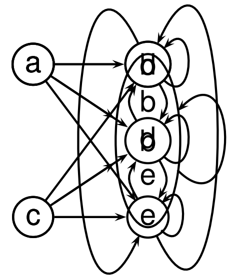
Steensgaard's Flow Insensitive Points-to Analysis

- 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

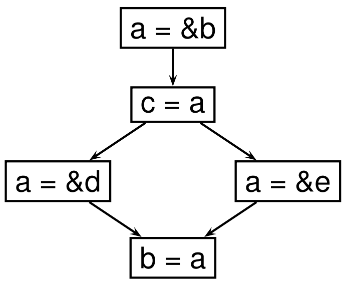


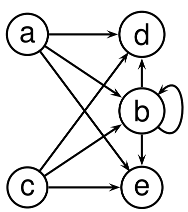
#	Constraint
1	$P_a \supseteq \{b\}$
2	MERGE(P_c, P_a)
3	$P_a \supseteq \{d\}$
4	$P_a \supseteq \{e\}$
5	MERGE(P_b, P_a)

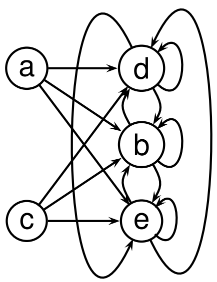


Comparing Anderson's and Steensgaard's Analyses

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

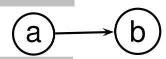




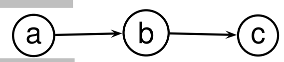


Comparing Anderson's and Steensgaard's Analyses

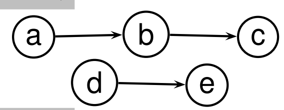
a = &b;



b = &c;

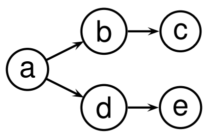


d = &e;

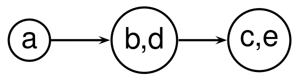


a = &d;

Subset based



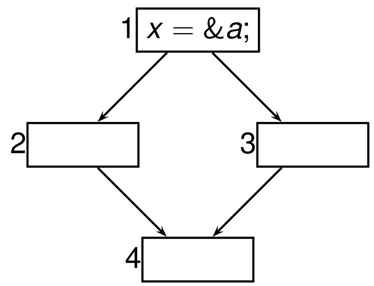
Equality based



Pointer Indirection Constraints

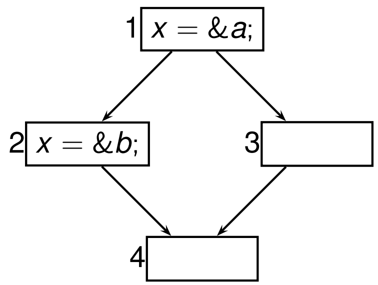
Stmt	Subset based	Equality based
a = *b	$P_a \supseteq P_c, \forall c \in P_b$	$\text{MERGE}(P_a, P_c), \forall c \in P_b$
*a = b	$P_c \supseteq P_b, \forall c \in P_a$	$\text{MERGE}(P_b, P_c), \forall c \in P_a$

Must Points-to Analysis



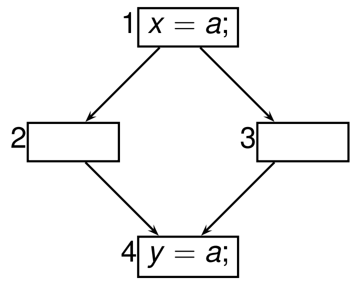
- x *definitely* points-to a at various points in the program
- $x \overset{D}{\rightarrow} a$

May Points-to Analysis



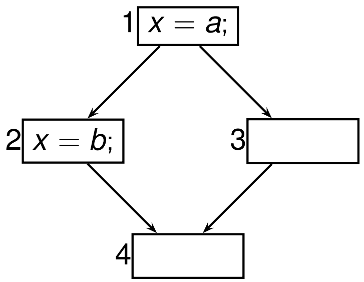
- At OUT of 2, x definitely points-to b
- At OUT of 3, x definitely points-to a
- At IN of 4, x *possibly* points-to a (or b)
  - $x \overset{P}{\rightarrow} \{a, b\}$

Must Alias Analysis



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

May Alias Analysis



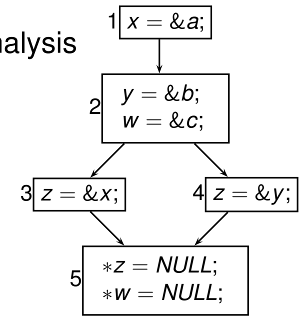
- At OUT of 2, *x* and *b* are must aliases
- At OUT of 3, *x* and *a* are must aliases
- At IN of 4, *x* can *possibly* be aliased with either *a* (or *b*)
  - $(x, a), (x, b)$
- If we say: (x, a, b), Is it *Precise? Safe?*

Must Pointer Analysis

- Makes sense only for Flow Sensitive analysis
- Why?
- Must analysis ⇒ Flow sensitive analysis
- Flow insensitive analysis ⇒ May analysis
- Why?

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

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

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

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

Flow Function: x = *y

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

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

Flow Function: *x = y

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

Strong update!!

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

Weak update!!

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

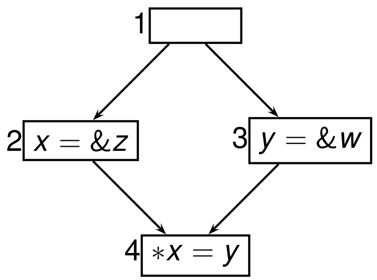
Safe Approximations for May and Must Points-to

- 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

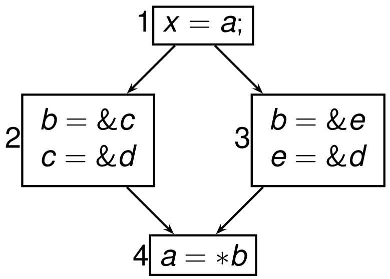
Non-Distributivity of Points-to Analysis

May Information



$z \rightarrow w$ is spurious

Must Information



$a \rightarrow d$ is missing