Liveness Based Pointer Analysis

Uday Khedker

(Joint Work with Alan Mycroft and Prashant Singh Rawat)

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

September 2012

Outline

- Introduction
- Background
- Formulating LFCPA
  (Liveness based Flow and Context Sensitive Points-to Analysis)
- Performing interprocedural analysis
- Measurements
- Conclusions

Reference:
Uday P. Khedker, Alan Mycroft, Prashant Singh Rawat. *Liveness Based Pointer Anaysis*. SAS 2012.

Part 1

Introduction

Why Pointer Analysis?

- 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 for practical usefulness
- Good pointer information could improve many applications of program analysis significantly

Pointer Analysis Musings

- Two Position Papers

- A Keynote Address

Pointer Analysis Musings

- Two Position Papers
  - 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
- A Keynote Address

Pointer Analysis Musings

- Two Position Papers
  - 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
- A Keynote Address
  - "The Worst thing that has happened to Computer Science is C because it brought pointers with it"
    Frances Allen, IITK Workshop, 2007

Pointer Analysis Musings

- Two Position Papers
  - 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
- A Keynote Address
  - "The Worst thing that has happened to Computer Science is C because it brought pointers with it"
    Frances Allen, IITK Workshop, 2007
- 2012 . . . ☹

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]

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!

The Engineering of Pointer Analysis

So what should we expect?

The Engineering of Pointer Analysis

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”

The Engineering of Pointer Analysis

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

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

The Engineering of Pointer Analysis

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

- "Fortunately many approximations exist"
- "Unfortunately too many approximations exist!"

Engineering of pointer analysis is much more dominant than its science

Pointer Analysis: Engineering or Science?

- Engineering view
- Science view

Pointer Analysis: Engineering or Science?

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

Pointer Analysis: Engineering or Science?

- Engineering view
  - Build quick approximations
  - The tyranny of (exclusive) OR!
    Precision OR Efficiency?
- Science view
  - Build clean abstractions
  - Can we harness the Genius of AND?
    Precision AND Efficiency?

The Scope of Our Points-to Analysis

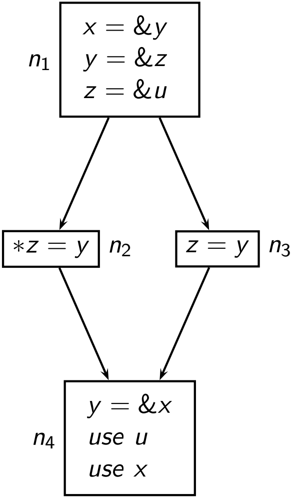
Attribute	Range of Options	Our Scope
Categories of data pointers	Static (Globals) Stack (Locals, Formals) Heap	Static (Globals) Stack (Locals, Formals)
Level	Intraprocedural, Interprocedural	Interprocedural
Flow Sensitivity	Full, Partial, None	Full
Context Sensitivity	Full, Partial, None	Full

- Heap and address escaping locals are handled conservatively
- Data flow information is safe but may be imprecise

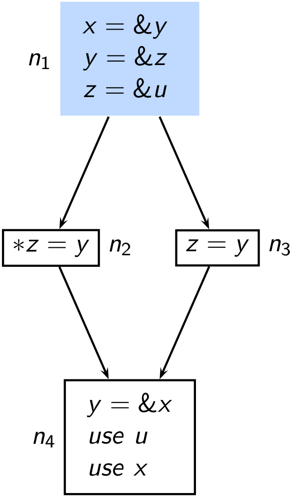
Part 2

Background

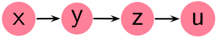
An Example of Flow Insensitive Points-to Analysis (Andersen's Approach aka Inclusion Based Approach)



An Example of Flow Insensitive Points-to Analysis (Andersen's Approach aka Inclusion Based Approach)



- x "points-to" y
- y "points-to" z
- z "points-to" u

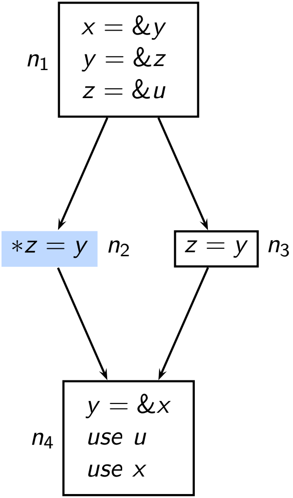


Points-to Graph

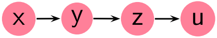
Constraints on Points-to Sets

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

An Example of Flow Insensitive Points-to Analysis (Andersen's Approach aka Inclusion Based Approach)



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

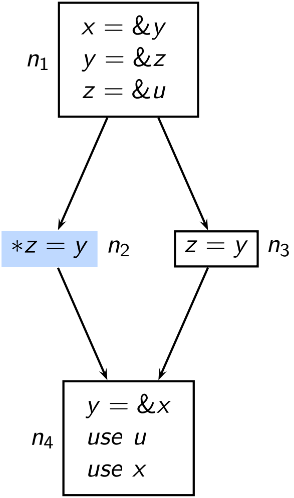


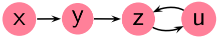
Points-to Graph

Constraints on Points-to Sets

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

An Example of Flow Insensitive Points-to Analysis (Andersen's Approach aka Inclusion Based Approach)



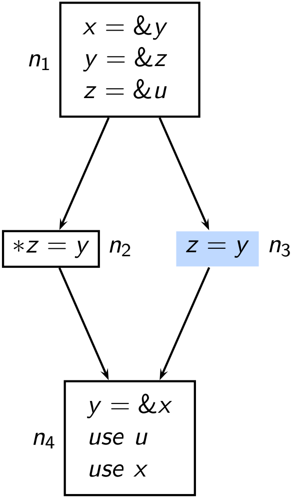


Points-to Graph

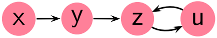
Constraints on Points-to Sets

$$\begin{aligned} P_x &\supseteq \{y\} \\ P_y &\supseteq \{z\} \\ P_z &\supseteq \{u\} \\ \forall w \in P_z,\ \ P_w &\supseteq P_y \end{aligned}$$

An Example of Flow Insensitive Points-to Analysis (Andersen's Approach aka Inclusion Based Approach)



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

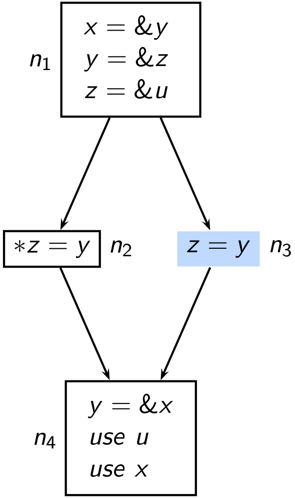


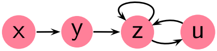
Points-to Graph

Constraints on Points-to Sets

$$\begin{aligned} P_x &\supseteq \{y\} \\ P_y &\supseteq \{z\} \\ P_z &\supseteq \{u\} \\ \forall w \in P_z,\ P_w &\supseteq P_y \\ P_z &\supseteq P_y \end{aligned}$$

An Example of Flow Insensitive Points-to Analysis (Andersen's Approach aka Inclusion Based Approach)



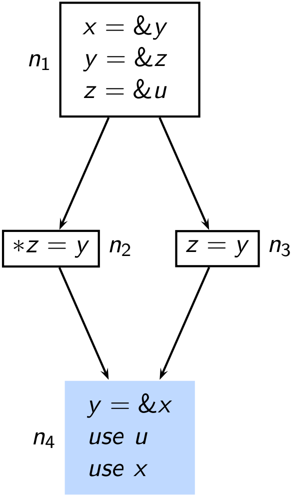


Points-to Graph

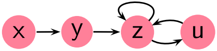
Constraints on Points-to Sets

$$\begin{aligned} P_x &\supseteq \{y\} \\ P_y &\supseteq \{z\} \\ P_z &\supseteq \{u\} \\ \forall w \in P_z,\; P_w &\supseteq P_y \\ P_z &\supseteq P_y \end{aligned}$$

An Example of Flow Insensitive Points-to Analysis (Andersen's Approach aka Inclusion Based Approach)



- y should point to x also

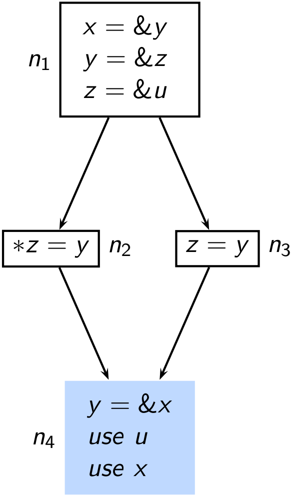


Points-to Graph

Constraints on Points-to Sets

$$
\begin{aligned}
P_x &\supseteq \{y\} \\
P_y &\supseteq \{z\} \\
P_z &\supseteq \{u\} \\
\forall w \in P_z,\ P_w &\supseteq P_y \\
P_z &\supseteq P_y \\
P_y &\supseteq \{x\}
\end{aligned}
$$

An Example of Flow Insensitive Points-to Analysis (Andersen's Approach aka Inclusion Based Approach)



n_1

$x = \&y$

$y = \&z$

$z = \&u$

n_2

$*z = y$

n_3

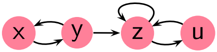
$z = y$

n_4

$y = \&x$

use u

use x

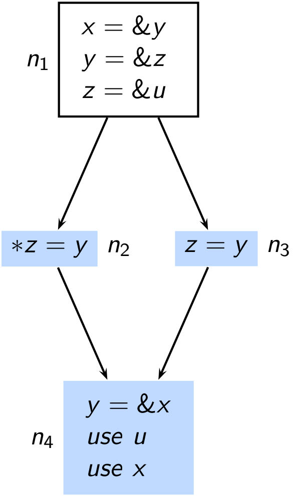


Points-to Graph

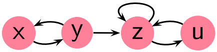
Constraints on Points-to Sets

$$\begin{aligned} P_x &\supseteq \{y\} \\ P_y &\supseteq \{z\} \\ P_z &\supseteq \{u\} \\ \forall w \in P_z,\ \ P_w &\supseteq P_y \\ P_z &\supseteq P_y \\ P_y &\supseteq \{x\} \end{aligned}$$

An Example of Flow Insensitive Points-to Analysis (Andersen's Approach aka Inclusion Based Approach)



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

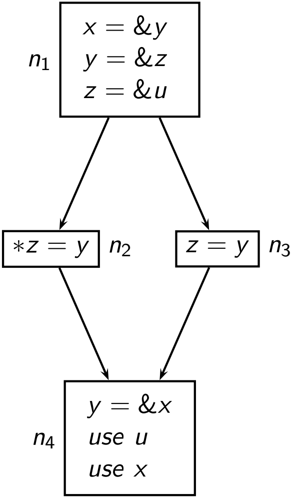


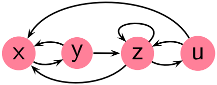
Points-to Graph

Constraints on Points-to Sets

$$
\begin{aligned}
P_x &\supseteq \{y\}\\
P_y &\supseteq \{z\}\\
P_z &\supseteq \{u\}\\
\forall w \in P_z,\ P_w &\supseteq P_y\\
P_z &\supseteq P_y\\
P_y &\supseteq \{x\}
\end{aligned}
$$

An Example of Flow Insensitive Points-to Analysis (Andersen's Approach aka Inclusion Based Approach)



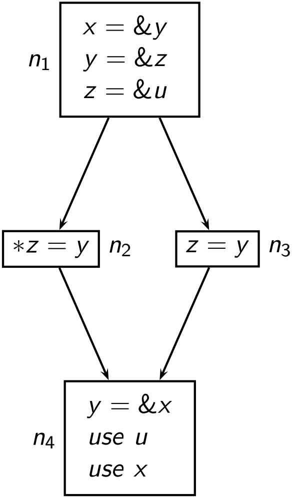


Points-to Graph

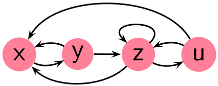
Constraints on Points-to Sets

$$\begin{aligned} P_x &\supseteq \{y\} \\ P_y &\supseteq \{z\} \\ P_z &\supseteq \{u\} \\ \forall w \in P_z,\ P_w &\supseteq P_y \\ P_z &\supseteq P_y \\ P_y &\supseteq \{x\} \end{aligned}$$

An Example of Flow Insensitive Points-to Analysis (Andersen's Approach aka Inclusion Based Approach)



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

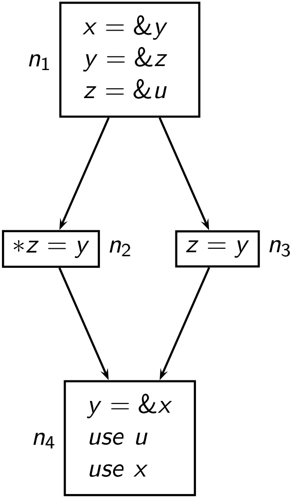


Points-to Graph

Constraints on Points-to Sets

$$
\begin{aligned}
P_x &\supseteq \{y\}\\
P_y &\supseteq \{z\}\\
P_z &\supseteq \{u\}\\
\forall w \in P_z,\ P_w &\supseteq P_y\\
P_z &\supseteq P_y\\
P_y &\supseteq \{x\}
\end{aligned}
$$

An Example of Flow Insensitive Points-to Analysis (Andersen's Approach aka Inclusion Based Approach)


n_1

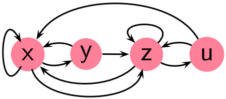
$x = \&y$
$y = \&z$
$z = \&u$

$*z = y$ n_2

$z = y$ n_3

n_4

$y = \&x$
use u
use x

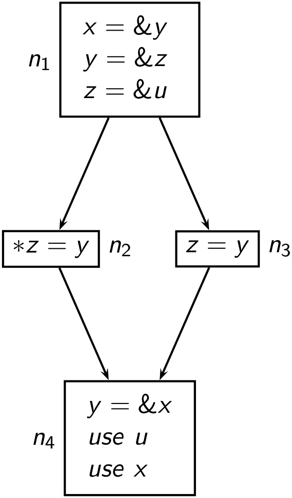


Points-to Graph

Constraints on Points-to Sets

$$\begin{aligned} P_x &\supseteq \{y\} \\ P_y &\supseteq \{z\} \\ P_z &\supseteq \{u\} \\ \forall w \in P_z,\ P_w &\supseteq P_y \\ P_z &\supseteq P_y \\ P_y &\supseteq \{x\} \end{aligned}$$

An Example of Flow Insensitive Points-to Analysis (Steensgaard's Approach aka Equality Based Approach)

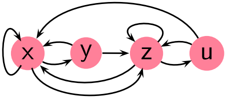


n_1: $x = \&y$, $y = \&z$, $z = \&u$

n_2: $*z = y$

n_3: $z = y$

n_4: $y = \&x$, use u, use x

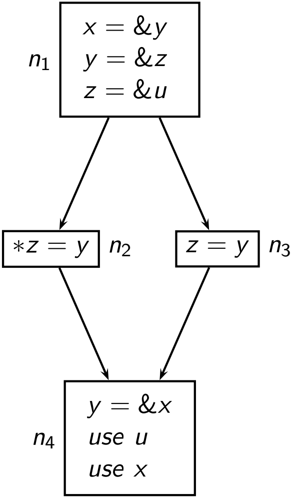


Andersen's Points-to Graph

Constraints on Points-to Sets

$$
\begin{aligned}
P_x &\supseteq \{y\}\\
P_y &\supseteq \{z\}\\
P_z &\supseteq \{u\}\\
\forall w \in P_z,\; P_w &\supseteq P_y\\
P_z &\supseteq P_y\\
P_y &\supseteq \{x\}
\end{aligned}
$$

An Example of Flow Insensitive Points-to Analysis (Steensgaard's Approach aka Equality Based Approach)



n_1: $x = \&y$, $y = \&z$, $z = \&u$

n_2: $*z = y$

n_3: $z = y$

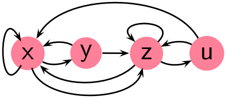
n_4: $y = \&x$, use u, use x

For each "lhs = rhs"

- Equate the points-to sets of lhs and rhs

  Effectively, create new assignment "rhs = lhs"

- In practice, an efficient algorithm computes the combined effect



Andersen's Points-to Graph

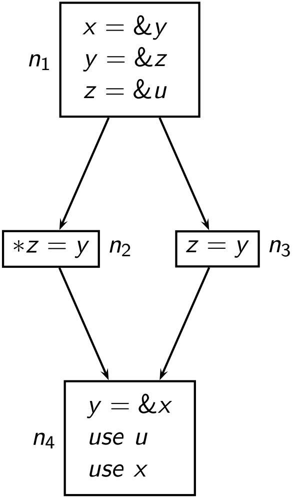
Constraints on Points-to Sets

$$\begin{aligned} P_x &\supseteq \{y\} \\ P_y &\supseteq \{z\} \\ P_z &\supseteq \{u\} \\ \forall w \in P_z,\ \ P_w &\supseteq P_y \\ P_z &\supseteq P_y \\ P_y &\supseteq \{x\} \end{aligned}$$

New Constraints

$$\begin{aligned} \forall w \in P_z,\ \ P_w &\subseteq P_y \\ P_z &\subseteq P_y \end{aligned}$$

An Example of Flow Insensitive Points-to Analysis (Steensgaard's Approach aka Equality Based Approach)



n_1

$x = \&y$
$y = \&z$
$z = \&u$

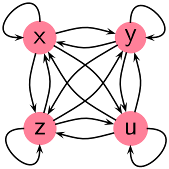
$*z = y$ n_2

$z = y$ n_3

n_4

$y = \&x$
use u
use x

- For this example, we get a complete graph



Steensgaard's Points-to Graph

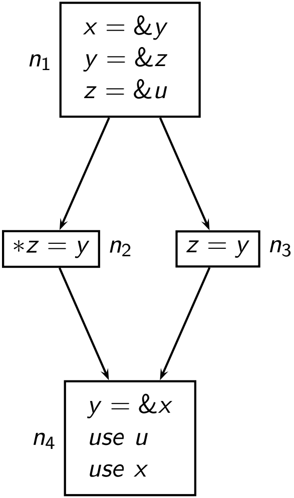
Constraints on Points-to Sets

$$\begin{aligned} P_x &\supseteq \{y\} \\ P_y &\supseteq \{z\} \\ P_z &\supseteq \{u\} \\ \forall w \in P_z,\ \ P_w &\supseteq P_y \\ P_z &\supseteq P_y \\ P_y &\supseteq \{x\} \end{aligned}$$

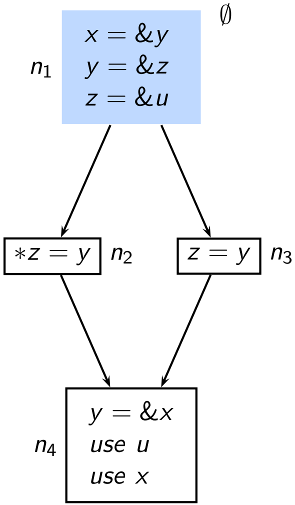
New Constraints

$$\begin{aligned} \forall w \in P_z,\ \ P_w &\subseteq P_y \\ P_z &\subseteq P_y \end{aligned}$$

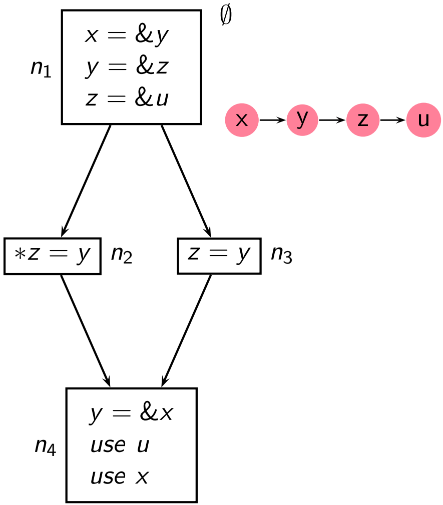
An Example of Flow Sensitive Points-to Analysis



An Example of Flow Sensitive Points-to Analysis



An Example of Flow Sensitive Points-to Analysis



$\emptyset$

n_1

$x = \&y$

$y = \&z$

$z = \&u$

n_2 $*z = y$

n_3 $z = y$

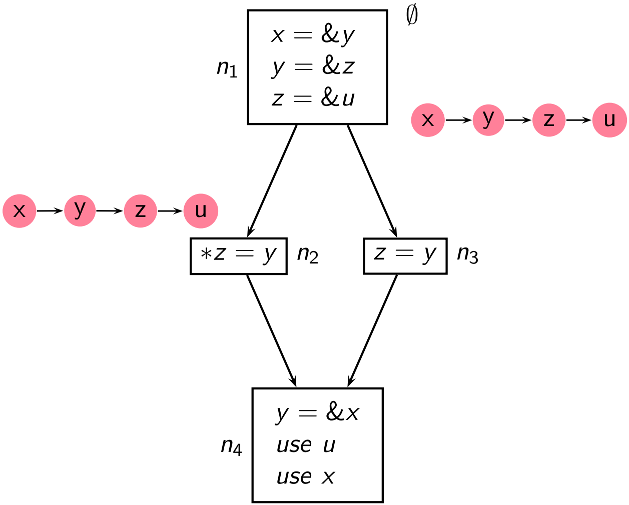
n_4

$y = \&x$

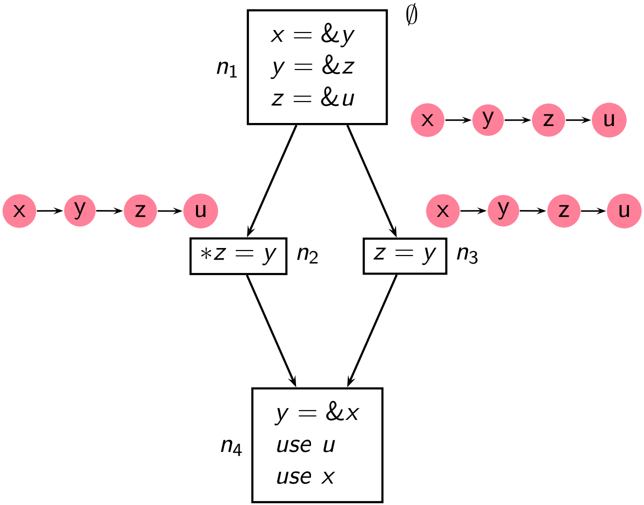
$use\ u$

$use\ x$

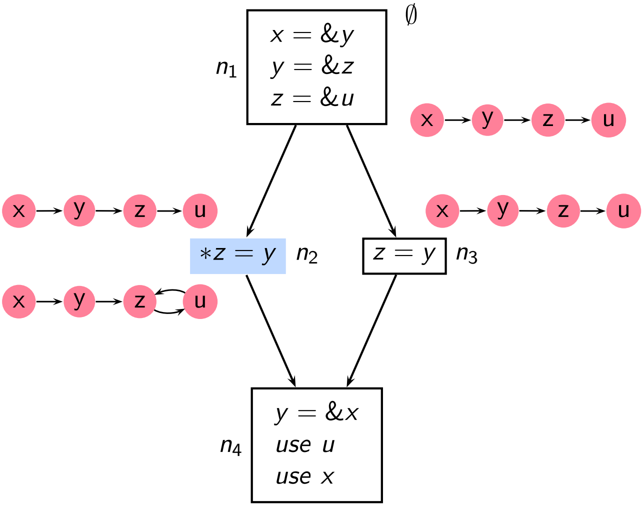
An Example of Flow Sensitive Points-to Analysis



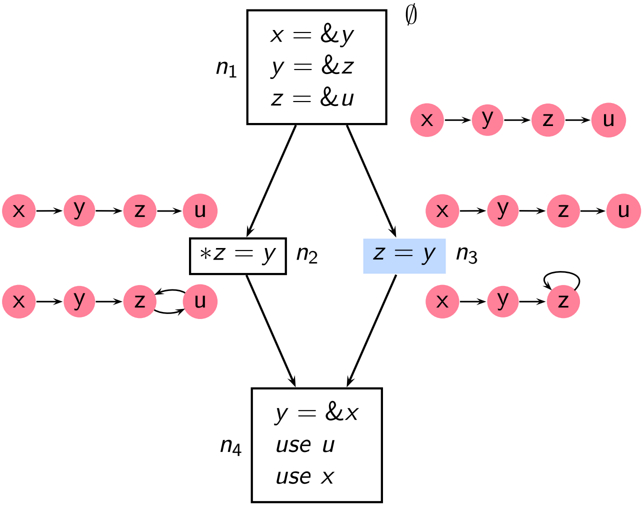
An Example of Flow Sensitive Points-to Analysis



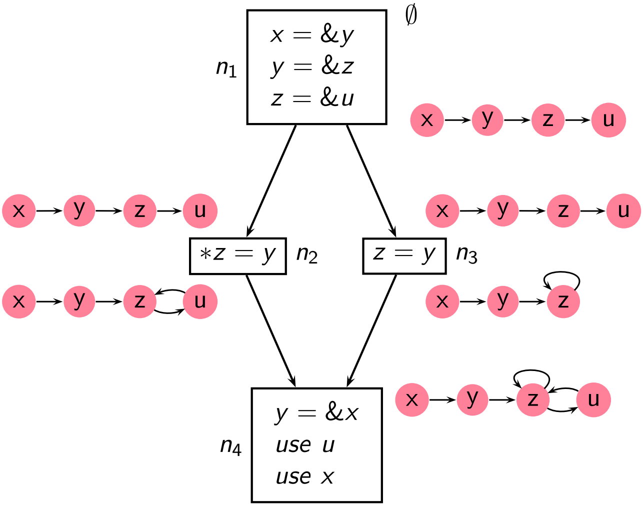
An Example of Flow Sensitive Points-to Analysis



An Example of Flow Sensitive Points-to Analysis



An Example of Flow Sensitive Points-to Analysis



n_1

$x = \&y$

$y = \&z$

$z = \&u$

$\emptyset$

n_2

$*z = y$

n_3

$z = y$

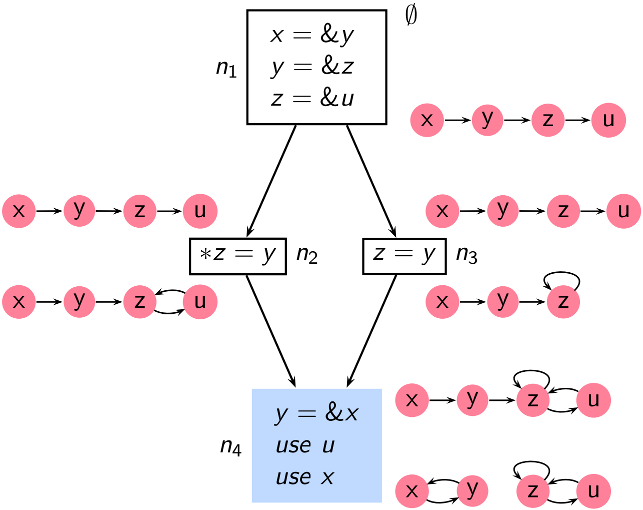
n_4

$y = \&x$

use u

use x

An Example of Flow Sensitive Points-to Analysis



$\emptyset$

n_1

$x = \&y$

$y = \&z$

$z = \&u$

$*z = y$ n_2

$z = y$ n_3

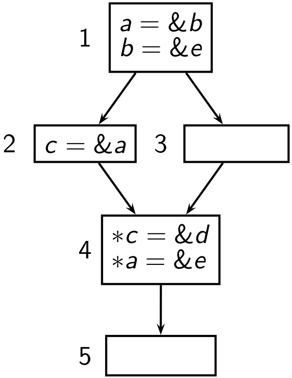
n_4

$y = \&x$

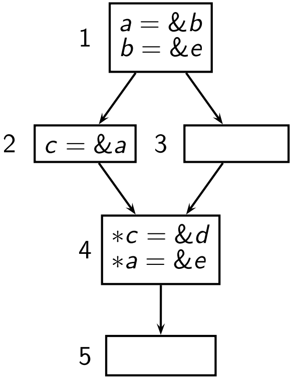
use u

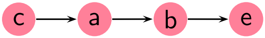
use x

Flow Sensitive Points-to Analysis: May and Must Variants

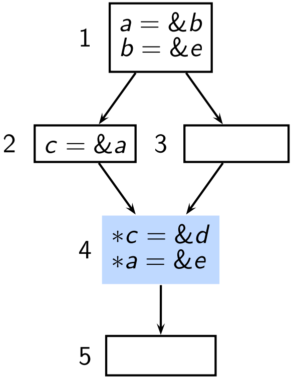


Flow Sensitive Points-to Analysis: May and Must Variants



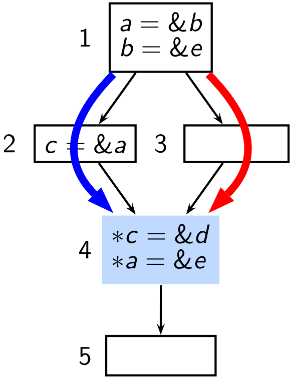
-  at the entry of 4

Flow Sensitive Points-to Analysis: May and Must Variants



- c → a → b → e at the entry of 4
- Should a → b be killed by assignment $*c = \&d$?

Flow Sensitive Points-to Analysis: May and Must Variants

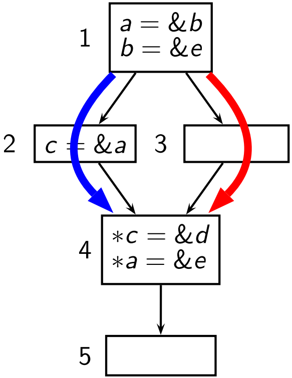


- c → a → b → e at the entry of 4
- Should a → b be killed by assignment

  $*c = \&d$?

  No because c points to a along path $1, 2, 4$ but not along path $1, 3, 4$

Flow Sensitive Points-to Analysis: May and Must Variants

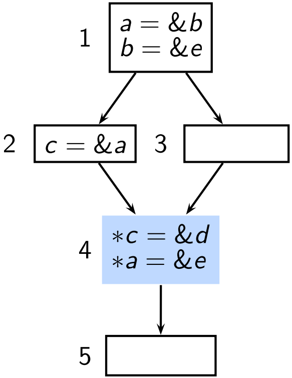


- $c \xrightarrow{\text{MAY}} a \longrightarrow b \longrightarrow e$ at the entry of 4
- Should $a \longrightarrow b$ be killed by assignment

  $*c = \&d$?

  No because c points to a along path $1, 2, 4$ but not along path $1, 3, 4$

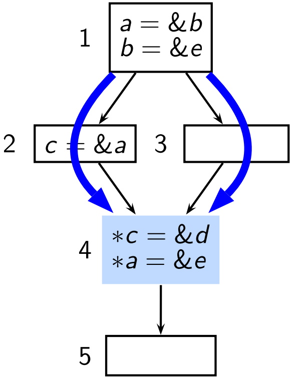
Flow Sensitive Points-to Analysis: May and Must Variants



- $c \xrightarrow{\text{MAY}} a \rightarrow b \rightarrow e$ at the entry of 4
- Should $a \rightarrow b$ be killed by assignment $*c = \&d$?

  No because c points to a along path $1, 2, 4$ but not along path $1, 3, 4$
- Should $b \rightarrow e$ be killed by assignment $*a = \&e$?

Flow Sensitive Points-to Analysis: May and Must Variants

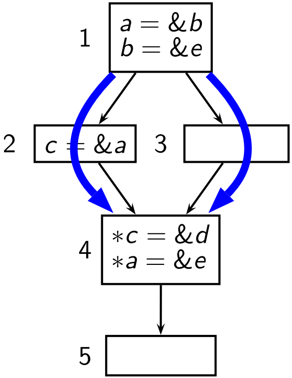


- $c \xrightarrow{\text{MAY}} a \xrightarrow{\text{MUST}} b \longrightarrow e$ at the entry of 4
- Should $a \longrightarrow b$ be killed by assignment $*c = \&d$?

  No because c points to a along path $1, 2, 4$ but not along path $1, 3, 4$
- Should $b \longrightarrow e$ be killed by assignment $*a = \&e$?

  Yes because a points to b along both the paths

Flow Sensitive Points-to Analysis: May and Must Variants



- $c \xrightarrow{\text{MAY}} a \xrightarrow{\text{MUST}} b \rightarrow e$ at the entry of 4
- Should $a \rightarrow b$ be killed by assignment

  $*c = \&d$?

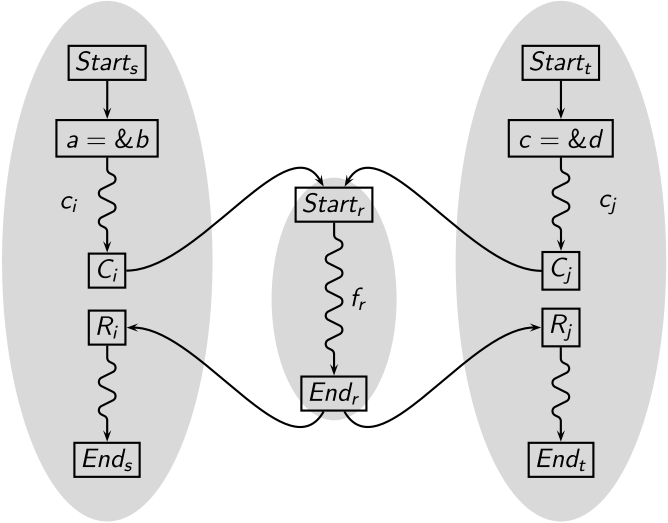
  No because c points to a along path 1, 2, 4 but not along path 1, 3, 4
- Should $b \rightarrow e$ be killed by assignment

  $*a = \&e$?

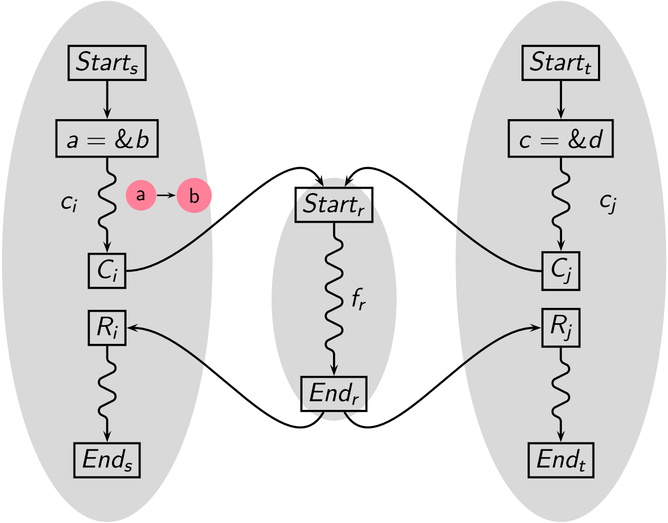
  Yes because a points to b along both the paths
- Must points-to information is required for killing May points-to information

  (and vice-versa)

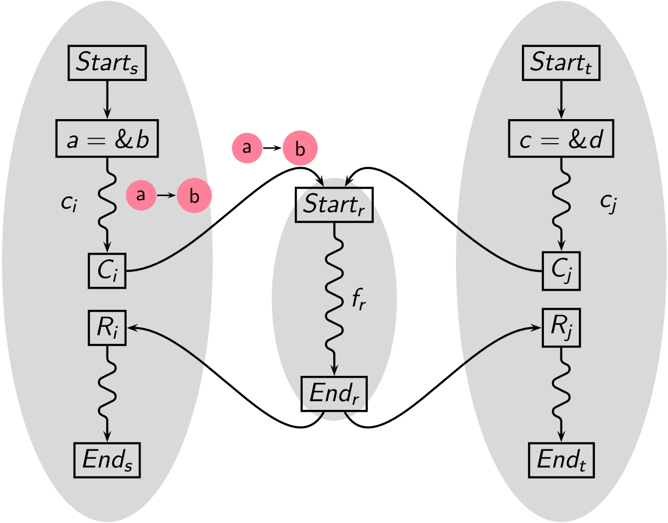
Context Sensitivity in Interprocedural Analysis



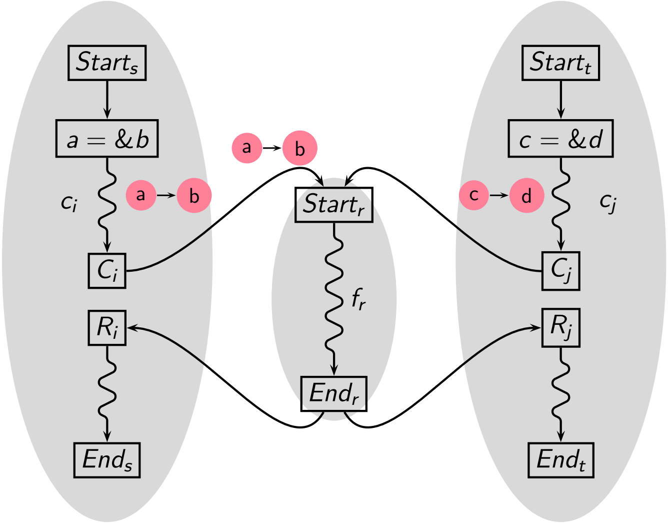
Context Sensitivity in Interprocedural Analysis



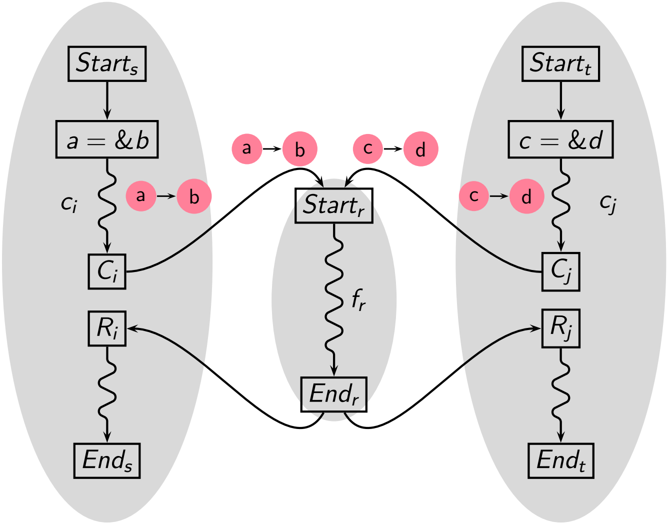
Context Sensitivity in Interprocedural Analysis



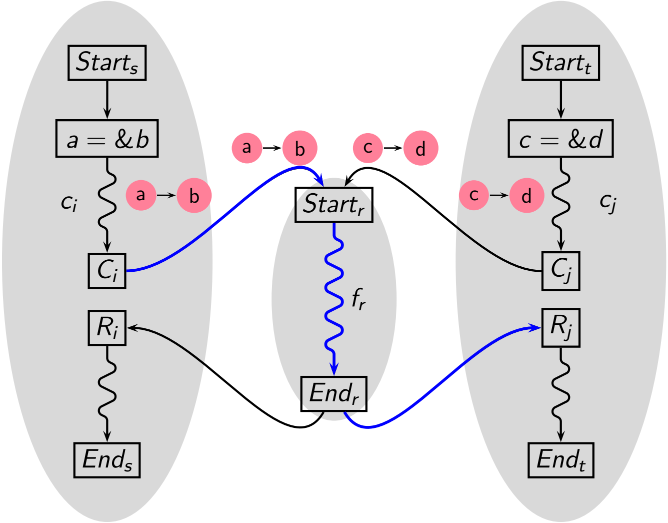
Context Sensitivity in Interprocedural Analysis



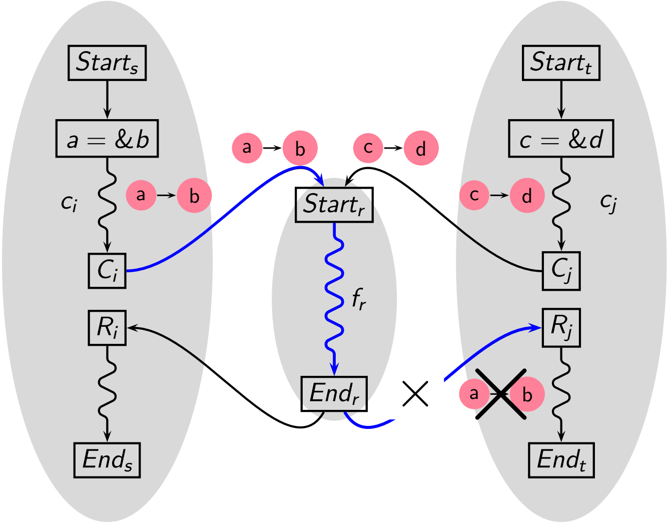
Context Sensitivity in Interprocedural Analysis



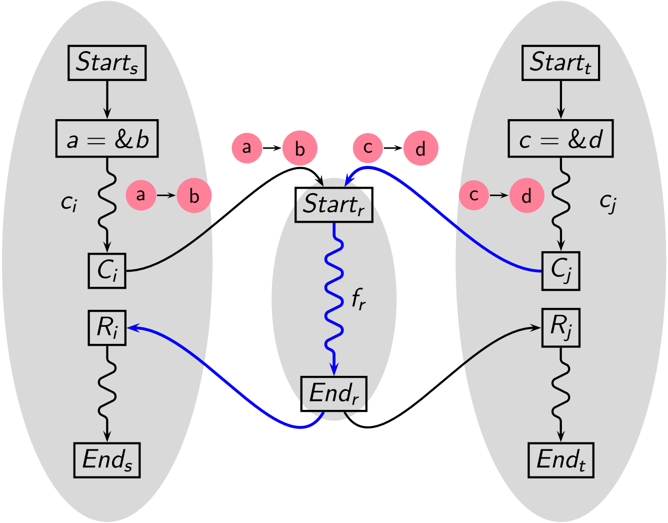
Context Sensitivity in Interprocedural Analysis



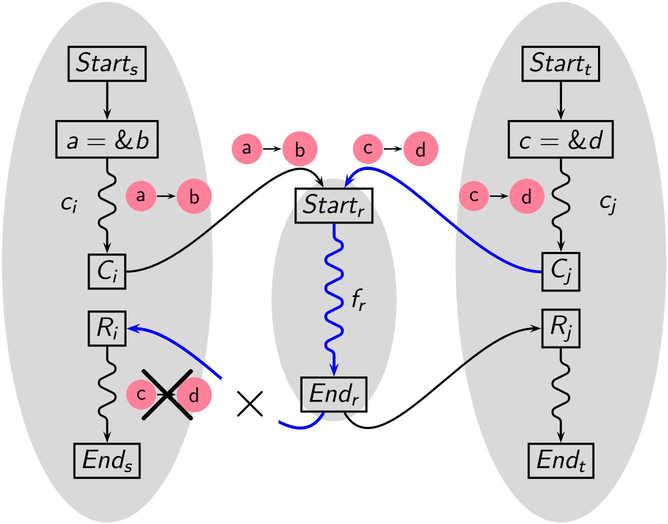
Context Sensitivity in Interprocedural Analysis



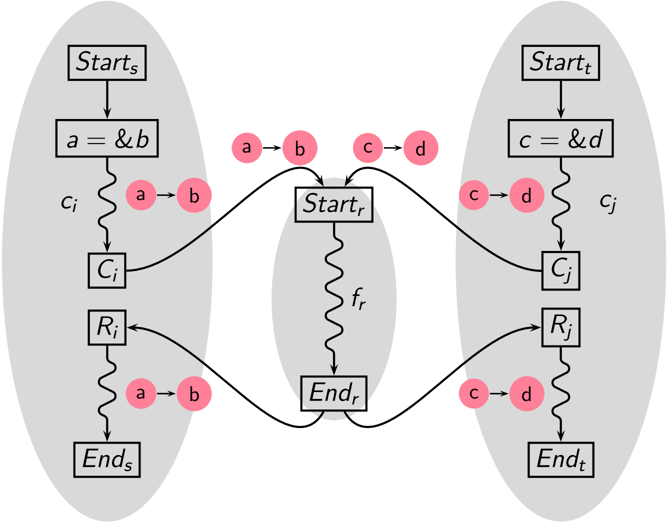
Context Sensitivity in Interprocedural Analysis



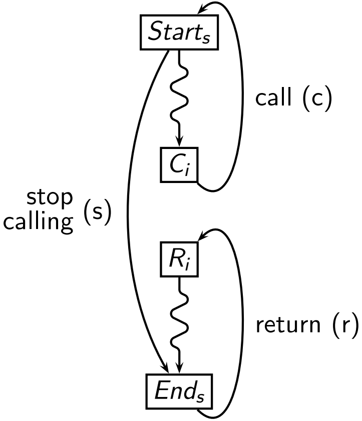
Context Sensitivity in Interprocedural Analysis



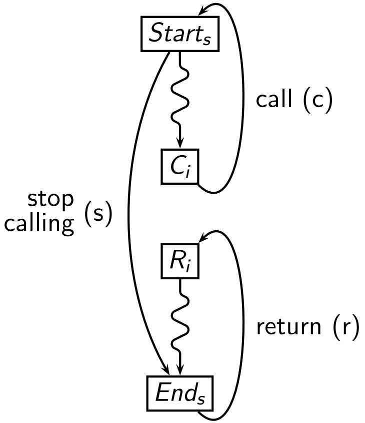
Context Sensitivity in Interprocedural Analysis



Context Sensitivity in the Presence of Recursion

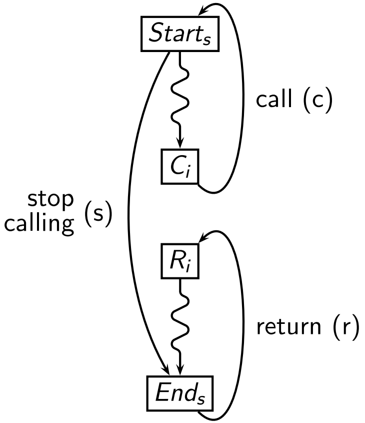


Context Sensitivity in the Presence of Recursion



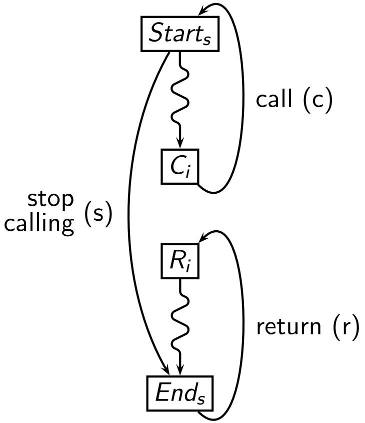
- Paths from $Start_s$ to End_s should constitute a context free language $c^n s r^n$

Context Sensitivity in the Presence of Recursion



- Paths from $Start_s$ to End_s should constitute a context free language $c^n s r^n$
- Many interprocedural analyses treat cycle of recursion as an SCC and approximate paths by a regular language $c^* s r^*$

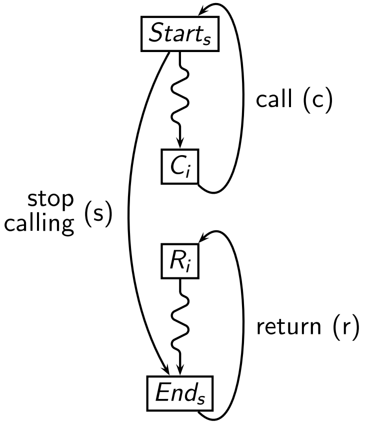
Context Sensitivity in the Presence of Recursion



- Paths from $Start_s$ to End_s should constitute a context free language $c^n s r^n$
- Many interprocedural analyses treat cycle of recursion as an SCC and approximate paths by a regular language $c^* s r^*$
- We do not know any practical points-to analysis that is fully context sensitive

  Most context sensitive approaches

Context Sensitivity in the Presence of Recursion

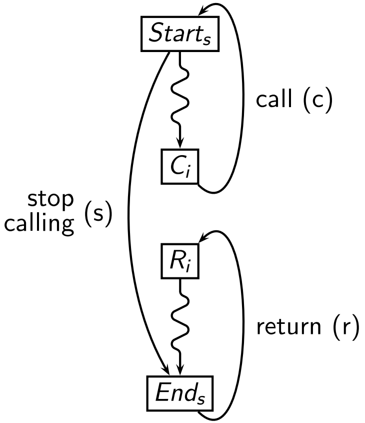


- Paths from $Start_s$ to End_s should constitute a context free language $c^n s r^n$
- Many interprocedural analyses treat cycle of recursion as an SCC and approximate paths by a regular language $c^* s r^*$
- We do not know any practical points-to analysis that is fully context sensitive

  Most context sensitive approaches

  - either do not consider recursion, or

Context Sensitivity in the Presence of Recursion

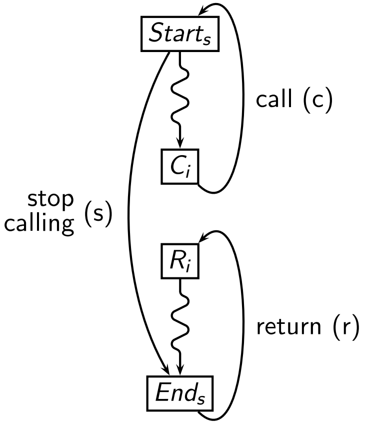


- Paths from $Start_s$ to End_s should constitute a context free language $c^n s r^n$
- Many interprocedural analyses treat cycle of recursion as an SCC and approximate paths by a regular language $c^* s r^*$
- We do not know any practical points-to analysis that is fully context sensitive

  Most context sensitive approaches

  - either do not consider recursion, or
  - do not consider recursive pointer manipulation (e.g. "$p = *p$"), or

Context Sensitivity in the Presence of Recursion

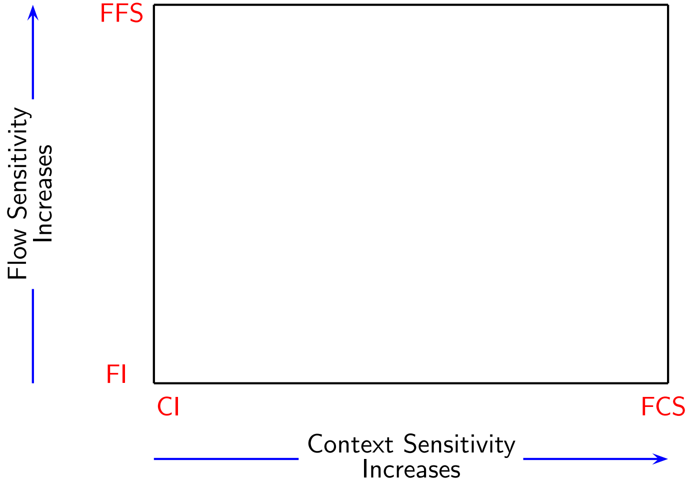


- Paths from $Start_s$ to End_s should constitute a context free language $c^n s r^n$
- Many interprocedural analyses treat cycle of recursion as an SCC and approximate paths by a regular language $c^* s r^*$
- We do not know any practical points-to analysis that is fully context sensitive

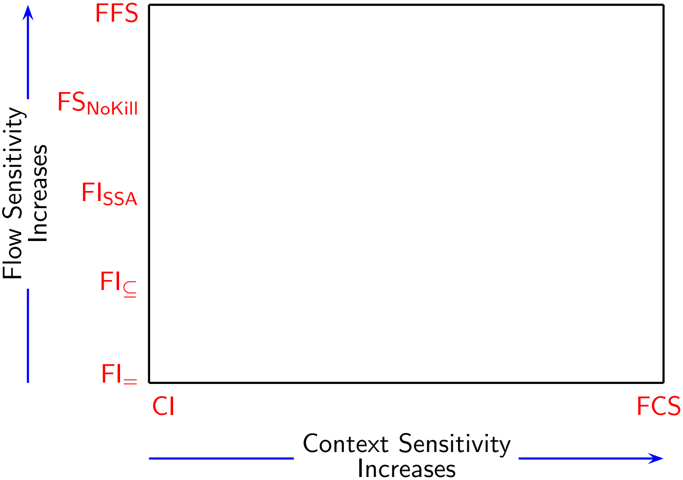
  Most context sensitive approaches

  - either do not consider recursion, or
  - do not consider recursive pointer manipulation (e.g. "$p = *p$"), or
  - are context insensitive in recursion

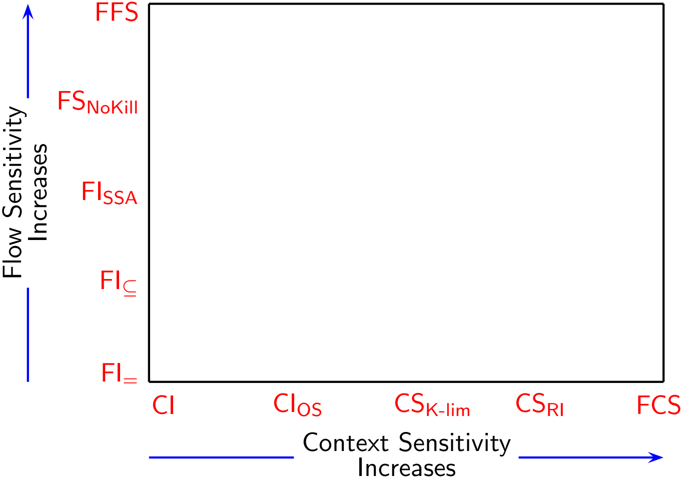
Pointer Analysis: An Engineer's Landscape



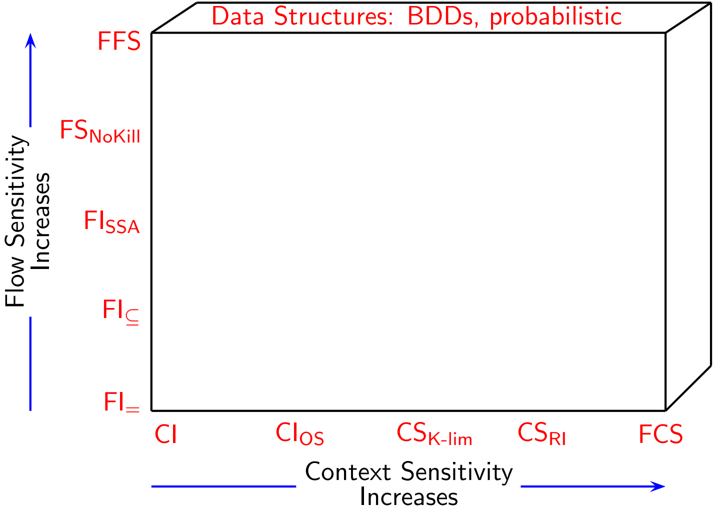
Pointer Analysis: An Engineer's Landscape



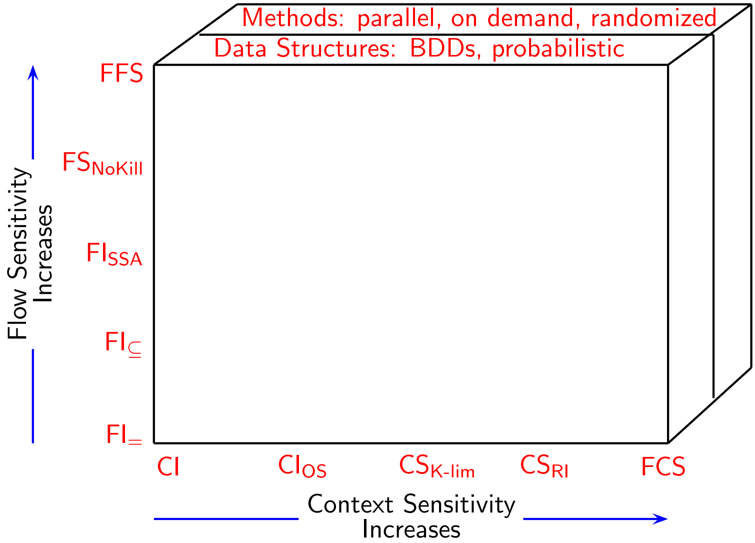
Pointer Analysis: An Engineer's Landscape



Pointer Analysis: An Engineer's Landscape



Pointer Analysis: An Engineer's Landscape



Methods: parallel, on demand, randomized

Data Structures: BDDs, probabilistic

Flow Sensitivity Increases

FFS

FS_{NoKill}

FI_{SSA}

$FI_{\subseteq}$

$FI_{=}$

CI

CI_{OS}

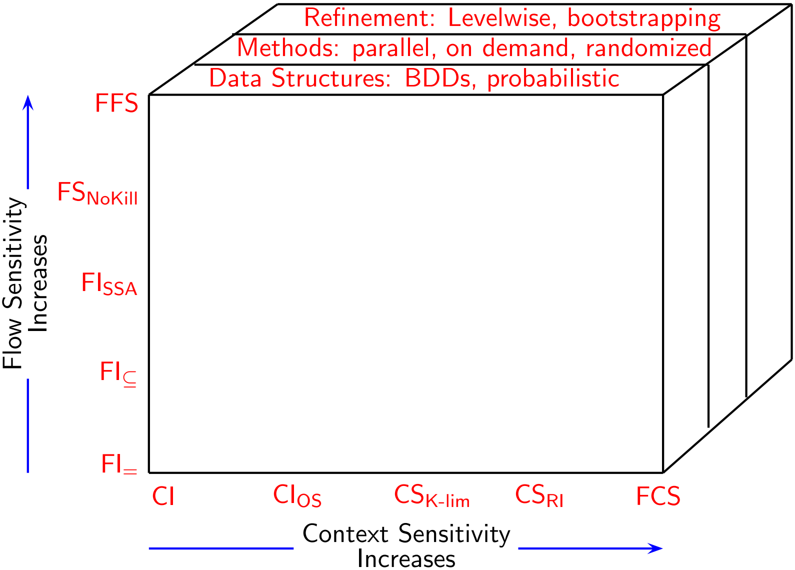
$CS_{K\text{-}lim}$

CS_{RI}

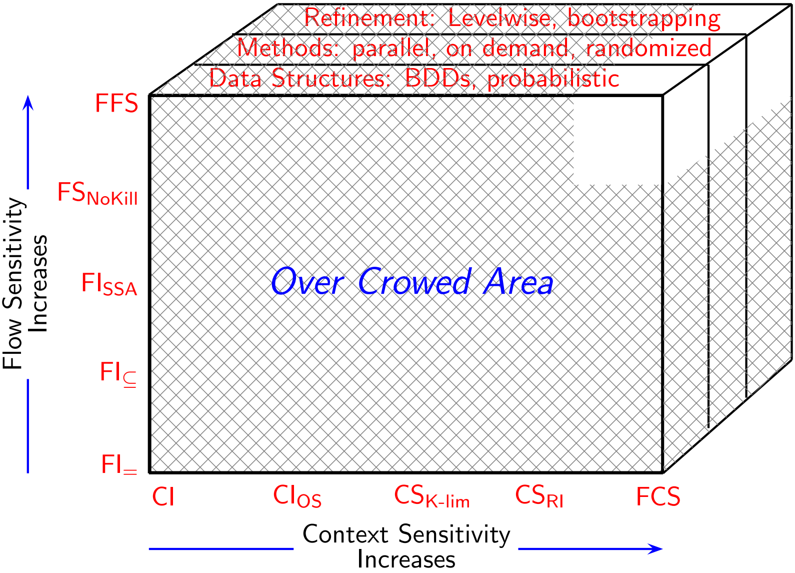
FCS

Context Sensitivity Increases

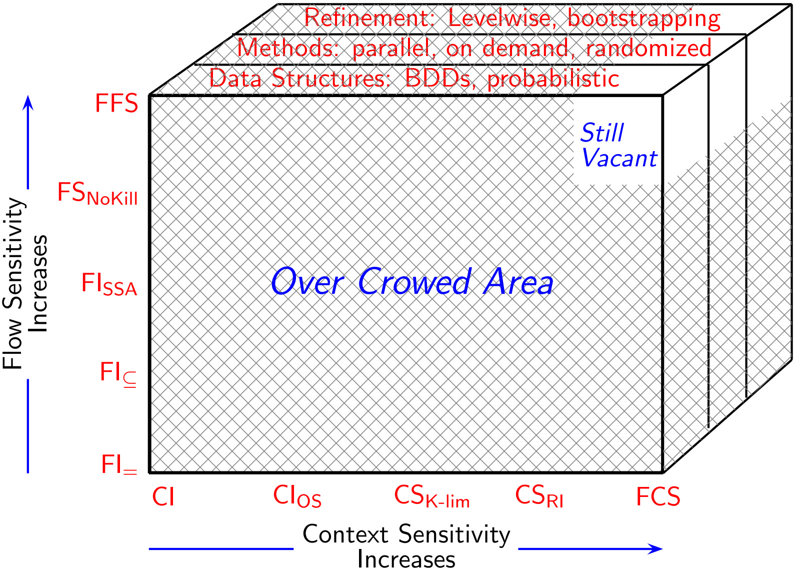
Pointer Analysis: An Engineer's Landscape


Refinement: Levelwise, bootstrapping
Methods: parallel, on demand, randomized
Data Structures: BDDs, probabilistic
FFS
$\text{FS}_{\text{NoKill}}$
FI_{SSA}
$\text{FI}_{\subseteq}$
$\text{FI}_{=}$
Flow Sensitivity Increases
CI
CI_{OS}
$\text{CS}_{\text{K-lim}}$
CS_{RI}
FCS
Context Sensitivity Increases

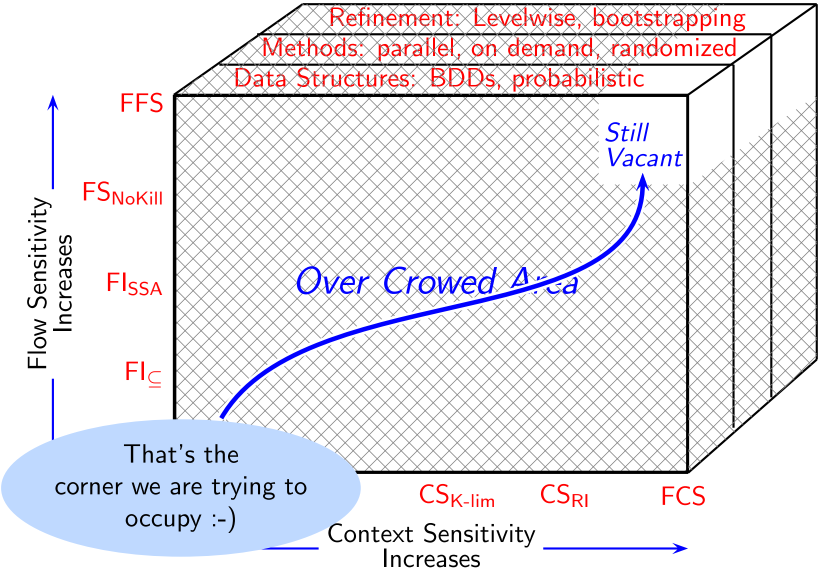
Pointer Analysis: An Engineer's Landscape


Refinement: Levelwise, bootstrapping
Methods: parallel, on demand, randomized
Data Structures: BDDs, probabilistic
FFS
FS_{NoKill}
FI_{SSA}
$FI_{\subseteq}$
$FI_{=}$
Flow Sensitivity Increases
Over Crowed Area
CI
CI_{OS}
$CS_{K\text{-}lim}$
CS_{RI}
FCS
Context Sensitivity Increases

Pointer Analysis: An Engineer's Landscape


Refinement: Levelwise, bootstrapping
Methods: parallel, on demand, randomized
Data Structures: BDDs, probabilistic
Still Vacant
Over Crowed Area
Flow Sensitivity Increases
FFS
$\text{FS}_{\text{NoKill}}$
FI_{SSA}
$\text{FI}_{\subseteq}$
$\text{FI}_{=}$
CI
CI_{OS}
$\text{CS}_{\text{K-lim}}$
CS_{RI}
FCS
Context Sensitivity Increases

Pointer Analysis: An Engineer's Landscape

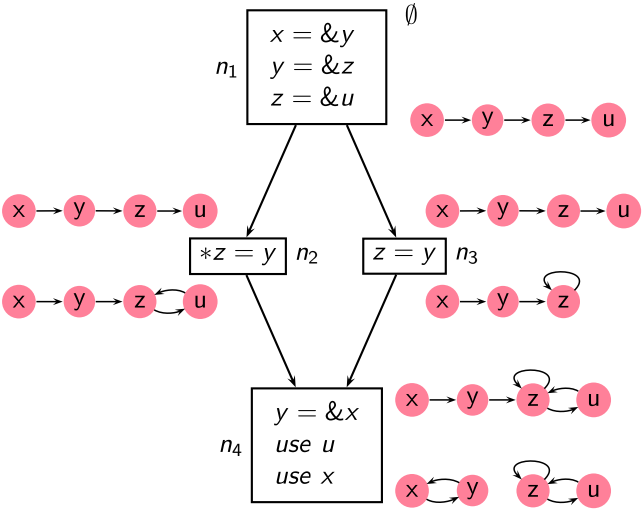

Refinement: Levelwise, bootstrapping
Methods: parallel, on demand, randomized
Data Structures: BDDs, probabilistic
FFS
FS_{NoKill}
FI_{SSA}
$FI_{\subseteq}$
Flow Sensitivity Increases
Still Vacant
Over Crowed Area
$CS_{K\text{-lim}}$
CS_{RI}
FCS
Context Sensitivity Increases

That's the corner we are trying to occupy :-)

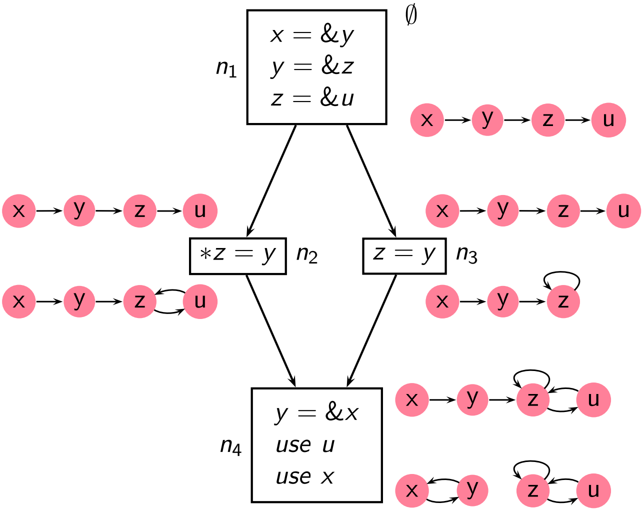
Part 3

Formulating LFCPA

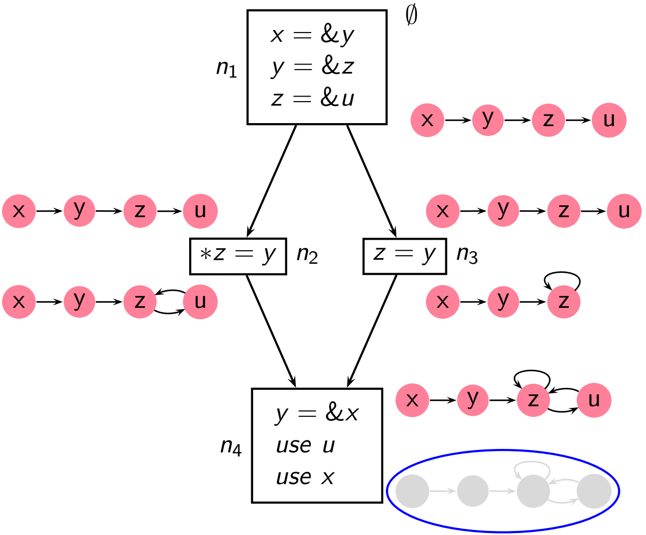
Our Motivating Example for Intraprocedural Formulation



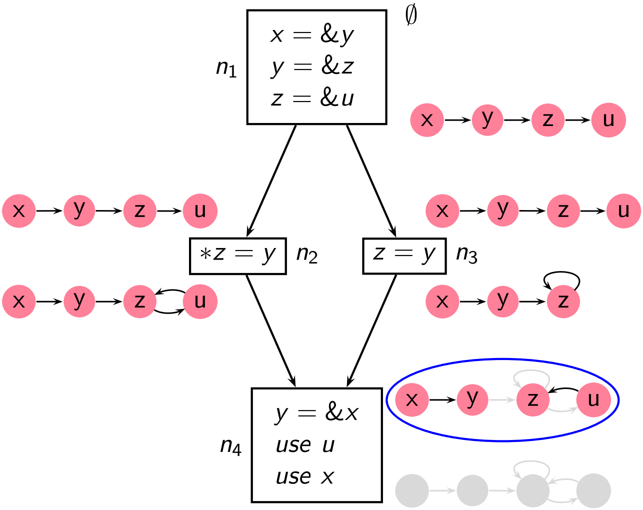
Is All This Information Useful?



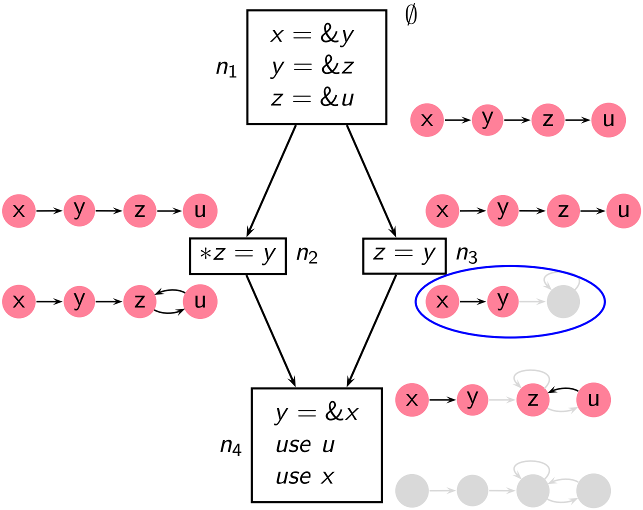
Is All This Information Useful?



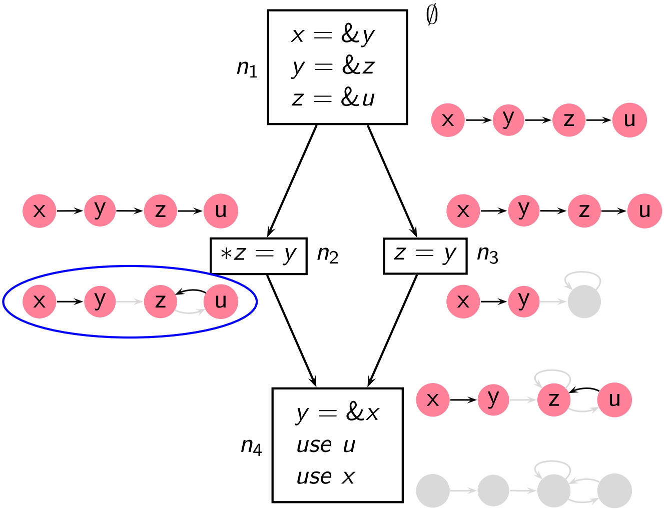
Is All This Information Useful?



Is All This Information Useful?



Is All This Information Useful?



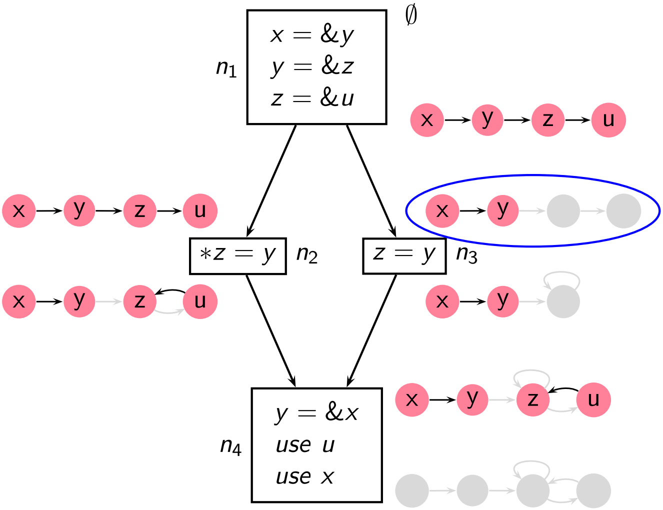
n_1: $x = \&y$, $y = \&z$, $z = \&u$

n_2: $*z = y$

n_3: $z = y$

n_4: $y = \&x$, use u, use x

Is All This Information Useful?



The L and P of LFCPA

Mutual dependence of liveness and points-to information

- Define points-to information only for live pointers
- For pointer indirections, define liveness information using points-to information

The F and C of LFCPA

- Use call strings method for full flow and context sensitivity
- Use value based termination of call strings construction for efficiency [Khedker, Karkare. CC 2008]

Use of Strong Liveness

- Simple liveness considers every use of a variable as useful
- Strong liveness checks the liveness of the result before declaring the operands to be live

Use of Strong Liveness

- Simple liveness considers every use of a variable as useful
- Strong liveness checks the liveness of the result before declaring the operands to be live
- Strong liveness is more precise than simple liveness

Data Flow Equations

$$Lout_n = \begin{cases} \emptyset & n \text{ is } End_p \\ \displaystyle\bigcup_{s \in succ(n)} Lin_s & \text{otherwise} \end{cases}$$

$$Lin_n = \left(Lout_n - Kill_n \right) \cup Ref_n$$

$$Ain_n = \begin{cases} Lin_n \times \{?\} & n \text{ is } Start_p \\ \left(\displaystyle\bigcup_{p \in pred(n)} Aout_p \right) \Bigg|_{Lin_n} & \text{otherwise} \end{cases}$$

$$Aout_n = \left(\left(Ain_n - \left(Kill_n \times \mathbf{V} \right) \right) \cup \left(Def_n \times Pointee_n \right) \right) \Bigg|_{Lout_n}$$

- $Lin/Lout$: set of Live pointers, $Ain/Aout$: sets of mAy points-to pairs

Data Flow Equations

$$Lout_n = \begin{cases} \emptyset & n \text{ is } End_p \\ \bigcup\limits_{s \in succ(n)} Lin_s & \text{otherwise} \end{cases}$$

$$Lin_n = \left(Lout_n - \boxed{Kill_n}\right) \cup \boxed{Ref_n}$$

$$Ain_n = \begin{cases} Lin_n \times \{?\} & n \text{ is } Start_p \\ \left(\bigcup\limits_{p \in pred(n)} Aout_p\right)\Bigg|_{Lin_n} & \text{otherwise} \end{cases}$$

$$Aout_n = \left(\left(Ain_n - \left(\boxed{Kill_n} \times \mathbf{V}\right)\right) \cup \left(\boxed{Def_n} \times \boxed{Pointee_n}\right)\right)\Bigg|_{Lout_n}$$

- $Lin/Lout$: set of Live pointers, $Ain/Aout$: sets of mAy points-to pairs
- Ref_n, $Kill_n$, Def_n, and $Pointee_n$ are defined in terms of Ain_n

Data Flow Equations

$$Lout_n = \begin{cases} \emptyset & n \text{ is } End_p \\ \bigcup\limits_{s \in succ(n)} Lin_s & \text{otherwise} \end{cases}$$

$$Lin_n = \left(Lout_n - Kill_n\right) \cup \boxed{Ref_n}$$

Ref_n is defined in terms of $Lout_n$

$$Ain_n = \begin{cases} Lin_n \times \{?\} & n \text{ is } Start_p \\ \left.\left(\bigcup\limits_{p \in pred(n)} Aout_p\right)\right|_{Lin_n} & \text{otherwise} \end{cases}$$

$$Aout_n = \left.\left(\left(Ain_n - \left(Kill_n \times \mathbf{V}\right)\right) \cup \left(Def_n \times Pointee_n\right)\right)\right|_{Lout_n}$$

- $Lin/Lout$: set of Live pointers, $Ain/Aout$: sets of mAy points-to pairs
- Ref_n, $Kill_n$, Def_n, and $Pointee_n$ are defined in terms of Ain_n

Data Flow Equations

$$Lout_n = \begin{cases} \emptyset & n \text{ is } End_p \\ \displaystyle\bigcup_{s\in succ(n)} Lin_s & \text{otherwise} \end{cases}$$

$$Lin_n = \left(Lout_n - Kill_n\right) \cup Ref_n$$

$$Ain_n = \begin{cases} Lin_n \times \{?\} & n \text{ is } Start_p \\ \left(\displaystyle\bigcup_{p\in pred(n)} Aout_p\right)\Bigg|_{\boxed{Lin_n}} & \text{otherwise} \end{cases}$$

$$Aout_n = \left(\left(Ain_n - \left(Kill_n \times \mathbf{V}\right)\right) \cup \left(Def_n \times Pointee_n\right)\right)\Bigg|_{\boxed{Lout_n}}$$

Ain_n and $Aout_n$ are restricted to Lin_n and $Lout_n$

- $Lin/Lout$: set of Live pointers, $Ain/Aout$: sets of mAy points-to pairs
- Ref_n, $Kill_n$, Def_n, and $Pointee_n$ are defined in terms of Ain_n

Data Flow Equations

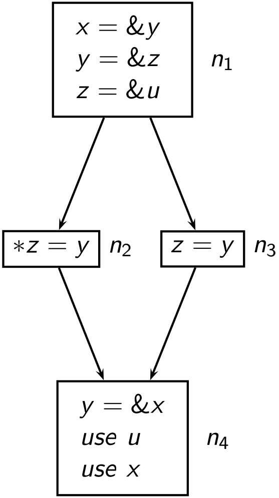
$$
\begin{aligned}
Lout_n &= \begin{cases} \emptyset & n \text{ is } End_p \\ \displaystyle\bigcup_{s \in succ(n)} Lin_s & \text{otherwise} \end{cases} \\
Lin_n &= \left(Lout_n - \; Kill_n \right) \cup \; Ref_n \\
Ain_n &= \begin{cases} Lin_n \times \{?\} & n \text{ is } Start_p \\ \left(\displaystyle\bigcup_{p \in pred(n)} Aout_p \right) \Bigg|_{Lin_n} & \text{otherwise} \end{cases} \\
Aout_n &= \left(\left(Ain_n - \left(\; Kill_n \; \times \mathbf{V} \right) \right) \cup \left(\; Def_n \; \times \; Pointee_n \; \right) \right) \Bigg|_{Lout_n}
\end{aligned}
$$

- $Lin/Lout$: set of Live pointers, $Ain/Aout$: sets of mAy points-to pairs
- Ref_n, $Kill_n$, Def_n, and $Pointee_n$ are defined in terms of Ain_n

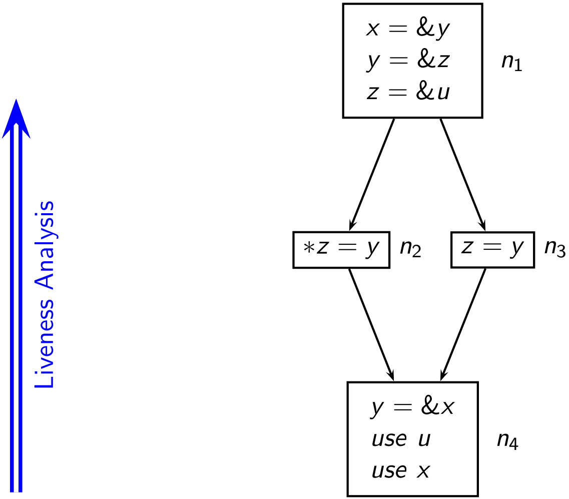
Motivating Example Revisited

- For convenience, we show complete sweeps of liveness and points-to analysis repeatedly
- This is not required by the computation
- The data flow equations define a single fixed point computation

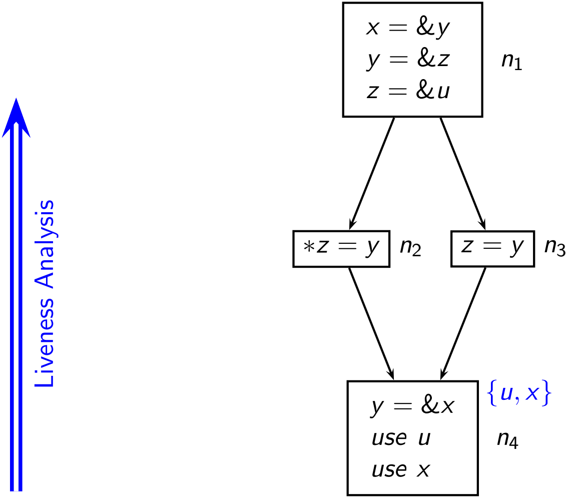
First Round of Liveness Analysis and Points-to Analysis



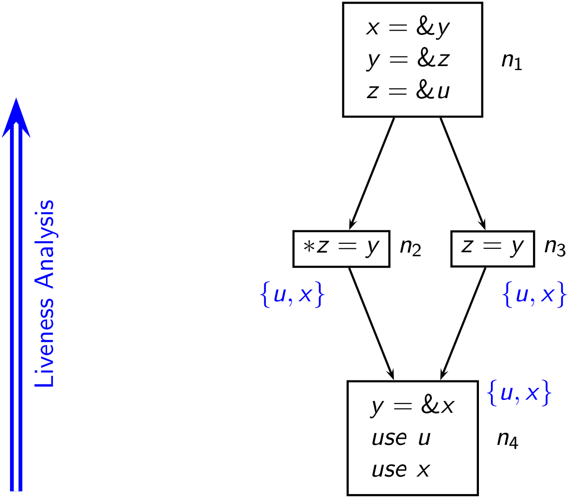
First Round of Liveness Analysis and Points-to Analysis



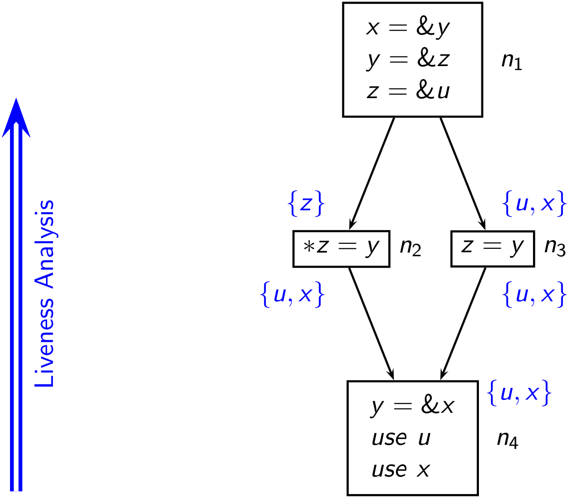
First Round of Liveness Analysis and Points-to Analysis



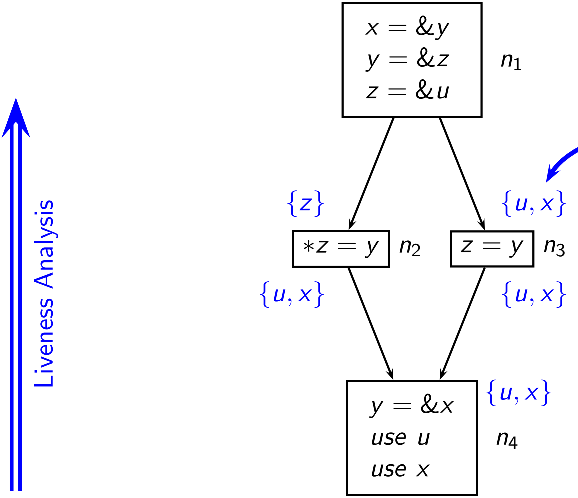
First Round of Liveness Analysis and Points-to Analysis



First Round of Liveness Analysis and Points-to Analysis

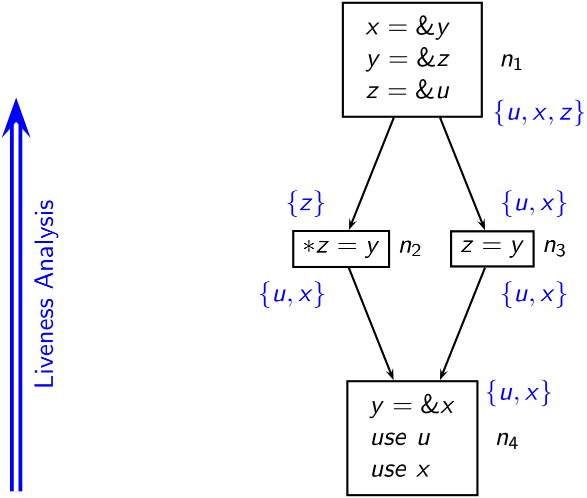


First Round of Liveness Analysis and Points-to Analysis

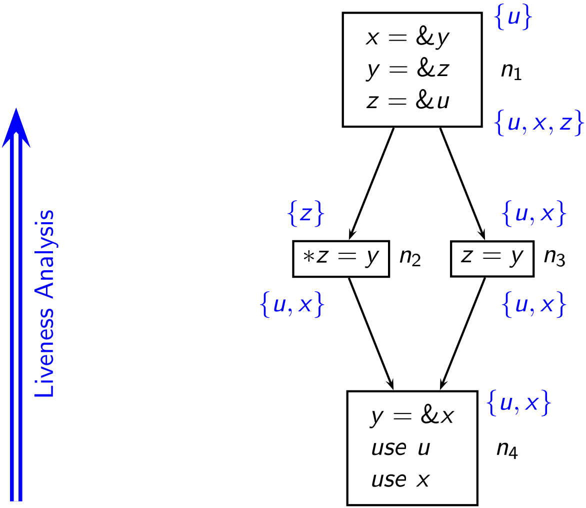


Strong liveness: y is not made live because z is not live

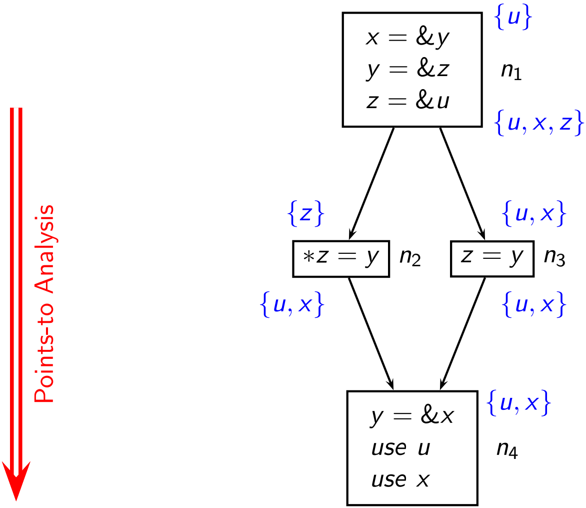
First Round of Liveness Analysis and Points-to Analysis



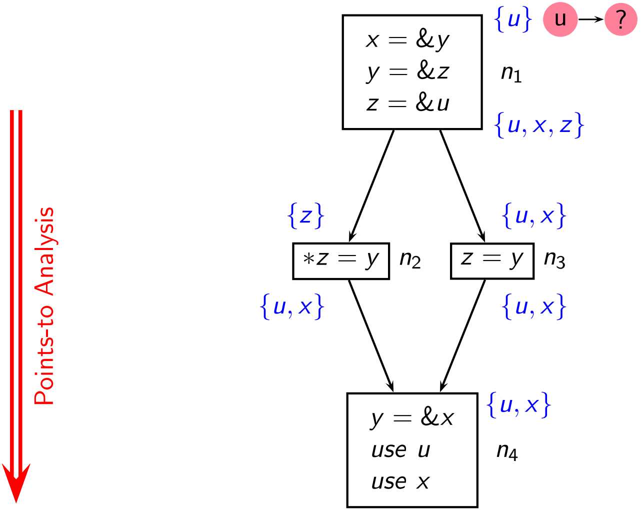
First Round of Liveness Analysis and Points-to Analysis



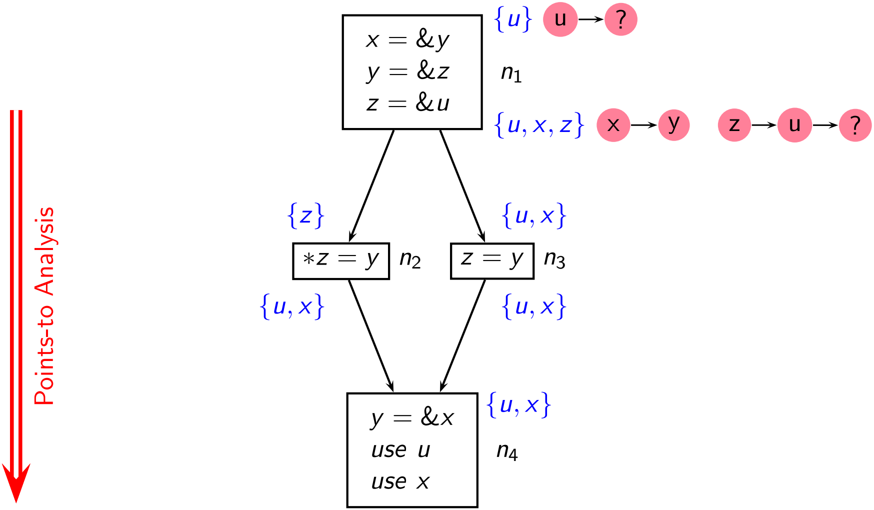
First Round of Liveness Analysis and Points-to Analysis



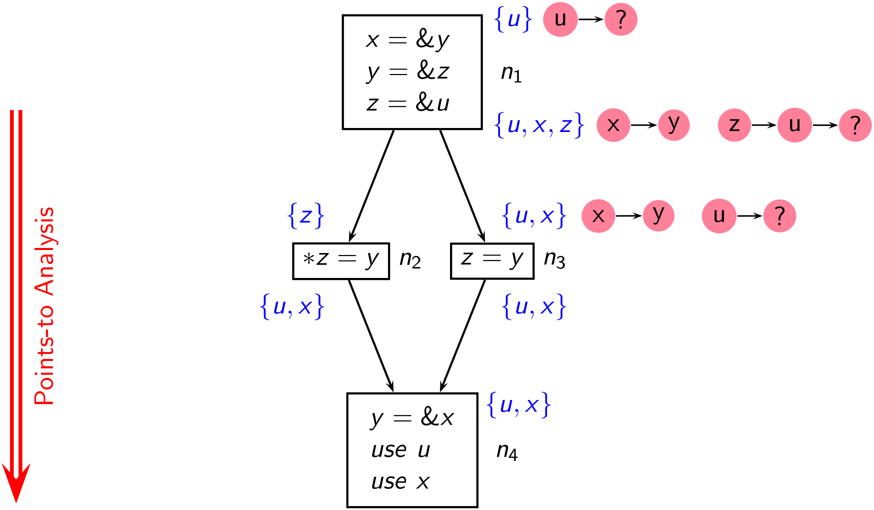
First Round of Liveness Analysis and Points-to Analysis



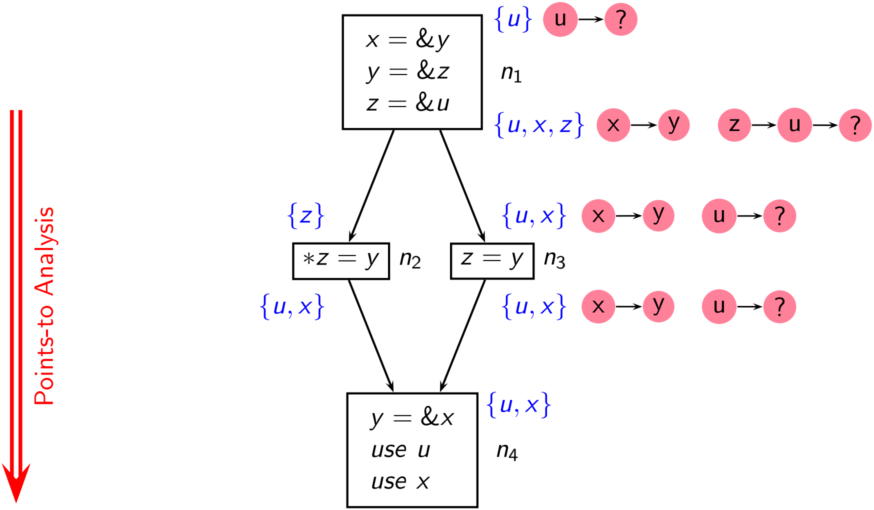
First Round of Liveness Analysis and Points-to Analysis



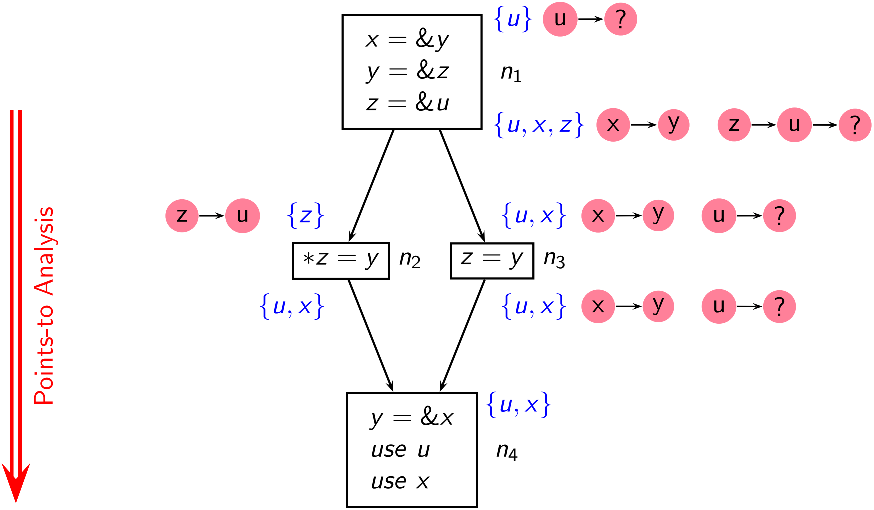
First Round of Liveness Analysis and Points-to Analysis



First Round of Liveness Analysis and Points-to Analysis



First Round of Liveness Analysis and Points-to Analysis



Points-to Analysis

$\{u\}$ u → ?

$x = \&y$
$y = \&z$
$z = \&u$

n_1

$\{u, x, z\}$ x → y z → u → ?

z → u $\{z\}$

$*z = y$ n_2

$\{u, x\}$

$\{u, x\}$ x → y u → ?

$z = y$ n_3

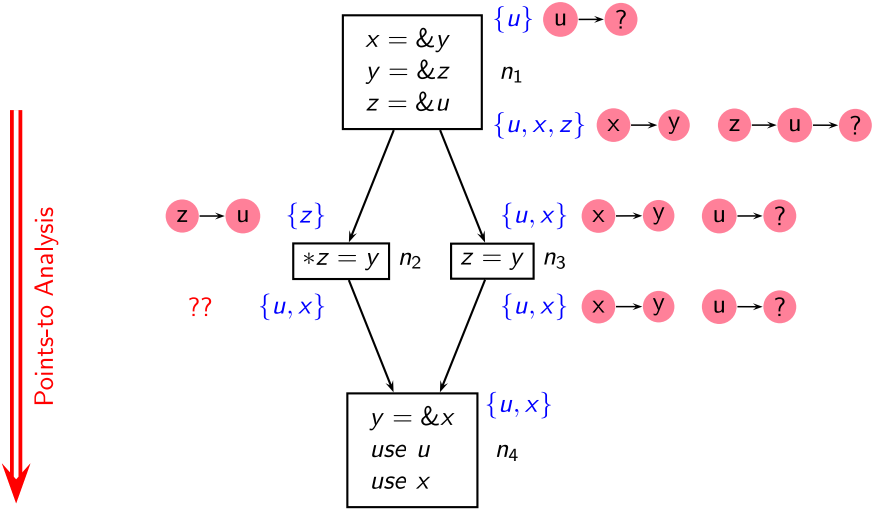
$\{u, x\}$ x → y u → ?

$\{u, x\}$

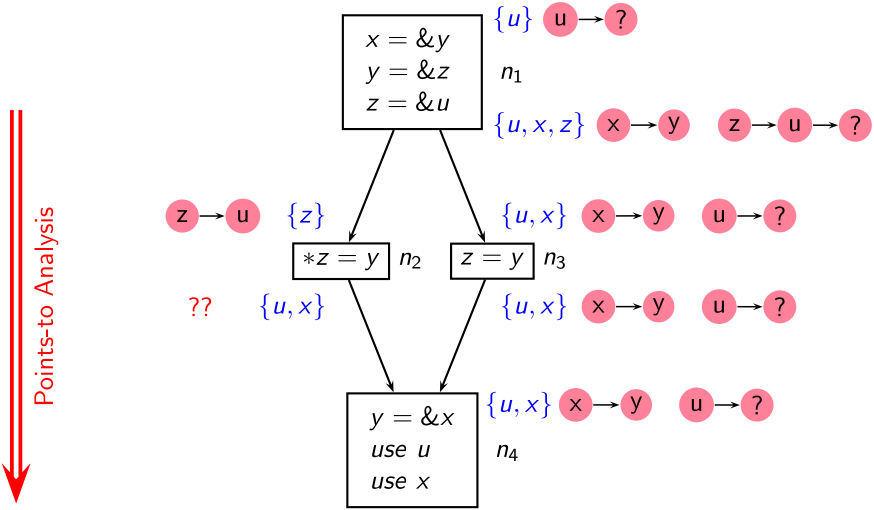
$y = \&x$
$use\ u$
$use\ x$

n_4

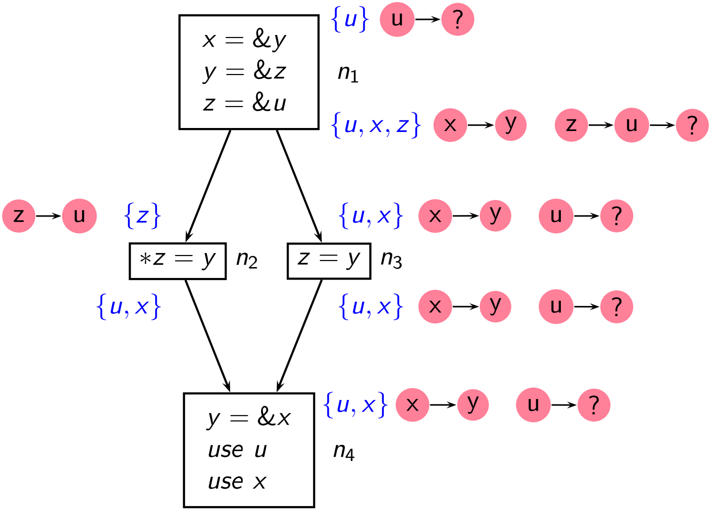
First Round of Liveness Analysis and Points-to Analysis



First Round of Liveness Analysis and Points-to Analysis



Second Round of Liveness Analysis and Points-to Analysis



n_1: $x = \&y$, $y = \&z$, $z = \&u$

$\{u\}$ u → ?

$\{u, x, z\}$ x → y z → u → ?

n_2: $*z = y$

z → u $\{z\}$

$\{u, x\}$

n_3: $z = y$

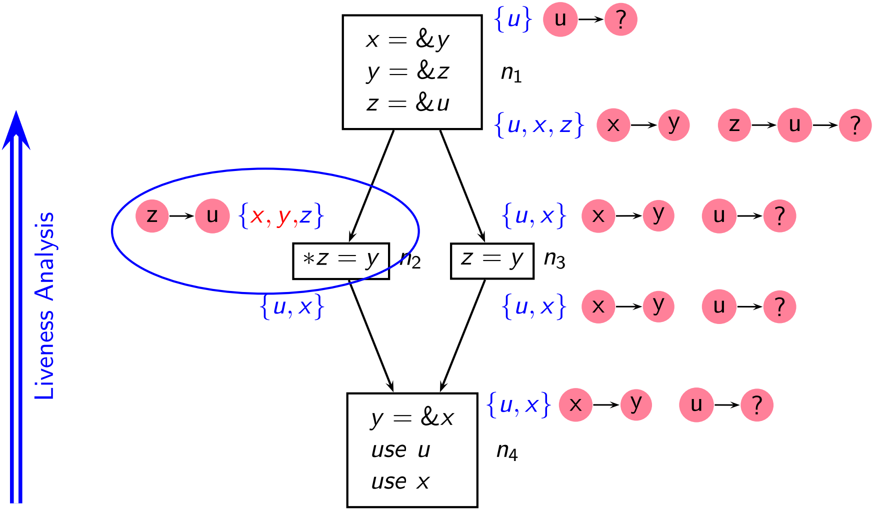
$\{u, x\}$ x → y u → ?

$\{u, x\}$ x → y u → ?

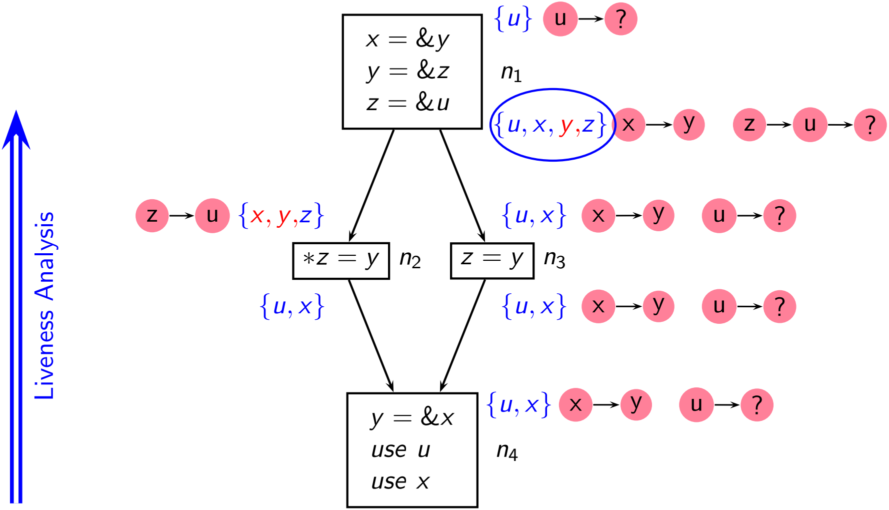
n_4: $y = \&x$, use u, use x

$\{u, x\}$ x → y u → ?

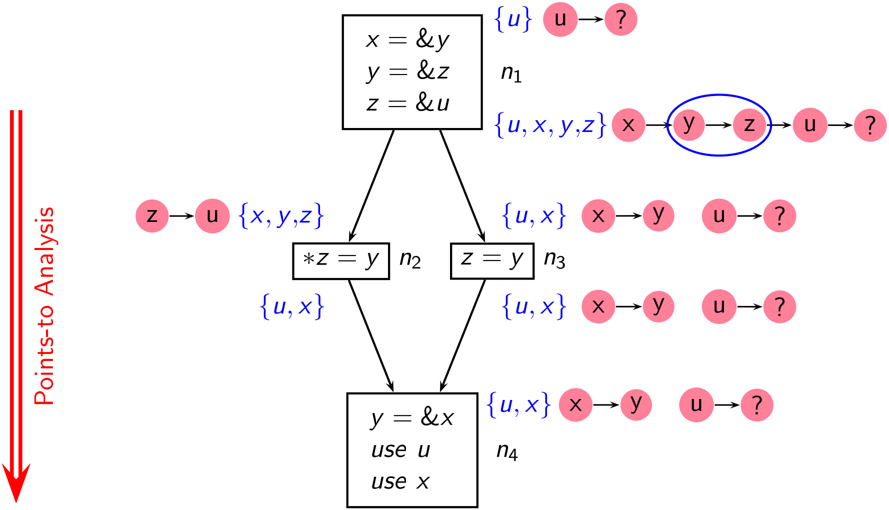
Second Round of Liveness Analysis and Points-to Analysis



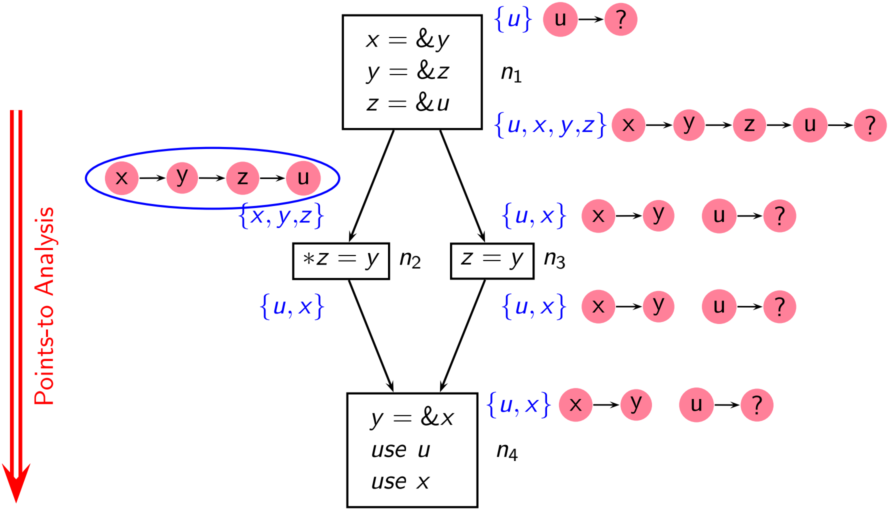
Second Round of Liveness Analysis and Points-to Analysis



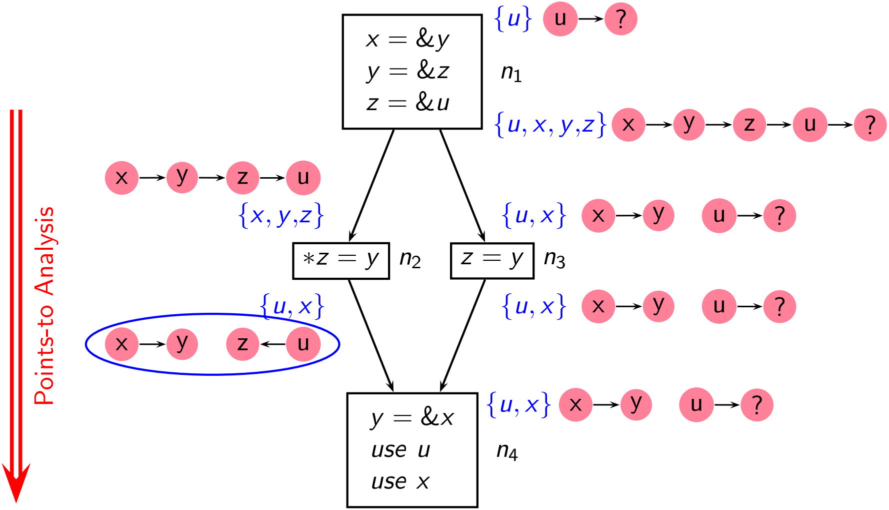
Second Round of Liveness Analysis and Points-to Analysis



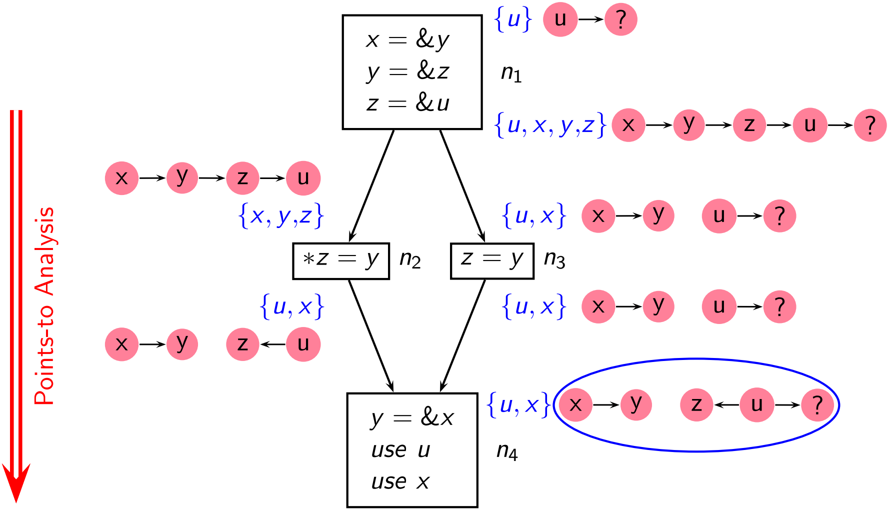
Second Round of Liveness Analysis and Points-to Analysis



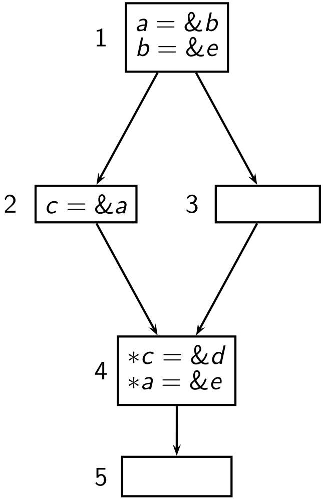
Second Round of Liveness Analysis and Points-to Analysis



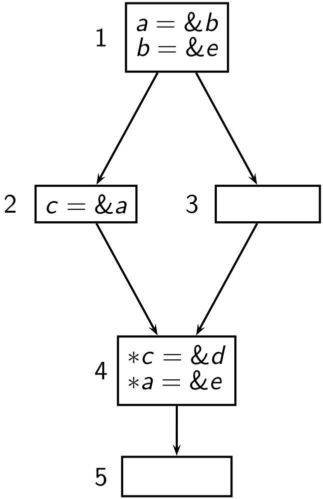
Second Round of Liveness Analysis and Points-to Analysis



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

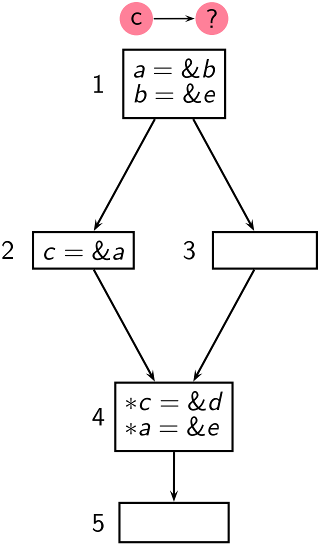


Discovering Must Points-to Information from May Points-to Information



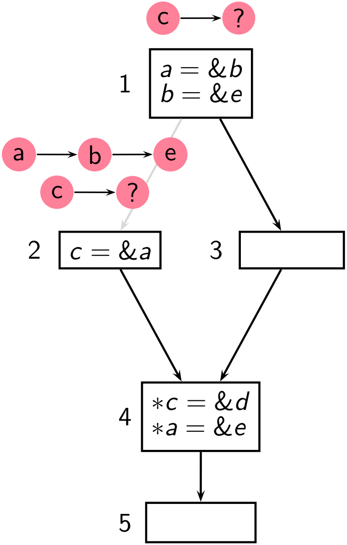
- c is live at program entry

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



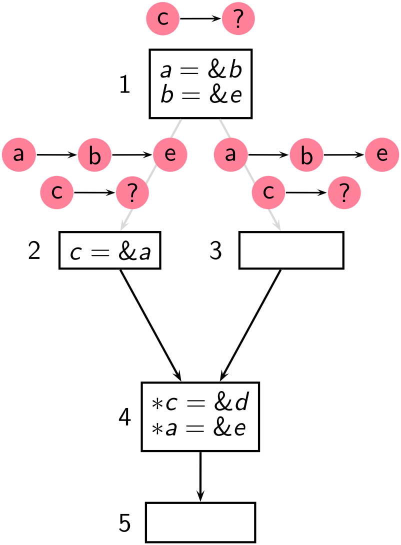
- c is live at program entry
- Assume that c points to “?” at program entry

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



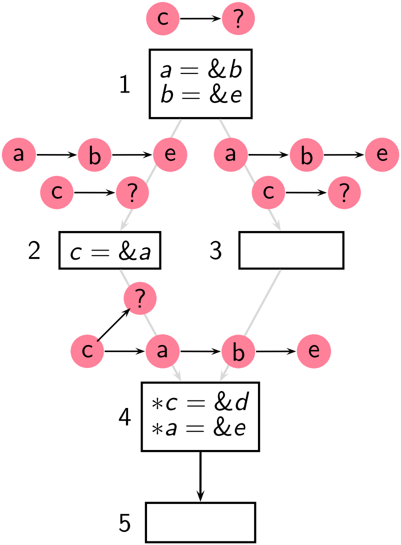
- c is live at program entry
- Assume that c points to "?" at program entry
- Perform usual may points-to analysis

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



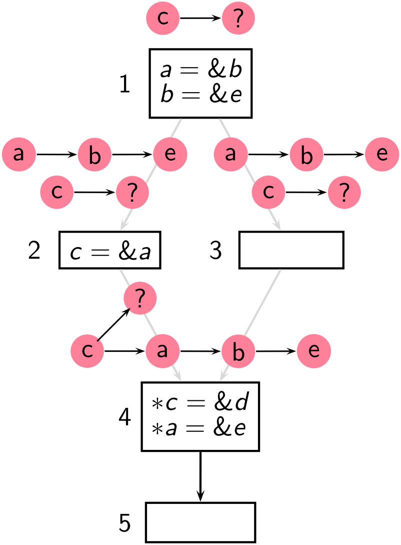
- c is live at program entry
- Assume that c points to "?" at program entry
- Perform usual may points-to analysis

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



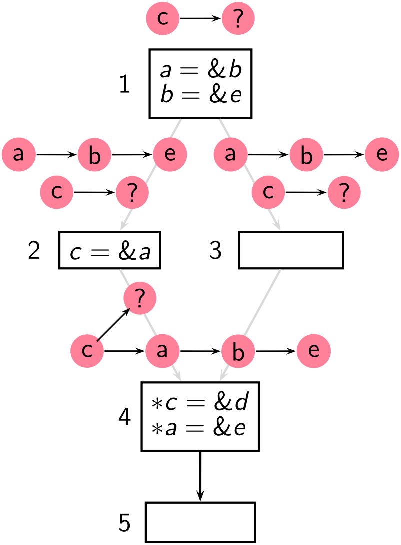
- c is live at program entry
- Assume that c points to “?” at program entry
- Perform usual may points-to analysis

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



- c is live at program entry
- Assume that c points to “?” at program entry
- 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



- c is live at program entry
- Assume that c points to "?" at program entry
- 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

Part 4

Interprocedural Analysis

Call Strings Method Using Value Based Termination

- The classical Sharir-Pnueli call string method with a small change in the termination criteria
  - Classical approach [Sharir, Pnueli. 1981]
    Construct all call strings upto the length $K \cdot (|L| + 1)^2$
    - L is the lattice of data flow values and K is the maximum number of distinct call sites in any call chain
    - This bound is for general frameworks. For simpler frameworks such as separable or bit vector frameworks, the bounds are smaller
  - Our approach [Khedker, Karkare. 2008]
    Use equivalence of data flow values

A Points-to Analysis Example to Show the Difference

```
main()
{  x = &y;
   z = &x;
   y = &z;
   p(); /* C1 */
}

p()
{  if (...)
   {  p(); /* C2 */
      x = *x;
   }
}
```

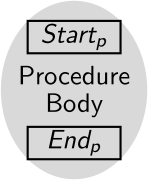
- Number of distinct call sites in a call chain $K = 2$.
- Number of variables: 3
- Number of distinct points-to pairs: $3 \times 3 = 9$
- L is powerset of all points-to pairs
- $\mid L \mid = 2^9$
- Length of the longest call string in Sharir-Pnueli method $2 \times (|L| + 1)^2 = 2^{19} + 2^{10} + 1 = 5, 25, 313$
- All call strings upto this length must be constructed by the Sharir-Pnueli method!

A Points-to Analysis Example to Show the Difference

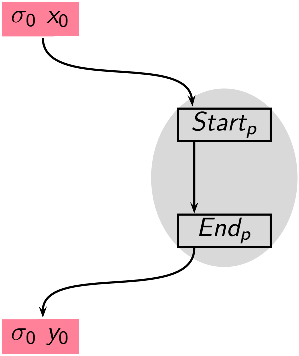
```
main()
{  x = &y;
   z = &x;
   y = &z;
   p(); /* C1 */
}

p()
{  if (...)
   {  p(); /* C2 */
      x = *x;
   }
}
```

- Value based termination requires only three call strings: λ, c_1, and c_1c_2

Value Based Termination of Call String Construction

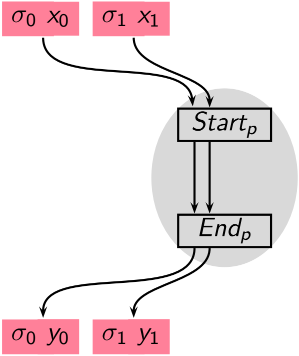


Value Based Termination of Call String Construction



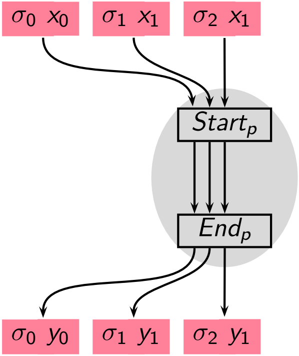
- Context sensitive analysis retains distinct data values for each context reaching a procedure

Value Based Termination of Call String Construction



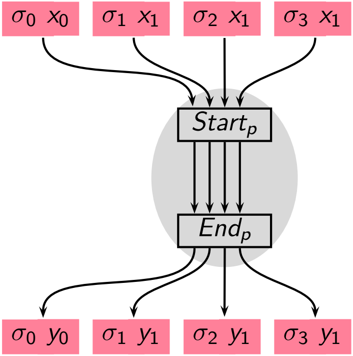
- Context sensitive analysis retains distinct data values for each context reaching a procedure

Value Based Termination of Call String Construction



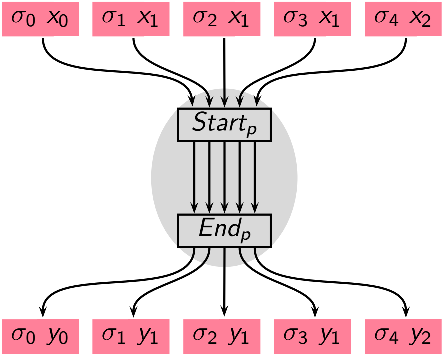
- Context sensitive analysis retains distinct data values for each context reaching a procedure

Value Based Termination of Call String Construction



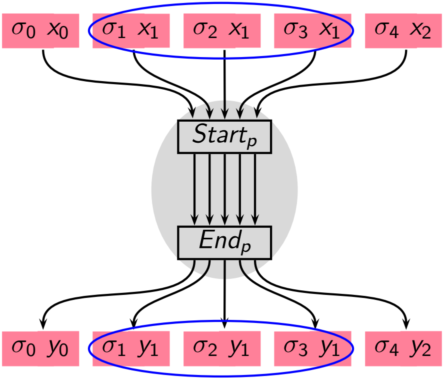
- Context sensitive analysis retains distinct data values for each context reaching a procedure

Value Based Termination of Call String Construction



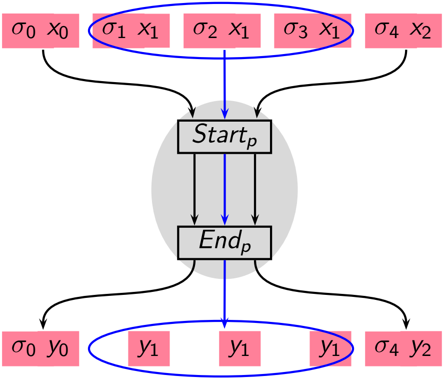
- Context sensitive analysis retains distinct data values for each context reaching a procedure

Value Based Termination of Call String Construction



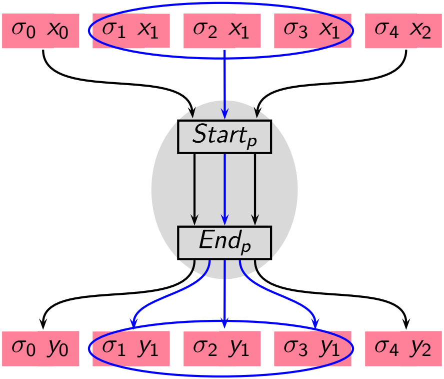
- Context sensitive analysis retains distinct data values for each context reaching a procedure
- Many data flow values could be identical

Value Based Termination of Call String Construction



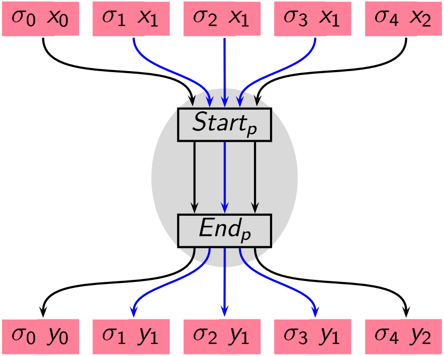
- Context sensitive analysis retains distinct data values for each context reaching a procedure
- Many data flow values could be identical
- It is sufficient to propagate a single representative data flow value

Value Based Termination of Call String Construction



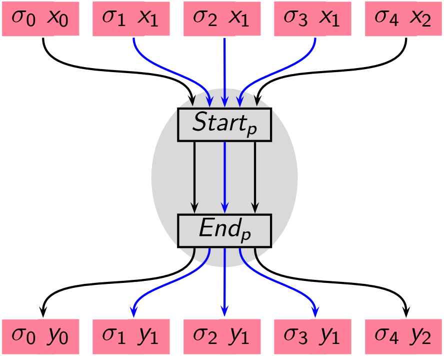
- Context sensitive analysis retains distinct data values for each context reaching a procedure
- Many data flow values could be identical
- It is sufficient to propagate a single representative data flow value
- We only need to regenerate the missing contexts

Value Based Termination of Call String Construction



- Context sensitive analysis retains distinct data values for each context reaching a procedure
- Many data flow values could be identical
- It is sufficient to propagate a single representative data flow value
- We only need to regenerate the missing contexts
- Much fewer call strings are passed on to the callees

Value Based Termination of Call String Construction



- Context sensitive analysis retains distinct data values for each context reaching a procedure
- Many data flow values could be identical
- It is sufficient to propagate a single representative data flow value
- We only need to regenerate the missing contexts
- Much fewer call strings are passed on to the callees

The number of call strings is reduced without any loss of precision

Value Based Termination of Call String Construction

- Seem straight forward for non-recursive procedures
- What if a procedure is recursive?

Value Based Termination of Call String Construction

- Seem straight forward for non-recursive procedures
- What if a procedure is recursive?
- Read our CC 2008 paper, or my book, or my extra slides ☺

Part 5

Measurements

Implementation

- LTO framework of GCC 4.6.0
- Naive prototype implementation
  (Points-to sets implemented using linked lists)
- Implemented FCPA without liveness for comparison
- Comparison with GCC's flow and context insensitive method
- SPEC 2006 benchmarks

Analysis Time

Program	kLoC	Call Sites	Time in milliseconds			
			L-FCPA		FCPA	GPTA
			Liveness	Points-to		
lbm	0.9	33	0.55	0.52	1.9	5.2
mcf	1.6	29	1.04	0.62	9.5	3.4
libquantum	2.6	258	2.0	1.8	5.6	4.8
bzip2	3.7	233	4.5	4.8	28.1	30.2
parser	7.7	1123	1.2×10^3	145.6	4.3×10^5	422.12
sjeng	10.5	678	858.2	99.0	3.2×10^4	38.1
hmmer	20.6	1292	90.0	62.9	2.9×10^5	246.3
h264ref	36.0	1992	2.2×10^5	2.0×10^5	?	4.3×10^3

Unique Points-to Pairs

Program	kLoC	Call Sites	Unique points-to pairs		
			L-FCPA	FCPA	GPTA
lbm	0.9	33	12	507	1911
mcf	1.6	29	41	367	2159
libquantum	2.6	258	49	119	2701
bzip2	3.7	233	60	210	8.8×10^4
parser	7.7	1123	531	4196	1.9×10^4
sjeng	10.5	678	267	818	1.1×10^4
hmmer	20.6	1292	232	5805	1.9×10^6
h264ref	36.0	1992	1683	?	1.6×10^7

Precise Context Information is Small and Sparse

Program	Total no. of functions	No. and percentage of functions for call-string counts							
		0 call strings		1-4 call strings		5-8 call strings		9+ call strings	
		L-FCPA	FCPA	L-FCPA	FCPA	L-FCPA	FCPA	L-FCPA	FCPA
lbm	22	16 (72.7%)	3 (13.6%)	6 (27.3%)	19 (86.4%)	0	0	0	0
mcf	25	16 (64.0%)	3 (12.0%)	9 (36.0%)	22 (88.0%)	0	0	0	0
bzip2	100	88 (88.0%)	38 (38.0%)	12 (12.0%)	62 (62.0%)	0	0	0	0
libquantum	118	100 (84.7%)	56 (47.5%)	17 (14.4%)	62 (52.5%)	1 (0.8%)	0	0	0
sjeng	151	96 (63.6%)	37 (24.5%)	43 (28.5%)	45 (29.8%)	12 (7.9%)	15 (9.9%)	0	54 (35.8%)
hmmer	584	548 (93.8%)	330 (56.5%)	32 (5.5%)	175 (30.0%)	4 (0.7%)	26 (4.5%)	0	53 (9.1%)
parser	372	246 (66.1%)	76 (20.4%)	118 (31.7%)	135 (36.3%)	4 (1.1%)	63 (16.9%)	4 (1.1%)	98 (26.3%)
	9+ call strings in L-FCPA: Tot 4, Min 10, Max 52, Mean 32.5, Median 29, Mode 10								
h264ref	624	351 (56.2%)	?	240 (38.5%)	?	14 (2.2%)	?	19 (3.0%)	?
	9+ call strings in L-FCPA: Tot 14, Min 9, Max 56, Mean 27.9, Median 24, Mode 9								

Precise Usable Pointer Information is Small and Sparse

Program	Total	No. and percentage of basic blocks (BBs) for points-to (pt) pair counts							
	no. of	0 pt pairs		1-4 pt pairs		5-8 pt pairs		9+ pt pairs	
	BBs	L-FCPA	FCPA	L-FCPA	FCPA	L-FCPA	FCPA	L-FCPA	FCPA
lbm	252	229 (90.9%)	61 (24.2%)	23 (9.1%)	82 (32.5%)	0	66 (26.2%)	0	43 (17.1%)
mcf	472	356 (75.4%)	160 (33.9%)	116 (24.6%)	2 (0.4%)	0	1 (0.2%)	0	309 (65.5%)
libquantum	1642	1520 (92.6%)	793 (48.3%)	119 (7.2%)	796 (48.5%)	3 (0.2%)	46 (2.8%)	0	7 (0.4%)
bzip2	2746	2624 (95.6%)	1085 (39.5%)	118 (4.3%)	12 (0.4%)	3 (0.1%)	12 (0.4%)	1 (0.0%)	1637 (59.6%)
	9+ pt pairs in L-FCPA: Tot 1, Min 12, Max 12, Mean 12.0, Median 12, Mode 12								
sjeng	6000	4571 (76.2%)	3239 (54.0%)	1208 (20.1%)	12 (0.2%)	221 (3.7%)	41 (0.7%)	0	2708 (45.1%)
hmmer	14418	13483 (93.5%)	8357 (58.0%)	896 (6.2%)	21 (0.1%)	24 (0.2%)	91 (0.6%)	15 (0.1%)	5949 (41.3%)
	9+ pt pairs in L-FCPA: Tot 6, Min 10, Max 16, Mean 13.3, Median 13, Mode 10								
parser	6875	4823 (70.2%)	1821 (26.5%)	1591 (23.1%)	25 (0.4%)	252 (3.7%)	154 (2.2%)	209 (3.0%)	4875 (70.9%)
	9+ pt pairs in L-FCPA: Tot 13, Min 9, Max 53, Mean 27.9, Median 18, Mode 9								
h264ref	21315	13729 (64.4%)	?	4760 (22.3%)	?	2035 (9.5%)	?	791 (3.7%)	?
	9+ pt pairs in L-FCPA: Tot 44, Min 9, Max 98, Mean 36.3, Median 31, Mode 9								

Part 6

Conclusions

Observations

- Usable pointer information is very small and sparse
- Data flow propagation in real programs seems to involve only a small subset of all possible data flow values
- Earlier approaches reported inefficiency and non-scalability because they computed far more information than the actual usable information

Conclusions

- Building quick approximations and compromising on precision may not be necessary for efficiency
- Building clean abstractions to separate the necessary information from redundant information is much more significant

Conclusions

- Building quick approximations and compromising on precision may not be necessary for efficiency
- Building clean abstractions to separate the necessary information from redundant information is much more significant

  Our experience of points-to analysis shows that

  - Use of liveness reduced the pointer information . . .
  - which reduced the number of contexts required . . .
  - which reduced the liveness and pointer information . . .

Conclusions

- Building quick approximations and compromising on precision may not be necessary for efficiency
- Building clean abstractions to separate the necessary information from redundant information is much more significant

  Our experience of points-to analysis shows that

  - Use of liveness reduced the pointer information ...
  - which reduced the number of contexts required ...
  - which reduced the liveness and pointer information ...
- Approximations should come *after* building abstractions rather than *before*

Future Work

- Redesign data structures by hiding them behind APIs
  Current version uses linked lists and linear search
- Incremental version
- Using precise pointer information in other passes in GCC
- Extend it to precise alias analysis of heap data

Parting Thoughts: The Larger Perspective

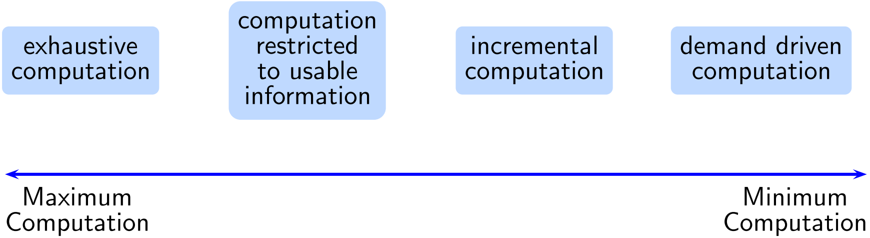
exhaustive computation

computation restricted to usable information

incremental computation

demand driven computation

Parting Thoughts: The Larger Perspective



exhaustive computation

computation restricted to usable information

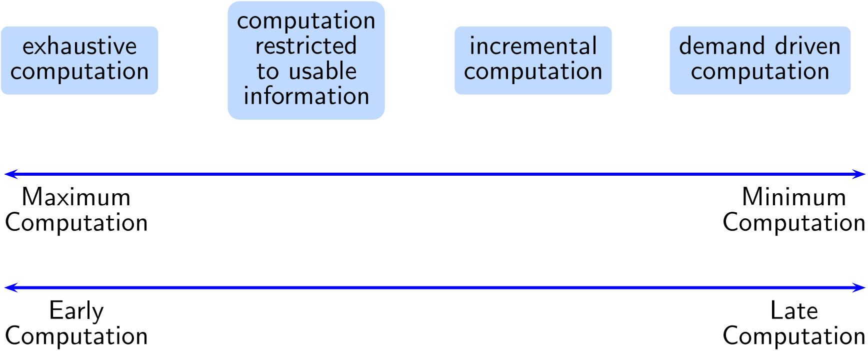
incremental computation

demand driven computation

Maximum Computation

Minimum Computation

Parting Thoughts: The Larger Perspective



exhaustive computation

computation restricted to usable information

incremental computation

demand driven computation

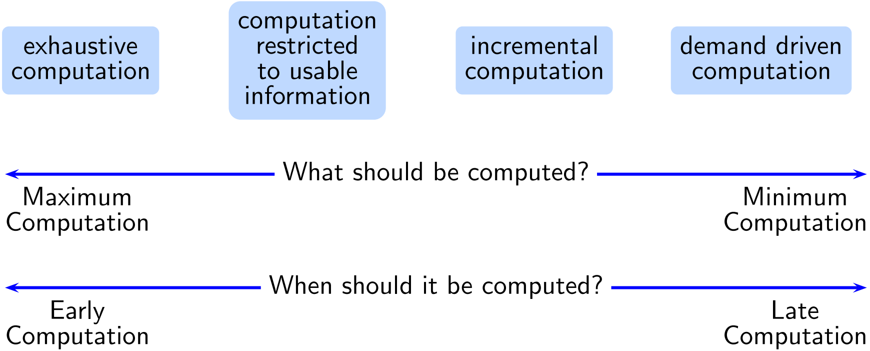
Maximum Computation

Minimum Computation

Early Computation

Late Computation

Parting Thoughts: The Larger Perspective



exhaustive computation

computation restricted to usable information

incremental computation

demand driven computation

What should be computed?

Maximum Computation

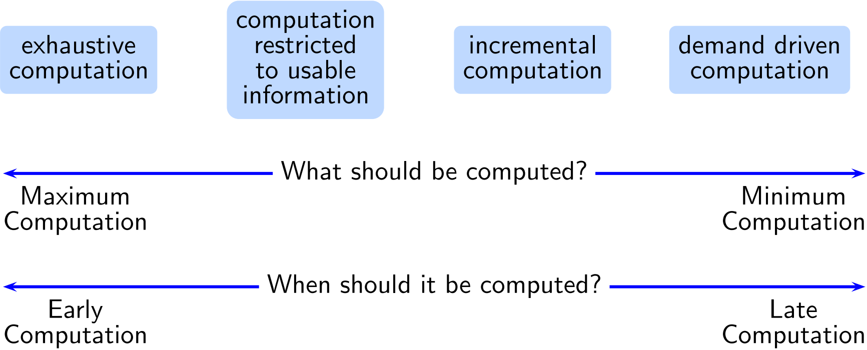
Minimum Computation

When should it be computed?

Early Computation

Late Computation

Parting Thoughts: The Larger Perspective



exhaustive computation

computation restricted to usable information

incremental computation

demand driven computation

What should be computed?

Maximum Computation

Minimum Computation

When should it be computed?

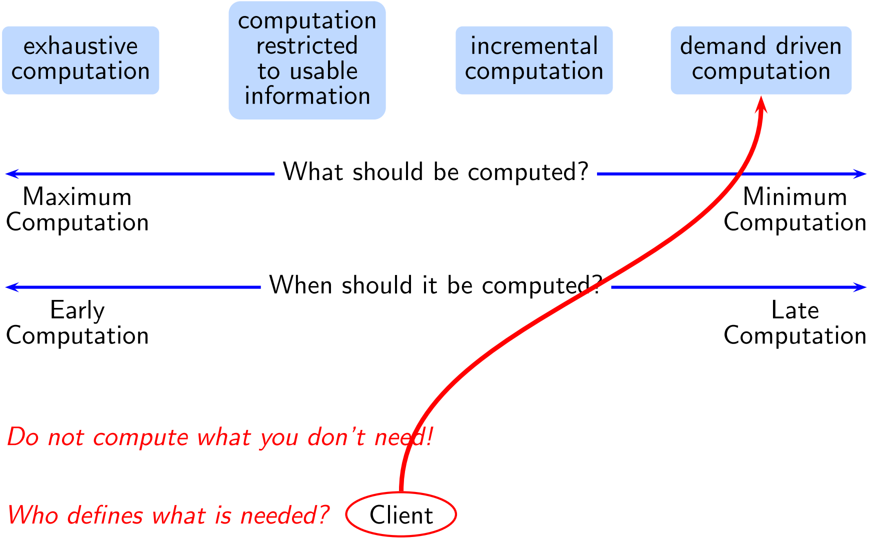
Early Computation

Late Computation

Do not compute what you don't need!

Who defines what is needed?

Parting Thoughts: The Larger Perspective



exhaustive computation

computation restricted to usable information

incremental computation

demand driven computation

What should be computed?

Maximum Computation

Minimum Computation

When should it be computed?

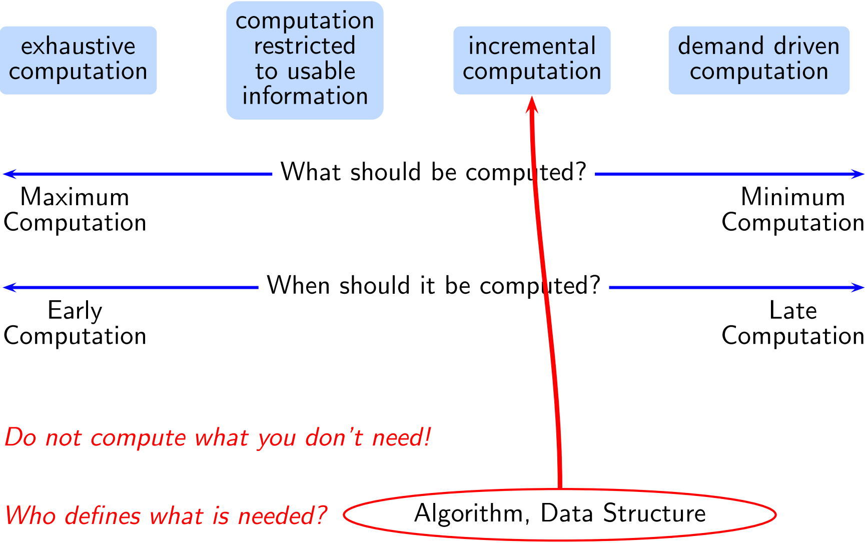
Early Computation

Late Computation

Do not compute what you don't need!

Who defines what is needed? Client

Parting Thoughts: The Larger Perspective



exhaustive computation

computation restricted to usable information

incremental computation

demand driven computation

What should be computed?

Maximum Computation

Minimum Computation

When should it be computed?

Early Computation

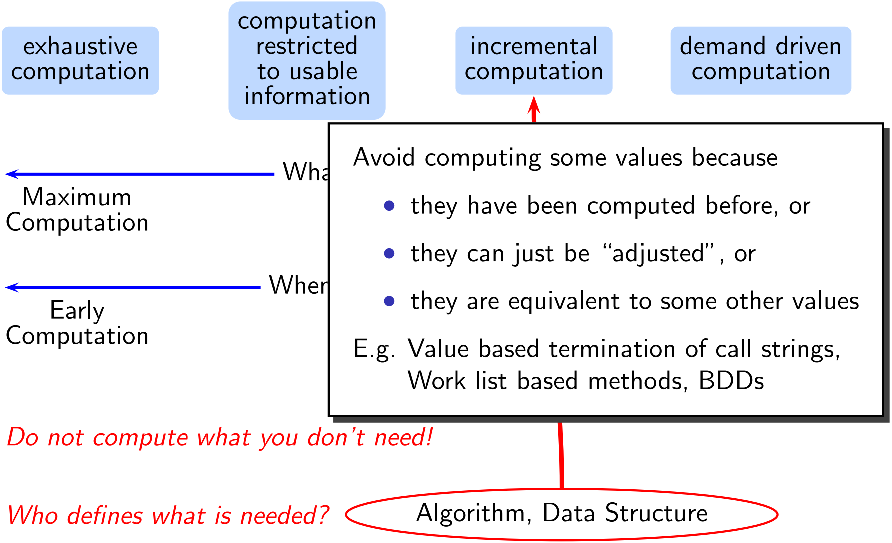
Late Computation

Do not compute what you don't need!

Who defines what is needed?

Algorithm, Data Structure

Parting Thoughts: The Larger Perspective



exhaustive computation

computation restricted to usable information

incremental computation

demand driven computation

Wha

Maximum Computation

Wher

Early Computation

Avoid computing some values because

- they have been computed before, or
- they can just be “adjusted”, or
- they are equivalent to some other values

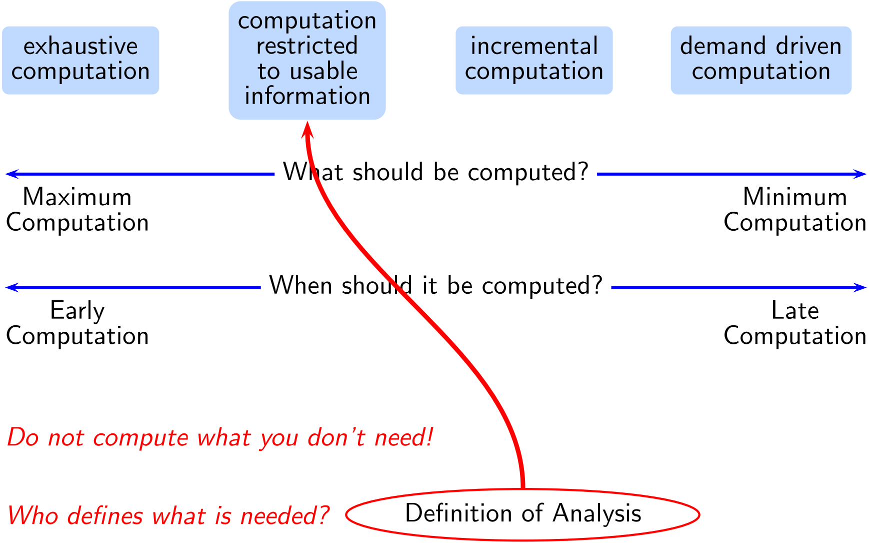
E.g. Value based termination of call strings, Work list based methods, BDDs

Do not compute what you don’t need!

Who defines what is needed?

Algorithm, Data Structure

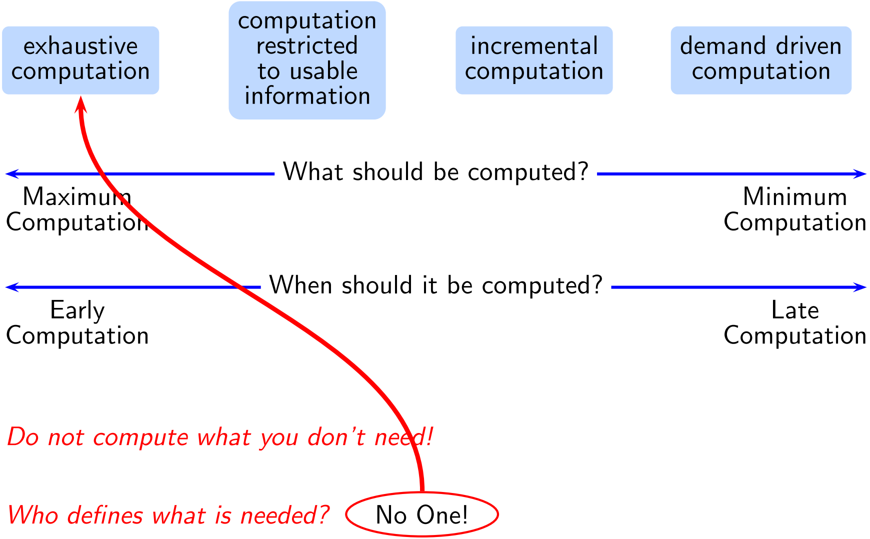
Parting Thoughts: The Larger Perspective



Do not compute what you don't need!

Who defines what is needed?

Parting Thoughts: The Larger Perspective



exhaustive computation

computation restricted to usable information

incremental computation

demand driven computation

What should be computed?

Maximum Computation

Minimum Computation

When should it be computed?

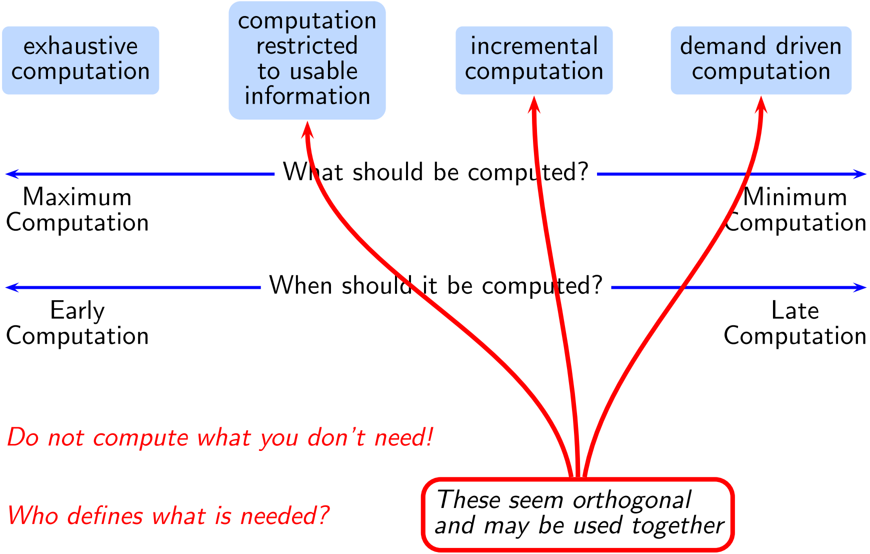
Early Computation

Late Computation

Do not compute what you don't need!

Who defines what is needed? No One!

Parting Thoughts: The Larger Perspective


exhaustive computation
computation restricted to usable information
incremental computation
demand driven computation
What should be computed?
Maximum Computation
Minimum Computation
When should it be computed?
Early Computation
Late Computation
These seem orthogonal and may be used together

Do not compute what you don't need!

Who defines what is needed?

Last But Not the Least

Thank You!