Theoretical Abstractions in Data Flow Analysis

Uday Khedker

(www.cse.iitb.ac.in/˜uday)

Department of Computer Science and Engineering,

Indian Institute of Technology, Bombay

August 2018

Part 1

About These Slides

Copyright

These slides constitute the lecture notes for CS618 Program Analysis course at IIT Bombay and have been made available as teaching material accompanying the book:

- Uday Khedker, Amitabha Sanyal, and Bageshri Karkare. *Data Flow Analysis: Theory and Practice*. CRC Press (Taylor and Francis Group). 2009.

  (Indian edition published by Ane Books in 2013)

Apart from the above book, some slides are based on the material from the following books

- M. S. Hecht. *Flow Analysis of Computer Programs*. Elsevier North-Holland Inc. 1977.
- F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer-Verlag. 1998.

Outline

- The need for a more general setting
- The set of data flow values
- The set of flow functions
- Solutions of data flow analyses
- Algorithms for performing data flow analysis
- Complexity of data flow analysis
- On Soundness and Precision of data flow analysis

Part 2

The Need for a More General Setting

What We Have Seen So Far . . .

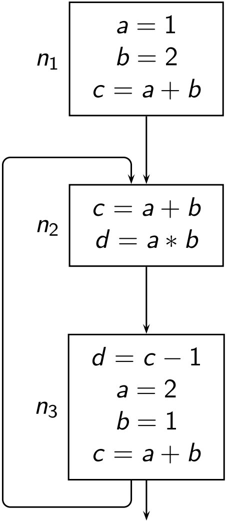
Analysis	Entity	Attribute at p	Paths	
Live variables	Variables	Use	Starting at p	Some
Available expressions	Expressions	Availability	Reaching p	All
Partially available expressions	Expressions	Availability	Reaching p	Some
Anticipable expressions	Expressions	Use	Starting at p	All
Reaching definitions	Definitions	Availability	Reaching p	Some
Partial redundancy elimination	Expressions	Profitable hoistability	Involving p	All

The Need for a More General Setting

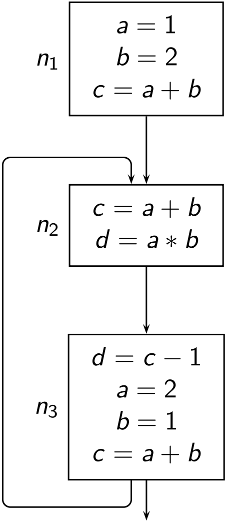
- We seem to have covered many variations
- Yet there are analyses that do not fit the same mould of bit vector frameworks
- We use an analysis called *Constant Propagation* to observe the differences

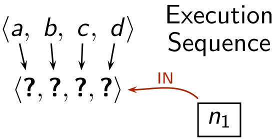
A variable v is a constant with value c at program point p if in every execution instance of p, the value of v is c

An Introduction to Constant Propagation

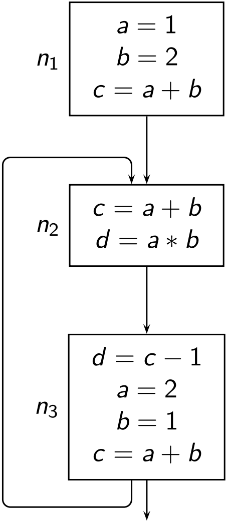


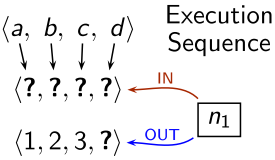
An Introduction to Constant Propagation



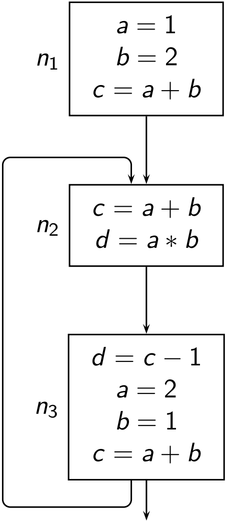


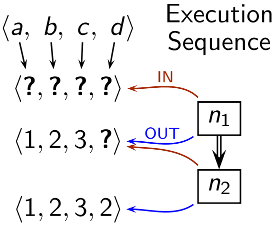
An Introduction to Constant Propagation



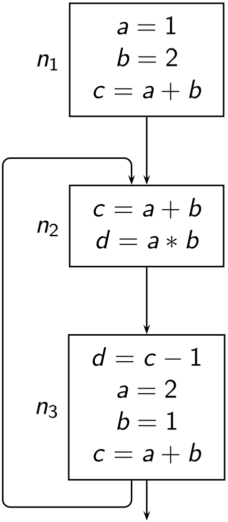


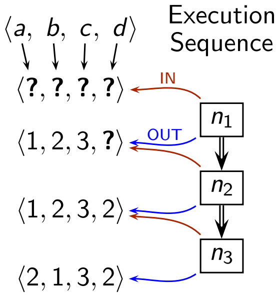
An Introduction to Constant Propagation



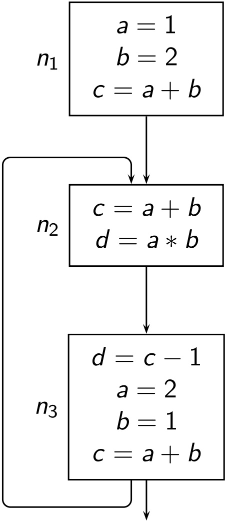


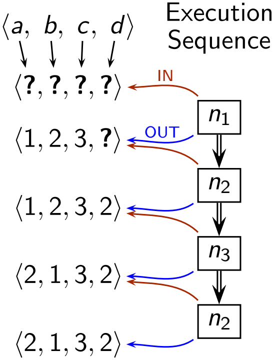
An Introduction to Constant Propagation



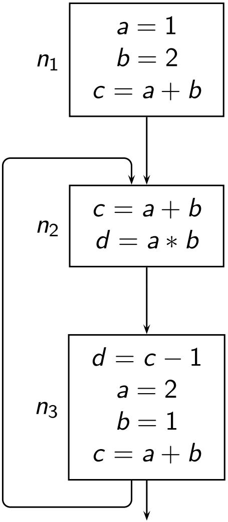


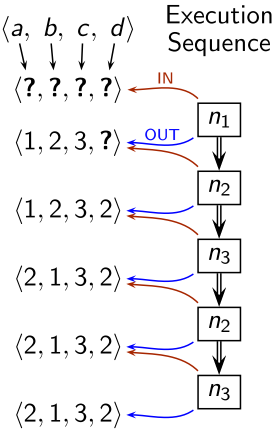
An Introduction to Constant Propagation



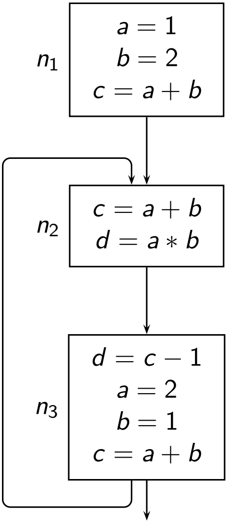


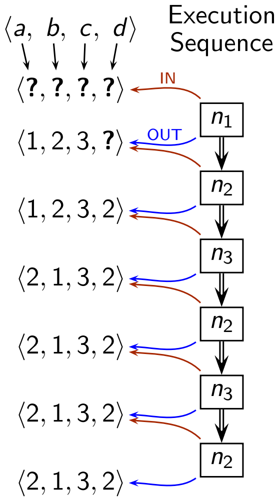
An Introduction to Constant Propagation



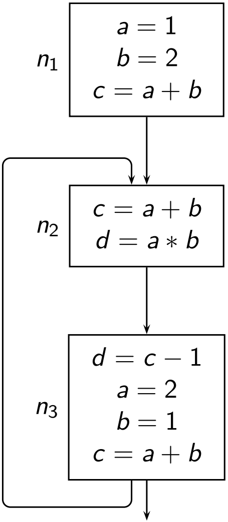


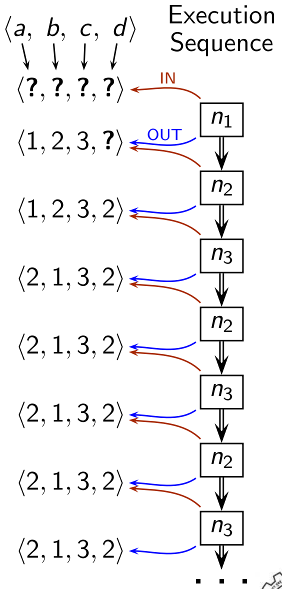
An Introduction to Constant Propagation



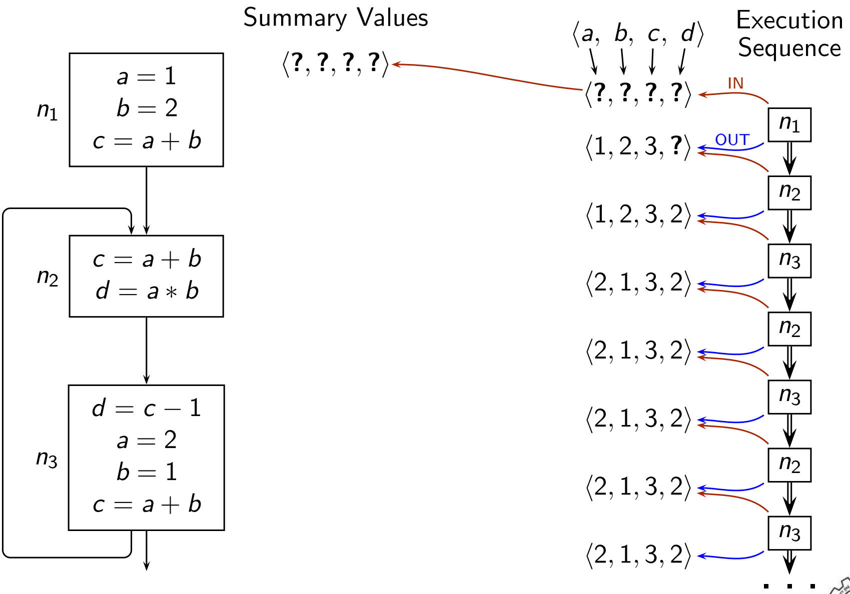


An Introduction to Constant Propagation

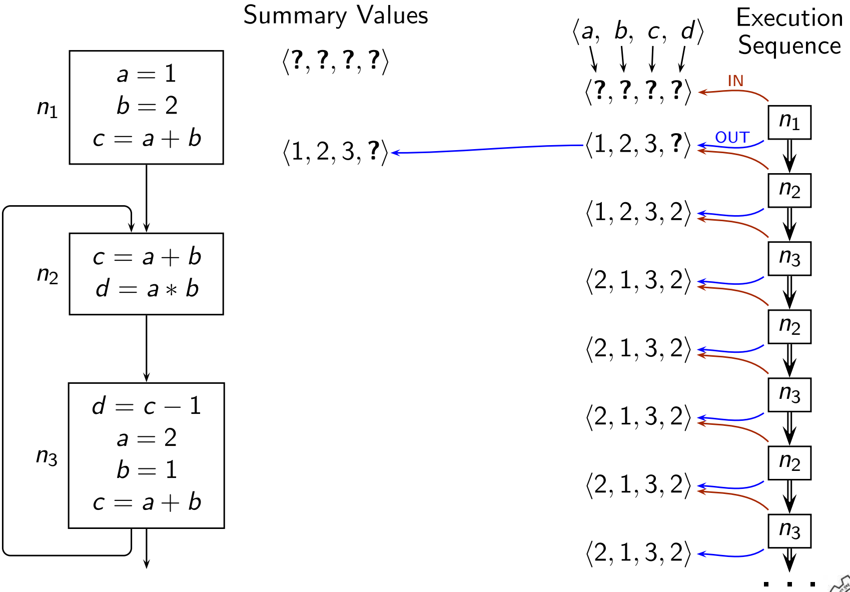




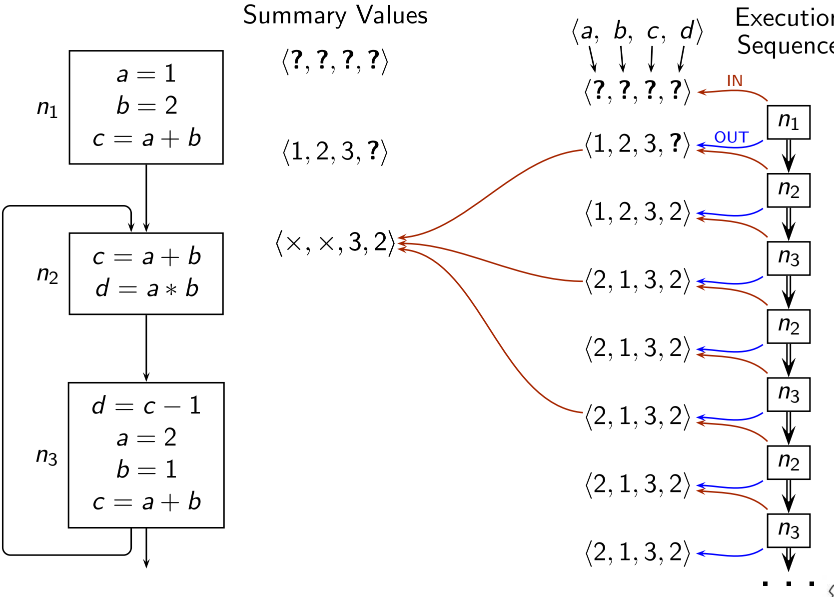
An Introduction to Constant Propagation



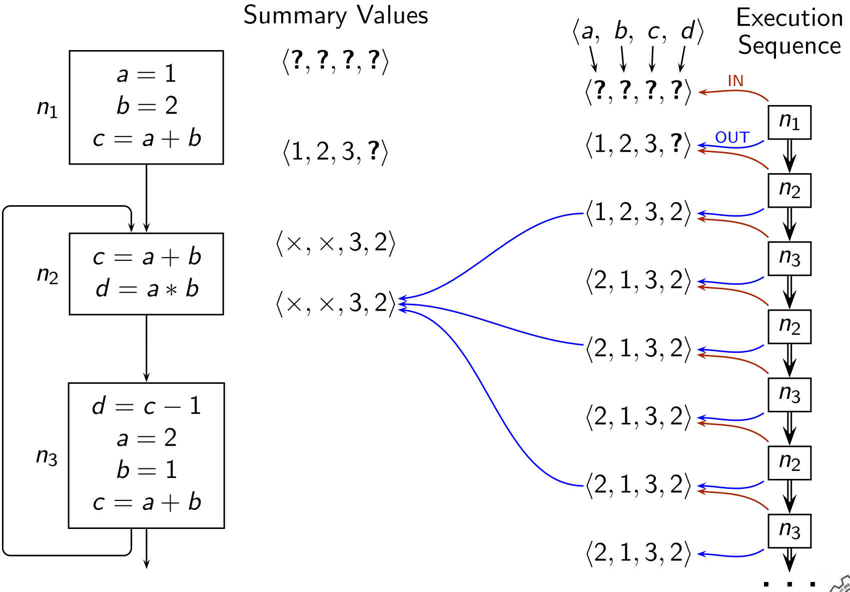
An Introduction to Constant Propagation



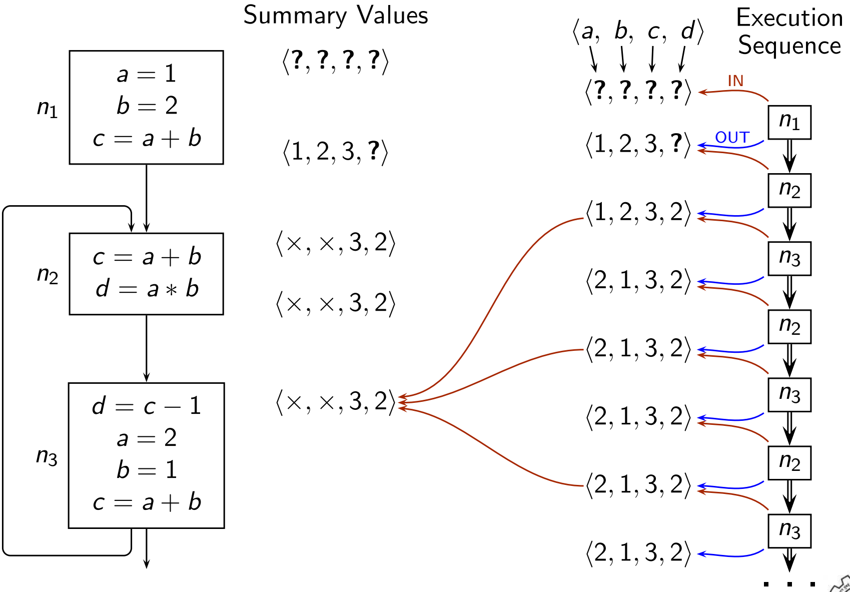
An Introduction to Constant Propagation



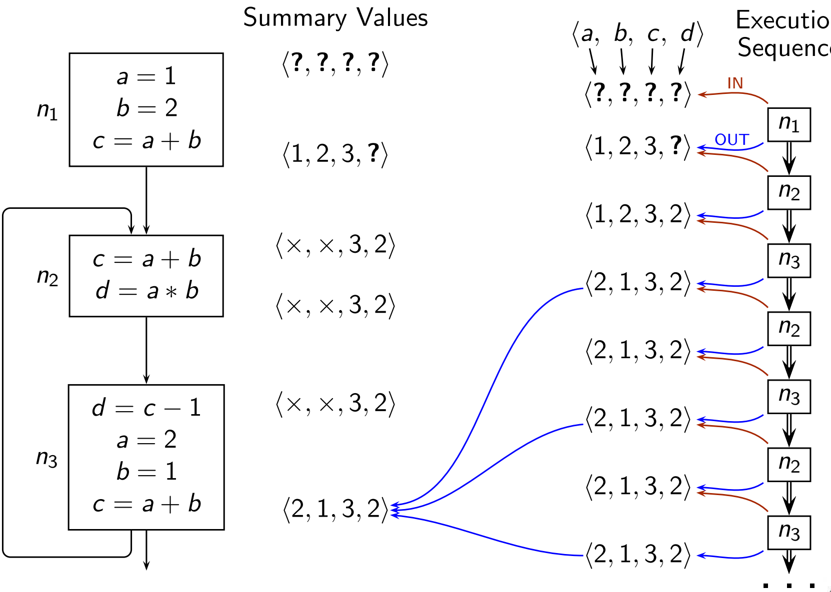
An Introduction to Constant Propagation



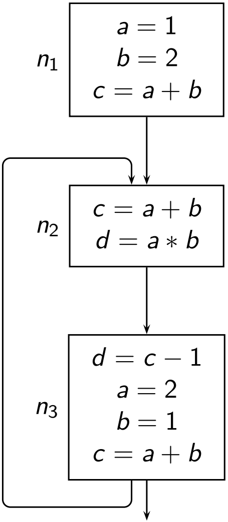
An Introduction to Constant Propagation



An Introduction to Constant Propagation



An Introduction to Constant Propagation



Summary Values

$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle$

$\langle 1, 2, 3, \mathbf{?} \rangle$

$\langle \times, \times, 3, 2 \rangle$

$\langle \times, \times, 3, 2 \rangle$

$\langle \times, \times, 3, 2 \rangle$

$\langle 2, 1, 3, 2 \rangle$

Desired Solution

Difference #1: Data Flow Values

- Tuples of the form $\langle \eta_1, \eta_2, \ldots, \eta_k \rangle$
  Or sets of pairs $(v_i, \eta_i))$ where
  η_i is the data flow value for i^{th} variable

  Unlike bit vector frameworks, value η_i is not 0 or 1 (i.e. true or false). Instead, it is one of the following:

  - **?** indicating that no values is known for v_i
  - $\times$ indicating that variable v_i could have multiple values
  - An integer constant c_1 if the value of v_i is known to be c_1 at compile time

Difference #2: Dependence of Data Flow Values Across Entities

- In bit vector frameworks, data flow values of different entities are independent

Difference #2: Dependence of Data Flow Values Across Entities

- In bit vector frameworks, data flow values of different entities are independent
  - Liveness of variable b does not depend on that of any other variable
  - Availability of expression $a * b$ does not depend on that of any other expression

Difference #2: Dependence of Data Flow Values Across Entities

- In bit vector frameworks, data flow values of different entities are independent
  - Liveness of variable b does not depend on that of any other variable
  - Availability of expression $a * b$ does not depend on that of any other expression
- Given a statement $a = b * c$, can the constantness of a be determined independently of the constantness of b and c?

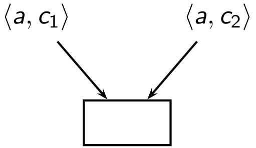
Difference #2: Dependence of Data Flow Values Across Entities

- In bit vector frameworks, data flow values of different entities are independent
  - Liveness of variable b does not depend on that of any other variable
  - Availability of expression $a * b$ does not depend on that of any other expression
- Given a statement $a = b * c$, can the constantness of a be determined independently of the constantness of b and c?

No

Difference #3: Confluence Operation

- Confluence operation $\langle a, c_1 \rangle \sqcap \langle a, c_2 \rangle$



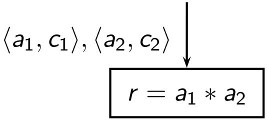
$\sqcap$	$\langle a, ? \rangle$	$\langle a, \times \rangle$	$\langle a, c_1 \rangle$
$\langle a, ? \rangle$	$\langle a, ? \rangle$	$\langle a, \times \rangle$	$\langle a, c_1 \rangle$
$\langle a, \times \rangle$	$\langle a, \times \rangle$	$\langle a, \times \rangle$	$\langle a, \times \rangle$
$\langle a, c_2 \rangle$	$\langle a, c_2 \rangle$	$\langle a, \times \rangle$	If $c_1 = c_2$ $\langle a, c_1 \rangle$ Otherwise $\langle a, \times \rangle$

- This is neither $\cap$ nor $\cup$

  What are its properties?

Difference #4: Flow Functions for Constant Propagation

- Flow function for $r = a_1 * a_2$



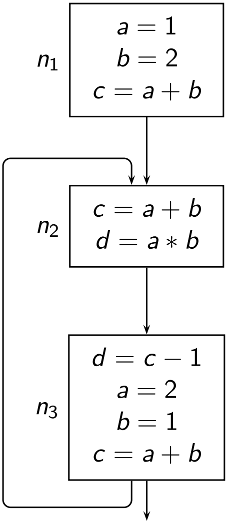
mult	$\langle a_1, \mathbf{?}\rangle$	$\langle a_1, \times\rangle$	$\langle a_1, c_1\rangle$
$\langle a_2, \mathbf{?}\rangle$	$\langle r, \mathbf{?}\rangle$	$\langle r, \times\rangle$	$\langle r, \mathbf{?}\rangle$
$\langle a_2, \times\rangle$	$\langle r, \times\rangle$	$\langle r, \times\rangle$	$\langle r, \times\rangle$
$\langle a_2, c_2\rangle$	$\langle r, \mathbf{?}\rangle$	$\langle r, \times\rangle$	$\langle r, (c_1 * c_2)\rangle$

- This cannot be expressed in the form

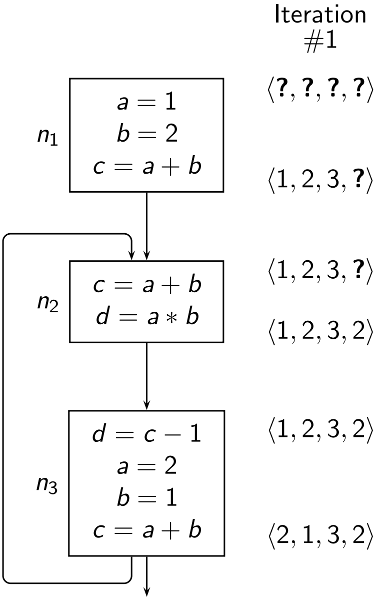
$$f_n(X) = \text{Gen}_n \cup (X - \text{Kill}_n)$$

where Gen_n and Kill_n are constant effects of block n

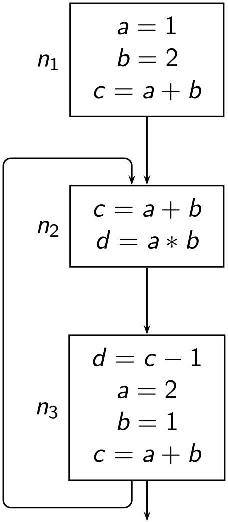
Difference #5: Solution Computed by Iterative Method



Difference #5: Solution Computed by Iterative Method

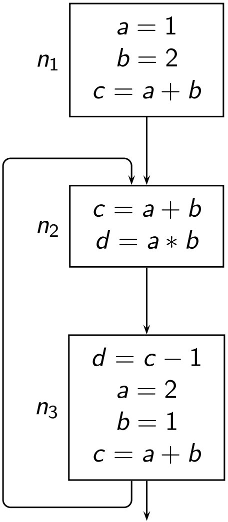


Difference #5: Solution Computed by Iterative Method



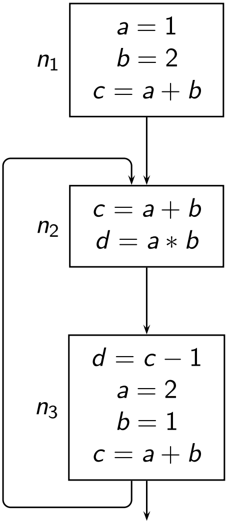
Iteration #1	Iteration #2
$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle$	$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle$
$\langle 1, 2, 3, \mathbf{?} \rangle$	$\langle 1, 2, 3, \mathbf{?} \rangle$
$\langle 1, 2, 3, \mathbf{?} \rangle$	$\langle \times, \times, 3, 2 \rangle$
$\langle 1, 2, 3, 2 \rangle$	$\langle \times, \times, \times, \times \rangle$
$\langle 1, 2, 3, 2 \rangle$	$\langle \times, \times, \times, \times \rangle$
$\langle 2, 1, 3, 2 \rangle$	$\langle 2, 1, 3, \times \rangle$

Difference #5: Solution Computed by Iterative Method



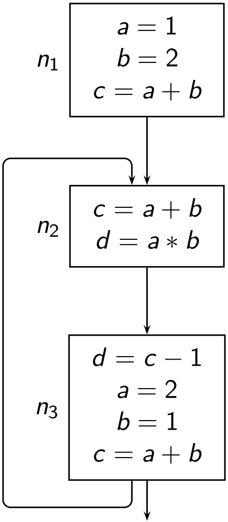
Iteration #1	Iteration #2	Iteration #3
$\langle \textbf{?}, \textbf{?}, \textbf{?}, \textbf{?} \rangle$	$\langle \textbf{?}, \textbf{?}, \textbf{?}, \textbf{?} \rangle$	$\langle \textbf{?}, \textbf{?}, \textbf{?}, \textbf{?} \rangle$
$\langle 1, 2, 3, \textbf{?} \rangle$	$\langle 1, 2, 3, \textbf{?} \rangle$	$\langle 1, 2, 3, \textbf{?} \rangle$
$\langle 1, 2, 3, \textbf{?} \rangle$	$\langle \times, \times, 3, 2 \rangle$	$\langle \times, \times, 3, \times \rangle$
$\langle 1, 2, 3, 2 \rangle$	$\langle \times, \times, \times, \times \rangle$	$\langle \times, \times, \times, \times \rangle$
$\langle 1, 2, 3, 2 \rangle$	$\langle \times, \times, \times, \times \rangle$	$\langle \times, \times, \times, \times \rangle$
$\langle 2, 1, 3, 2 \rangle$	$\langle 2, 1, 3, \times \rangle$	$\langle 2, 1, 3, \times \rangle$

Difference #5: Solution Computed by Iterative Method



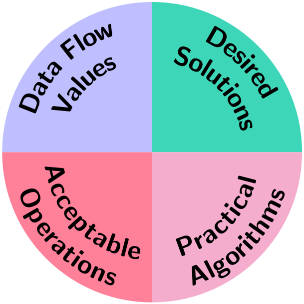
Iteration #1	Iteration #2	Iteration #3	Desired solution
$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle$	$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle$	$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle$	$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle$
$\langle 1, 2, 3, \mathbf{?} \rangle$	$\langle 1, 2, 3, \mathbf{?} \rangle$	$\langle 1, 2, 3, \mathbf{?} \rangle$	$\langle 1, 2, 3, \mathbf{?} \rangle$
$\langle 1, 2, 3, \mathbf{?} \rangle$	$\langle \times, \times, 3, 2 \rangle$	$\langle \times, \times, 3, \times \rangle$	$\langle \times, \times, 3, 2 \rangle$
$\langle 1, 2, 3, 2 \rangle$	$\langle \times, \times, \times, \times \rangle$	$\langle \times, \times, \times, \times \rangle$	$\langle \times, \times, 3, 2 \rangle$
$\langle 1, 2, 3, 2 \rangle$	$\langle \times, \times, \times, \times \rangle$	$\langle \times, \times, \times, \times \rangle$	$\langle \times, \times, 3, 2 \rangle$
$\langle 2, 1, 3, 2 \rangle$	$\langle 2, 1, 3, \times \rangle$	$\langle 2, 1, 3, \times \rangle$	$\langle 2, 1, 3, 2 \rangle$

Difference #5: Solution Computed by Iterative Method



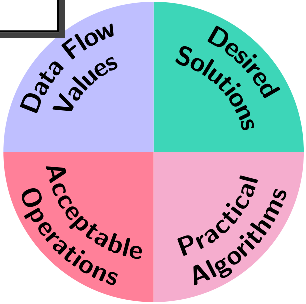
Iteration #1	Iteration #2	Iteration #3	Desired solution
$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle$	$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle$	$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle$	$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle$
$\langle 1, 2, 3, \mathbf{?} \rangle$	$\langle 1, 2, 3, \mathbf{?} \rangle$	$\langle 1, 2, 3, \mathbf{?} \rangle$	$\langle 1, 2, 3, \mathbf{?} \rangle$
$\langle 1, 2, 3, \mathbf{?} \rangle$	$\langle \times, \times, 3, 2 \rangle$	$\langle \times, \times, 3, \times \rangle$	$\langle \times, \times, 3, 2 \rangle$
$\langle 1, 2, 3, 2 \rangle$	$\langle \times, \times, \times, \times \rangle$	$\langle \times, \times, \times, \times \rangle$	$\langle \times, \times, 3, 2 \rangle$
$\langle 1, 2, 3, 2 \rangle$	$\langle \times, \times, \times, \times \rangle$	$\langle \times, \times, \times, \times \rangle$	$\langle \times, \times, 3, 2 \rangle$
$\langle 2, 1, 3, 2 \rangle$	$\langle 2, 1, 3, \times \rangle$	$\langle 2, 1, 3, \times \rangle$	$\langle 2, 1, 3, 2 \rangle$

Issues in Data Flow Analysis



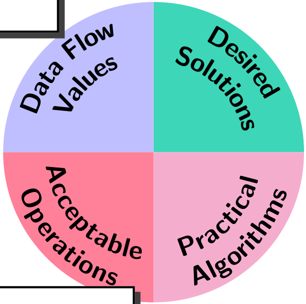
Issues in Data Flow Analysis

- Representation
- Approximation: Partial Order, Lattices



Issues in Data Flow Analysis

- Representation
- Approximation: Partial Order, Lattices

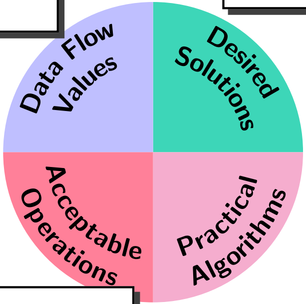


- Merge: Commutativity, Associativity, Idempotence
- Flow Functions: Monotonicity, Distributivity, Boundedness, Separability

Issues in Data Flow Analysis

- Representation
- Approximation: Partial Order, Lattices

- Existence, Computability
- Soundness, Precision

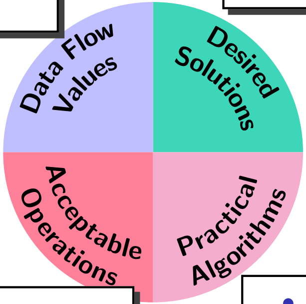


- Merge: Commutativity, Associativity, Idempotence
- Flow Functions: Monotonicity, Distributivity, Boundedness, Separability

Issues in Data Flow Analysis

- Representation
- Approximation: Partial Order, Lattices

- Existence, Computability
- Soundness, Precision



- Merge: Commutativity, Associativity, Idempotence
- Flow Functions: Monotonicity, Distributivity, Boundedness, Separability

- Complexity, efficiency
- Convergence
- Initialization

Part 3

Data Flow Values: An Overview

Data Flow Values: An Outline of Our Discussion

- The need to define the notion of abstraction
- Lattices, variants of lattices
- Relevance of lattices for data flow analysis
  - Partial order relation as approximation of data flow values
  - Meet operations as confluence of data flow values
- Constructing lattices
- Example of lattices

Part 4

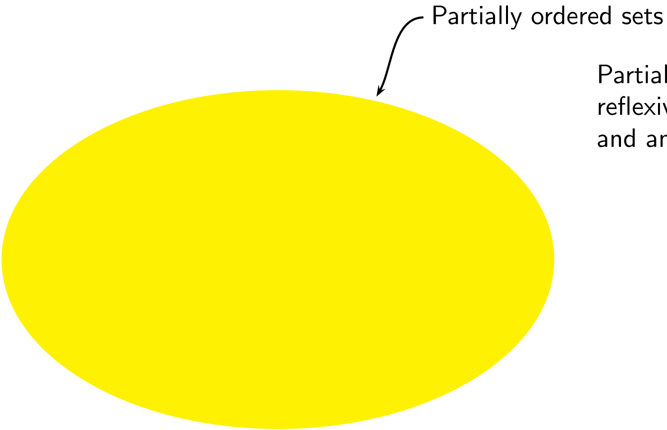
A Digression on Lattices

Partially Ordered Sets

Sets in which elements can be compared and ordered

- *Total order*. Every element in comparable with every element (including itself)
- *Discrete order*. Every element is comparable only with itself but not with any other element
- *Partial order*. An element is comparable with itself and some other elements but not necessarily with all elements

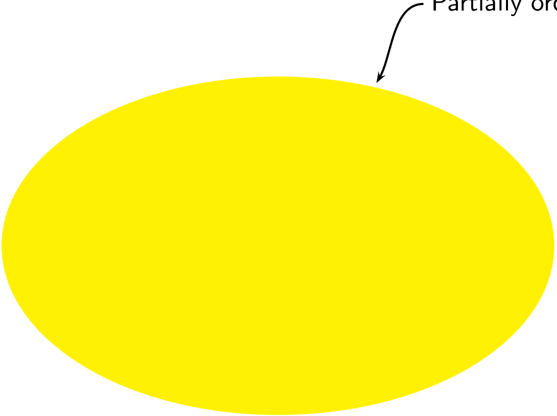
Partially Ordered Sets and Lattices



Partial order $\sqsubseteq$ is reflexive, transitive, and antisymmetric

Partially Ordered Sets and Lattices

Partially ordered sets

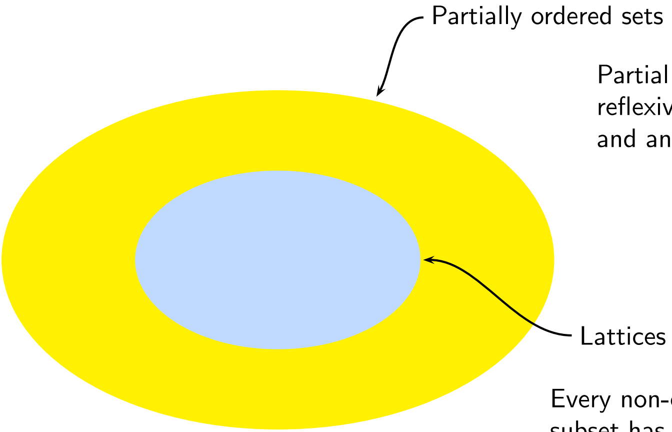


Partial order $\sqsubseteq$ is reflexive, transitive, and antisymmetric

A lower bound of x, y is u s.t. $u \sqsubseteq x$ and $u \sqsubseteq y$

An upper bound of x, y is u s.t. $x \sqsubseteq u$ and $y \sqsubseteq u$

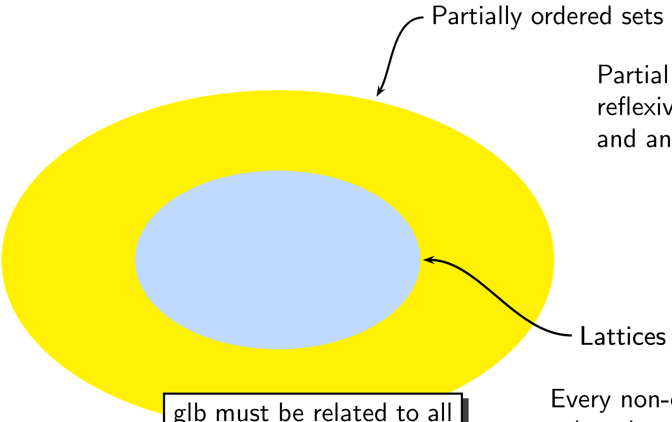
Partially Ordered Sets and Lattices



Partial order $\sqsubseteq$ is reflexive, transitive, and antisymmetric

Every non-empty finite subset has a greatest lower bound (glb) and a least upper bound (lub)

Partially Ordered Sets and Lattices



Partial order $\sqsubseteq$ is reflexive, transitive, and antisymmetric

Every non-empty finite subset has a greatest lower bound (glb) and a least upper bound (lub)

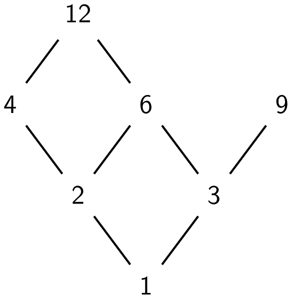
glb must be related to all other lower bounds. Hence it must be unique

Partially Ordered Sets

Set $\{1, 2, 3, 4, 6, 9, 12\}$ with $\sqsubseteq$ relation as "divides" (i.e. $a \sqsubseteq b$ iff a divides b)

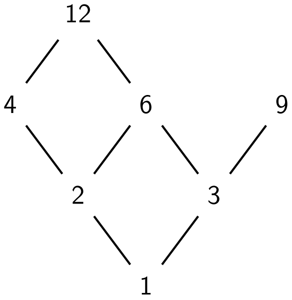
Partially Ordered Sets

Set $\{1, 2, 3, 4, 6, 9, 12\}$ with $\sqsubseteq$ relation as "divides" (i.e. $a \sqsubseteq b$ iff a divides b)



Partially Ordered Sets

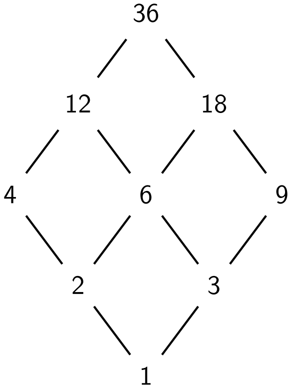
Set $\{1, 2, 3, 4, 6, 9, 12\}$ with $\sqsubseteq$ relation as "divides" (i.e. $a \sqsubseteq b$ iff a divides b)



Subset $\{4, 9, 6\}$ and $\{12, 9\}$ do not have an upper bound in the set

Lattice

Set $\{1, 2, 3, 4, 6, 9, 12, 18, 36\}$ with $\sqsubseteq$ relation as "divides"



Examples of Orderings on Strings

- Consider relations between strings in Σ^* over alphabet $\Sigma = \{a_1, a_2, \ldots, a_n\}$
  - The prefix, suffix, and substring relations are partial orders
  - If Σ is totally ordered, then the lexicographic order $\preceq$ is a total order
    Let $u, v, x, y, z \in \Sigma^*$, and let $a_i, a_j \in \Sigma$

$$u \preceq v \;\Leftrightarrow\; (v = u\,y) \vee (u = xa_iy \wedge v = xa_jz \wedge a_i < a_j)$$

Complete Lattice

- Lattice: A partially ordered set such that every non-empty finite subset has a glb and a lub

  Example: Lattice $\mathbb{Z}$ of integers under "less-than-equal-to" ($\leq$) relation

  - All finite subsets have a glb and a lub
  - Infinite subsets do not have a glb or a lub

Complete Lattice

- Lattice: A partially ordered set such that every non-empty finite subset has a glb and a lub

  Example: Lattice $\mathbb{Z}$ of integers under "less-than-equal-to" ($\leq$) relation

  - All finite subsets have a glb and a lub
  - Infinite subsets do not have a glb or a lub

- Complete Lattice: A lattice in which even $\emptyset$ and infinite subsets have a glb and a lub

Complete Lattice

- Lattice: A partially ordered set such that every non-empty finite subset has a glb and a lub

  Example: Lattice $\mathbb{Z}$ of integers under "less-than-equal-to" ($\leq$) relation

  - All finite subsets have a glb and a lub
  - Infinite subsets do not have a glb or a lub

- Complete Lattice: A lattice in which even $\emptyset$ and infinite subsets have a glb and a lub

  Example: Lattice $\mathbb{Z}$ of integers under $\leq$ relation with ∞ and $-\infty$

Complete Lattice

- Lattice: A partially ordered set such that every non-empty finite subset has a glb and a lub

  Example: Lattice $\mathbb{Z}$ of integers under "less-than-equal-to" ($\leq$) relation

  - All finite subsets have a glb and a lub
  - Infinite subsets do not have a glb or a lub

- Complete Lattice: A lattice in which even $\emptyset$ and infinite subsets have a glb and a lub

  Example: Lattice $\mathbb{Z}$ of integers under $\leq$ relation with ∞ and $-\infty$

  - ∞ is the top element denoted $\top$: $\forall i \in \mathbb{Z},\ i \leq \top$
  - $-\infty$ is the bottom element denoted $\bot$: $\forall i \in \mathbb{Z},\ \bot \leq i$

$\mathbb{Z} \cup \{\infty, -\infty\}$ is a Complete Lattice

- Infinite subsets of $\mathbb{Z} \cup \{\infty, -\infty\}$ have a glb and lub

$\mathbb{Z} \cup \{\infty, -\infty\}$ is a Complete Lattice

- Infinite subsets of $\mathbb{Z} \cup \{\infty, -\infty\}$ have a glb and lub
- What about the empty set?

$\mathbb{Z} \cup \{\infty, -\infty\}$ is a Complete Lattice

- Infinite subsets of $\mathbb{Z} \cup \{\infty, -\infty\}$ have a glb and lub
- What about the empty set?
  - glb($\emptyset$) is $\top$

$\mathbb{Z} \cup \{\infty, -\infty\}$ is a Complete Lattice

- Infinite subsets of $\mathbb{Z} \cup \{\infty, -\infty\}$ have a glb and lub
- What about the empty set?
  - glb($\emptyset$) is $\top$
    Every element of $\mathbb{Z} \cup \{\infty, -\infty\}$ is vacuously a lower bound of an element in $\emptyset$

    OR

    Every element in $\emptyset$ is stronger than every element in $\mathbb{Z} \cup \{\infty, -\infty\}$ (because there is no element in $\emptyset$)

$\mathbb{Z} \cup \{\infty, -\infty\}$ is a Complete Lattice

- Infinite subsets of $\mathbb{Z} \cup \{\infty, -\infty\}$ have a glb and lub
- What about the empty set?
  - glb($\emptyset$) is $\top$
    Every element of $\mathbb{Z} \cup \{\infty, -\infty\}$ is vacuously a lower bound of an element in $\emptyset$

    OR

    Every element in $\emptyset$ is stronger than every element in $\mathbb{Z} \cup \{\infty, -\infty\}$ (because there is no element in $\emptyset$)

    The greatest among these lower bounds is $\top$

$\mathbb{Z} \cup \{\infty, -\infty\}$ is a Complete Lattice

- Infinite subsets of $\mathbb{Z} \cup \{\infty, -\infty\}$ have a glb and lub
- What about the empty set?
  - glb($\emptyset$) is $\top$
    Every element of $\mathbb{Z} \cup \{\infty, -\infty\}$ is vacuously a lower bound of an element in $\emptyset$

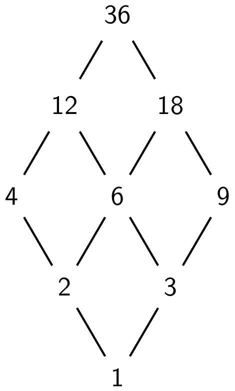
    OR

    Every element in $\emptyset$ is stronger than every element in $\mathbb{Z} \cup \{\infty, -\infty\}$ (because there is no element in $\emptyset$)

    The greatest among these lower bounds is $\top$
  - lub($\emptyset$) is $\bot$

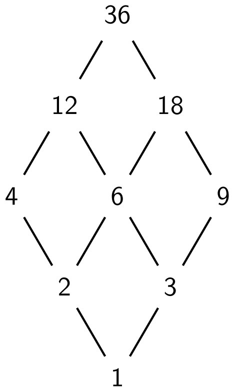
Operations on Lattices

- Meet ($\sqcap$) and Join ($\sqcup$)



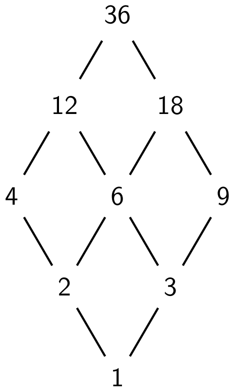
Operations on Lattices

- Meet ($\sqcap$) and Join ($\sqcup$)
  - $x \sqcap y$ computes the glb of x and y
    $z = x \sqcap y \Rightarrow z \sqsubseteq x \wedge z \sqsubseteq y$



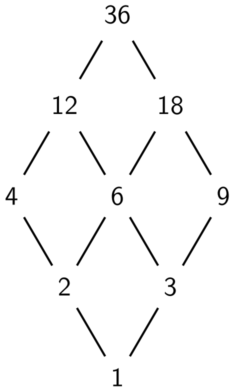
Operations on Lattices

- Meet ($\sqcap$) and Join ($\sqcup$)
  - $x \sqcap y$ computes the glb of x and y
    $z = x \sqcap y \Rightarrow z \sqsubseteq x \wedge z \sqsubseteq y$
  - $x \sqcup y$ computes the lub of x and y
    $z = x \sqcup y \Rightarrow z \sqsupseteq x \wedge z \sqsupseteq y$



Operations on Lattices

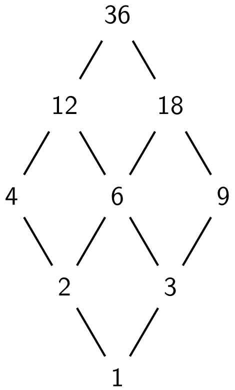
- Meet ($\sqcap$) and Join ($\sqcup$)
  - $x \sqcap y$ computes the glb of x and y
    $z = x \sqcap y \Rightarrow z \sqsubseteq x \wedge z \sqsubseteq y$
  - $x \sqcup y$ computes the lub of x and y
    $z = x \sqcup y \Rightarrow z \sqsupseteq x \wedge z \sqsupseteq y$
  - $\sqcap$ and $\sqcup$ are commutative, associative, and idempotent



Operations on Lattices

- Meet ($\sqcap$) and Join ($\sqcup$)
  - $x \sqcap y$ computes the glb of x and y
    $z = x \sqcap y \Rightarrow z \sqsubseteq x \wedge z \sqsubseteq y$
  - $x \sqcup y$ computes the lub of x and y
    $z = x \sqcup y \Rightarrow z \sqsupseteq x \wedge z \sqsupseteq y$
  - $\sqcap$ and $\sqcup$ are commutative, associative, and idempotent
- Top ($\top$) and Bottom ($\perp$) elements

$$
\begin{array}{lcl}
\forall x \in L,\ x \sqcap \top & = & x \\
\forall x \in L,\ x \sqcup \top & = & \top \\
\forall x \in L,\ x \sqcap \perp & = & \perp \\
\forall x \in L,\ x \sqcup \perp & = & x
\end{array}
$$

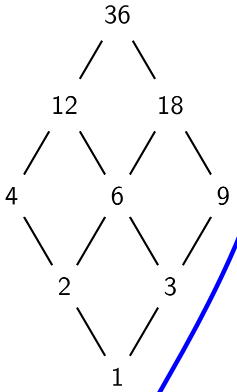


Operations on Lattices

- Meet ($\sqcap$) and Join ($\sqcup$)
  - $x \sqcap y$ computes the glb of x and y
    $z = x \sqcap y \Rightarrow z \sqsubseteq x \wedge z \sqsubseteq y$
  - $x \sqcup y$ computes the lub of x and y
    $z = x \sqcup y \Rightarrow z \sqsupseteq x \wedge z \sqsupseteq y$
  - $\sqcap$ and $\sqcup$ are commutative, associative, and idempotent
- Top ($\top$) and Bottom ($\bot$) elements

$$
\begin{aligned}
\forall x \in L,\ x \sqcap \top &= x \\
\forall x \in L,\ x \sqcup \top &= \top \\
\forall x \in L,\ x \sqcap \bot &= \bot \\
\forall x \in L,\ x \sqcup \bot &= x
\end{aligned}
$$

Greatest common divisor



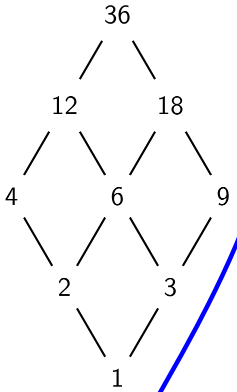
$x \sqcap y = gcd(x, y)$

Operations on Lattices

- Meet ($\sqcap$) and Join ($\sqcup$)
  - $x \sqcap y$ computes the glb of x and y
    $z = x \sqcap y \Rightarrow z \sqsubseteq x \wedge z \sqsubseteq y$
  - $x \sqcup y$ computes the lub of x and y
    $z = x \sqcup y \Rightarrow z \sqsupseteq x \wedge z \sqsupseteq y$
  - $\sqcap$ and $\sqcup$ are commutative, associative, and idempotent
- Top ($\top$) and Bottom ($\bot$) elements

$$
\begin{array}{lcl}
\forall x \in L,\ x \sqcap \top & = & x \\
\forall x \in L,\ x \sqcup \top & = & \top \\
\forall x \in L,\ x \sqcap \bot & = & \bot \\
\forall x \in L,\ x \sqcup \bot & = & x
\end{array}
$$

Greatest common divisor



$x \sqcap y = gcd(x, y)$

$x \sqcup y = lcm(x, y)$

Lowest common multiple

Partial Order and Operations

- For a lattice $\sqsubseteq$ induces $\sqcap$ and $\sqcup$ and vice-versa
- The choices of $\sqsubseteq$, $\sqcap$, and $\sqcup$ cannot be arbitrary

  They have to be

  - consistent with each other, and
  - definable in terms of each other
- For some variants of lattices, $\sqcap$ or $\sqcup$ may not exist

  Yet the requirement of its consistency with $\sqsubseteq$ cannot be violated

Finite Lattices are Complete

- Any given set of elements has a glb and a lub

Available Expressions Analysis

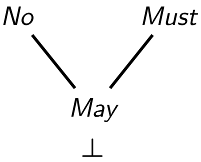
$$
\begin{tikzcd}
 & (\top) & \\
 & \{e_1, e_2, e_3\} \arrow[dl] \arrow[d] \arrow[dr] & \\
\{e_1, e_2\} \arrow[d] \arrow[dr] & \{e_1, e_3\} \arrow[dl] \arrow[dr] & \{e_2, e_3\} \arrow[dl] \arrow[d] \\
\{e_1\} \arrow[dr] & \{e_2\} \arrow[d] & \{e_3\} \arrow[dl] \\
 & \emptyset & \\
 & (\bot) &
\end{tikzcd}
$$

Partially Available Expressions Analysis

$$
\begin{tikzcd}
 & (\top) & \\
 & \emptyset \arrow[dl] \arrow[d] \arrow[dr] & \\
\{e_1\} \arrow[d] \arrow[dr] & \{e_2\} \arrow[dl] \arrow[dr] & \{e_3\} \arrow[dl] \arrow[d] \\
\{e_1, e_2\} \arrow[dr] & \{e_1, e_3\} \arrow[d] & \{e_2, e_3\} \arrow[dl] \\
 & \{e_1, e_2, e_3\} & \\
 & (\bot) &
\end{tikzcd}
$$

Lattice for May-Must Analysis

- There is no $\top$ among the natural values



Interpreting data flow values

- *No.* Information does not hold along any path
- *Must.* Information must hold along all paths
- *May.* Information may hold along some path

- An artificial $\top$ can be added

Some Variants of Lattices

A poset L is

- A lattice iff each non-empty finite subset of L has a glb and lub
- A complete lattice iff each subset of L has a glb and lub
- A meet semilattice iff each non-empty finite subset of L has a glb
- A join semilattice iff each non-empty finite subset of L has a lub
- A bounded lattice iff L is a lattice and has $\top$ and $\perp$ elements

A Bounded Lattice Need Not be Complete (1)

- Let A be all finite subsets of $\mathbb{Z}$
  Then, A is an infinite set
- The poset $L = (A \cup \{\mathbb{Z}\}, \subseteq)$ is a bounded lattice with $\top = \mathbb{Z}$ and $\bot = \emptyset$
  The join $\sqcup$ of this lattice is $\cup$
- To see why, consider a set $S \subseteq L$ containing *all* subsets of $\mathbb{Z}$ that do not contain the number 1

A Bounded Lattice Need Not be Complete (1)

- Let A be all finite subsets of $\mathbb{Z}$
  Then, A is an infinite set
- The poset $L = (A \cup \{\mathbb{Z}\}, \subseteq)$ is a bounded lattice with $\top = \mathbb{Z}$ and $\bot = \emptyset$
  The join $\sqcup$ of this lattice is $\cup$
- To see why, consider a set $S \subseteq L$ containing *all* subsets of $\mathbb{Z}$ that do not contain the number 1

  S contains *all* finite sets that do not contain 1
  - Since the number of such sets is infinite, their union is an infinite set
  - $\mathbb{Z} - \{1\}$ is not contained in L (the only infinite set in L is $\mathbb{Z}$)
  - S does not have a lub in L

A Bounded Lattice Need Not be Complete (1)

- Let A be all finite subsets of $\mathbb{Z}$

  Then, A is an infinite set

- The poset $L = (A \cup \{\mathbb{Z}\}, \subseteq)$ is a bounded lattice with $\top = \mathbb{Z}$ and $\bot = \emptyset$

  The join $\sqcup$ of this lattice is $\cup$

- To see why, consider a set $S \subseteq L$ containing *all* subsets of $\mathbb{Z}$ that do not contain the number 1

  S contains *all* finite sets that do not contain 1

  - Since the number of such sets is infinite, their union is an infinite set
  - $\mathbb{Z} - \{1\}$ is not contained in L (the only infinite set in L is $\mathbb{Z}$)
  - S does not have a lub in L

  Hence L is not complete

A Bounded Lattice Need Not be Complete (1)

- Let A be all finite subsets of $\mathbb{Z}$

  Then, A is an infinite set

- The poset $L = (A \cup \{\mathbb{Z}\}, \subseteq)$ is a bounded lattice with $\top = \mathbb{Z}$ and $\bot = \emptyset$

- It may be tempting to assume that $\mathbb{Z}$ is the lub of S because it is an upper bound of S and no other upper bound of S in the lattice is weaker $\mathbb{Z}$

- However, the join operation $\cup$ of L does not compute $\mathbb{Z}$ as the lub of S (because it must exclude 1)

- The join operation $\cup$ is inconsistent with the partial order $\supseteq$ of L. Hence we say that join does not exist for S

A Bounded Lattice Need Not be Complete (2)

- A bounded lattice L has a glb and lub of L in L
- A complete lattice L should have glb and lub of *all* subsets of L
- A lattice L should have glb and lub of *all* finite non-empty subsets of L

Ascending and Descending Chains

- Strictly ascending chain $x \sqsubset y \sqsubset \cdots \sqsubset z$
- Strictly descending chain $x \sqsupset y \sqsupset \cdots \sqsupset z$
- DCC: Descending Chain Condition
  All strictly descending chains are finite
- ACC: Ascending Chain Condition
  All strictly ascending chains are finite

Complete Lattice and Ascending and Descending Chains

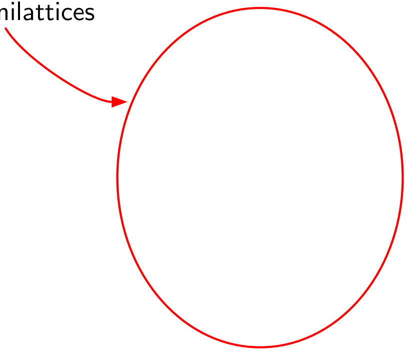
- If L satisfies acc and dcc, then
  - L has finite height, and
  - L is complete
- A complete lattice need not have finite height (i.e. strict chains may not be finite)

  Example:

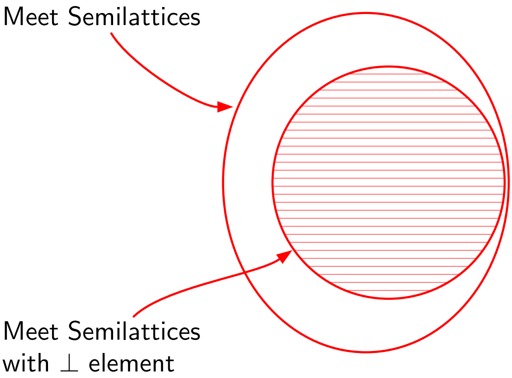
  Lattice of integers under $\leq$ relation with ∞ as $\top$ and $-\infty$ as $\perp$

Variants of Lattices

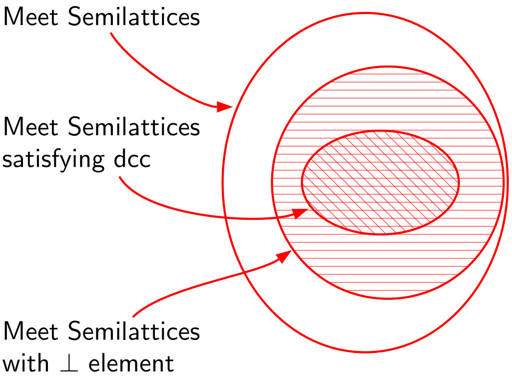
Meet Semilattices



Variants of Lattices

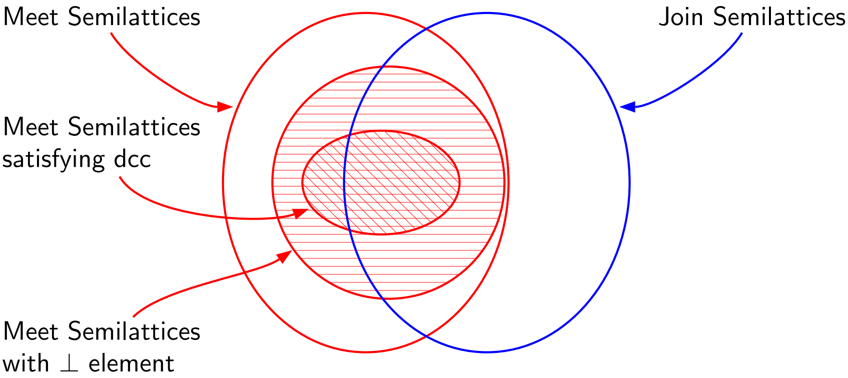


Variants of Lattices



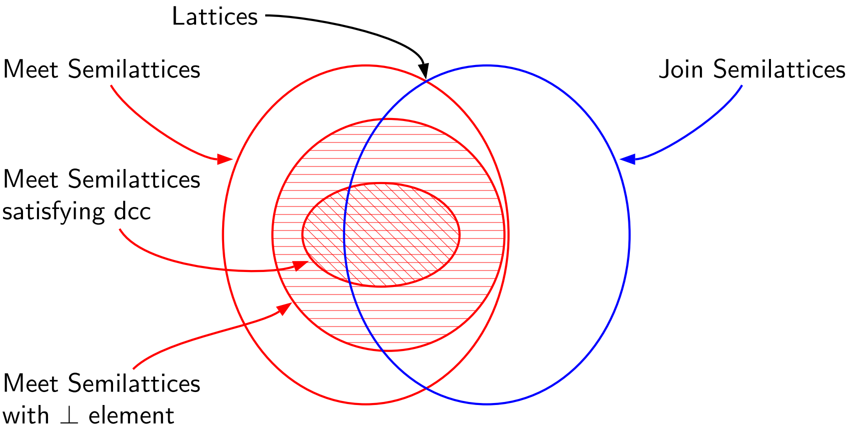
- dcc: descending chain condition

Variants of Lattices



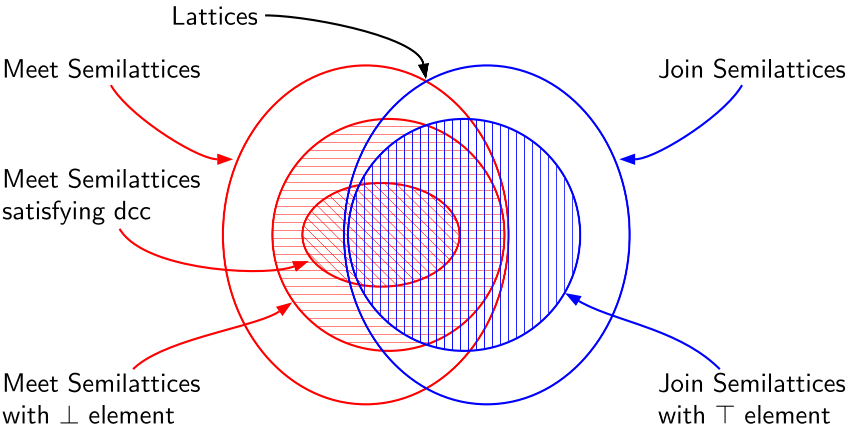
- dcc: descending chain condition

Variants of Lattices



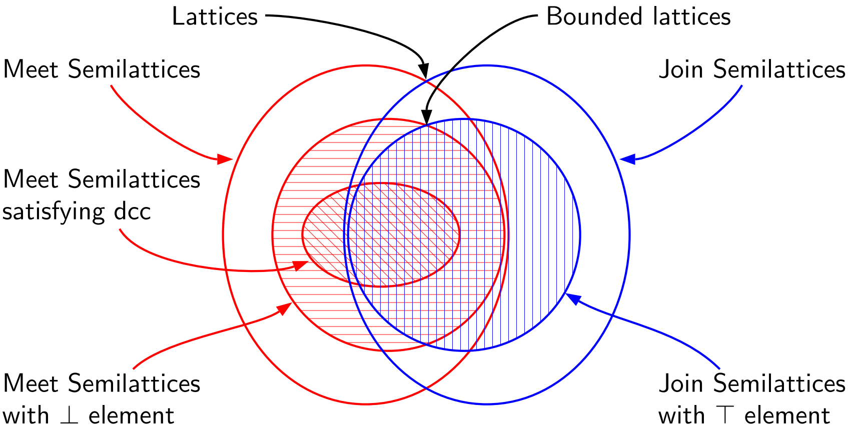
- dcc: descending chain condition

Variants of Lattices



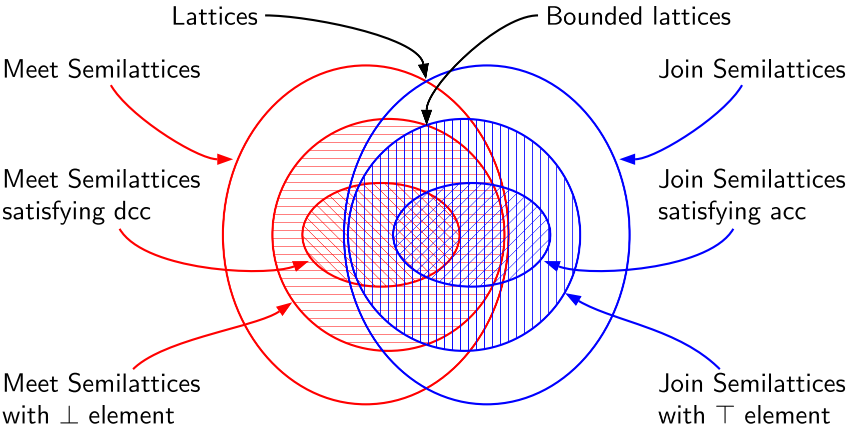
- dcc: descending chain condition

Variants of Lattices



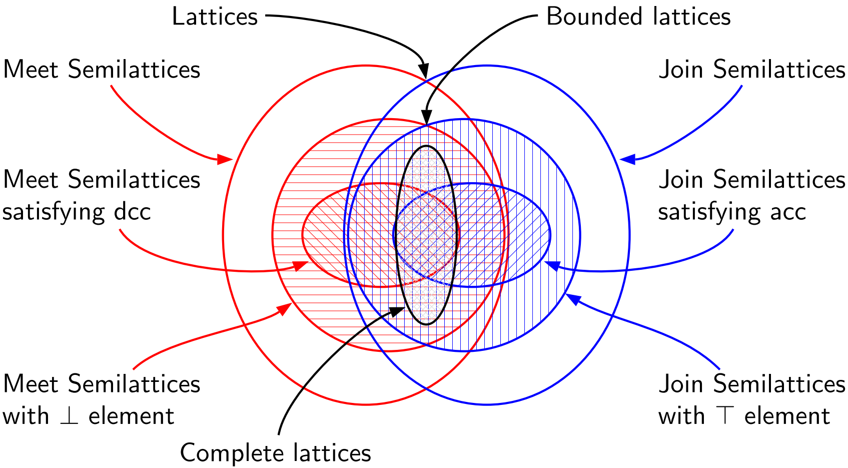
- dcc: descending chain condition

Variants of Lattices



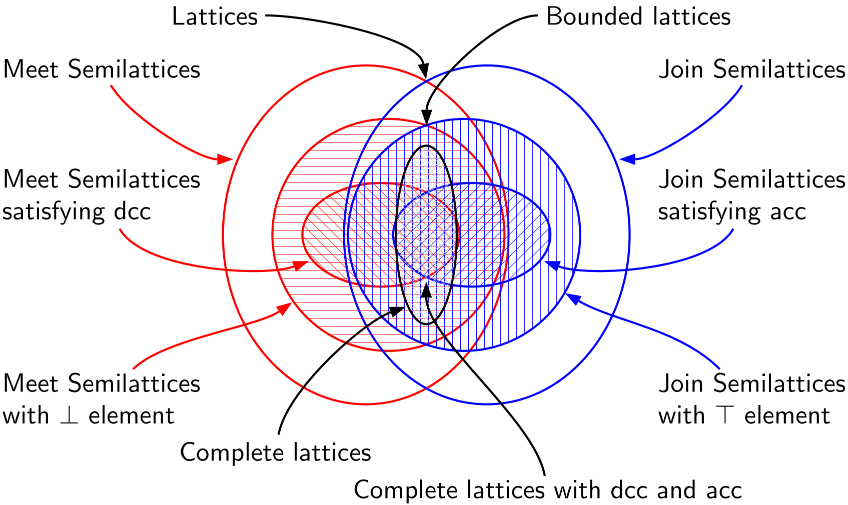
- dcc: descending chain condition
- acc: ascending chain condition

Variants of Lattices



- dcc: descending chain condition
- acc: ascending chain condition

Variants of Lattices



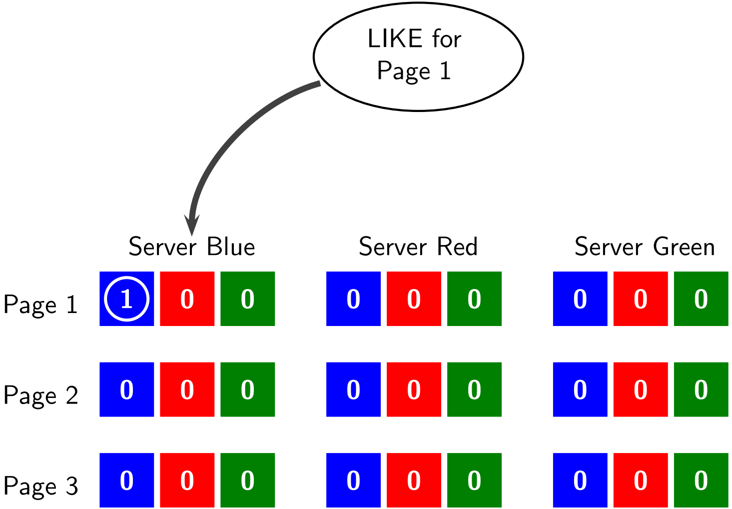
- dcc: descending chain condition
- acc: ascending chain condition

An Example of Lattices: Maintaining LIKE Counts on Cloud

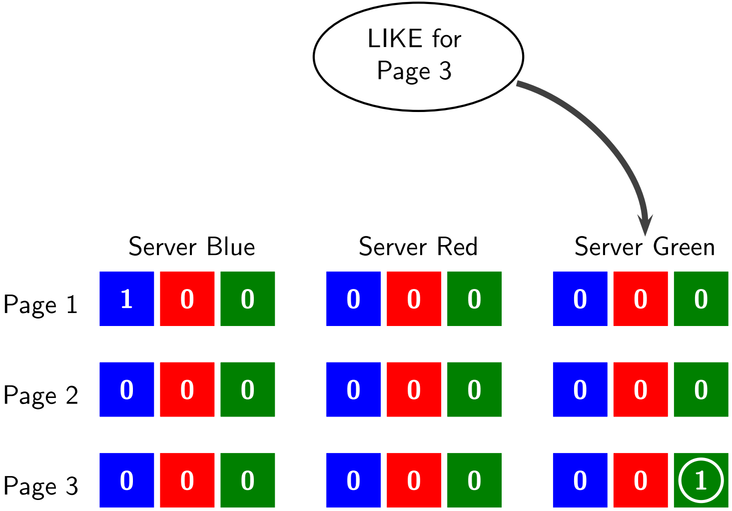
Maintain *n* servers and divide the traffic
– Each server maintains an *n*-tuple for each page
– Updates the counters for its own slot

	Server Blue			Server Red			Server Green		
Page 1	0	0	0	0	0	0	0	0	0
Page 2	0	0	0	0	0	0	0	0	0
Page 3	0	0	0	0	0	0	0	0	0

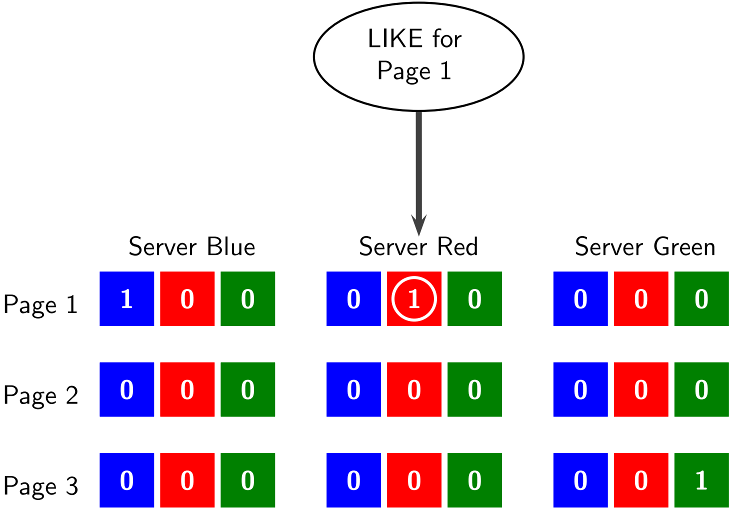
An Example of Lattices: Maintaining LIKE Counts on Cloud



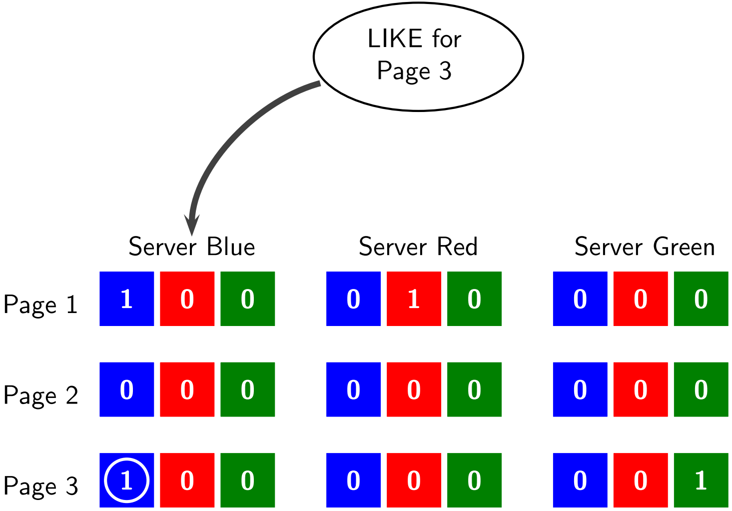
An Example of Lattices: Maintaining LIKE Counts on Cloud



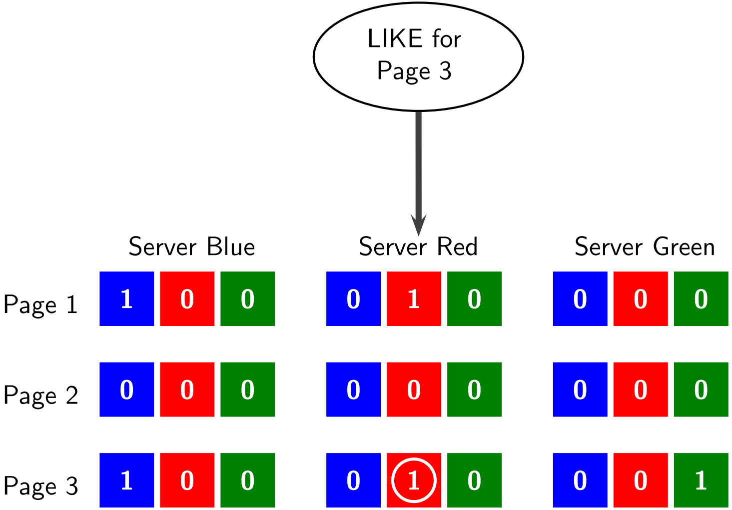
An Example of Lattices: Maintaining LIKE Counts on Cloud



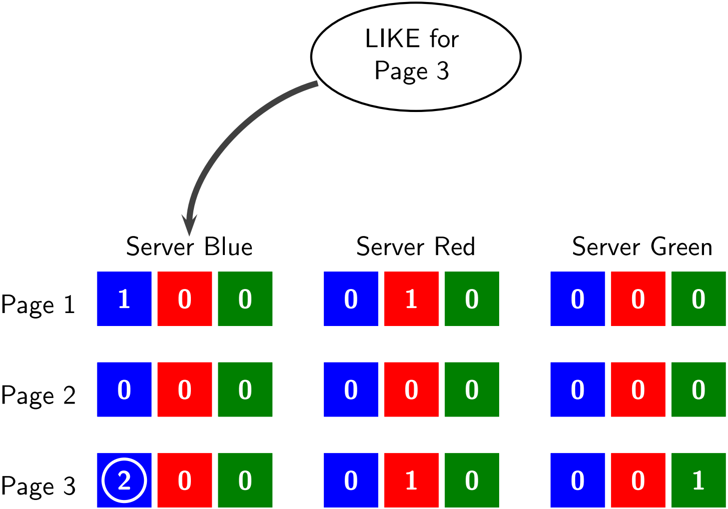
An Example of Lattices: Maintaining LIKE Counts on Cloud



An Example of Lattices: Maintaining LIKE Counts on Cloud



An Example of Lattices: Maintaining LIKE Counts on Cloud



An Example of Lattices: Maintaining LIKE Counts on Cloud

Synchronize:
– Send the data to other servers
– Update the counters using point-wise max

	Server Blue			Server Red			Server Green		
Page 1	1	0	0	0	1	0	0	0	0
Page 2	0	0	0	0	0	0	0	0	0
Page 3	2	0	0	0	1	0	0	0	1

An Example of Lattices: Maintaining LIKE Counts on Cloud

Synchronize:
– Send the data to other servers
– Update the counters using point-wise max

- Lattice of n-tuples using point-wise $\geq$ as the partial order

$$\langle x_1, x_2, \ldots, x_n \rangle \sqsubseteq \langle y_1, y_2, \ldots, y_n \rangle = (x_1 \geq y_1) \wedge (x_2 \geq y_2) \ldots \wedge (x_n \geq y_n)$$

- Tuples merged with max operation

$$\langle x_1, x_2, \ldots, x_n \rangle \sqcap \langle y_1, y_2, \ldots, y_n \rangle = \langle \max(x_1, y_1), \max(x_2, y_2), \ldots, \max(x_n, y_n) \rangle$$

Page 3

An Example of Lattices: Maintaining LIKE Counts on Cloud

Synchronize:
— Send the data to other servers
— Update the counters using point-wise max

	Server Blue			Server Red			Server Green		
Page 1	1	0	0	0	1	0	0	0	0
Page 2	0	0	0	0	0	0	0	0	0
Page 3	2	0	0	0	1	0	0	0	1

An Example of Lattices: Maintaining LIKE Counts on Cloud

Synchronize:
— Send the data to other servers
— Update the counters using point-wise max

	Server Blue			Server Red			Server Green		
Page 1	1	0	0	1	1	0	0	0	0
Page 2	0	0	0	0	0	0	0	0	0
Page 3	2	0	0	2	1	0	0	0	1

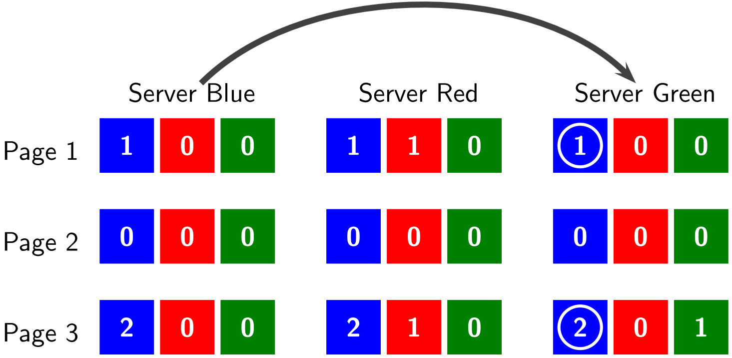
An Example of Lattices: Maintaining LIKE Counts on Cloud

Synchronize:
– Send the data to other servers
– Update the counters using point-wise max

	Server Blue			Server Red			Server Green		
Page 1	1	0	0	1	1	0	0	0	0
Page 2	0	0	0	0	0	0	0	0	0
Page 3	2	0	0	2	1	0	0	0	1

An Example of Lattices: Maintaining LIKE Counts on Cloud

Synchronize:
– Send the data to other servers
– Update the counters using point-wise max



An Example of Lattices: Maintaining LIKE Counts on Cloud

Synchronize:
- Send the data to other servers
- Update the counters using point-wise max

	Server Blue			Server Red			Server Green		
Page 1	1	0	0	1	1	0	1	0	0
Page 2	0	0	0	0	0	0	0	0	0
Page 3	2	0	0	2	1	0	2	0	1

An Example of Lattices: Maintaining LIKE Counts on Cloud

Synchronize:
— Send the data to other servers
— Update the counters using point-wise max

	Server Blue			Server Red			Server Green		
Page 1	1	1	0	1	1	0	1	0	0
Page 2	0	0	0	0	0	0	0	0	0
Page 3	2	1	0	2	1	0	2	0	1

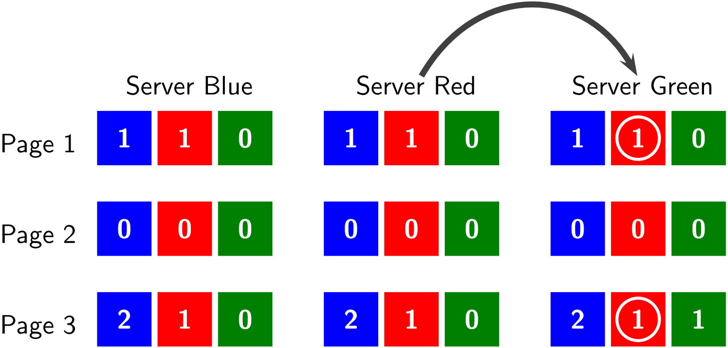
An Example of Lattices: Maintaining LIKE Counts on Cloud

Synchronize:
– Send the data to other servers
– Update the counters using point-wise max

	Server Blue			Server Red			Server Green		
Page 1	1	1	0	1	1	0	1	0	0
Page 2	0	0	0	0	0	0	0	0	0
Page 3	2	1	0	2	1	0	2	0	1

An Example of Lattices: Maintaining LIKE Counts on Cloud

Synchronize:
- Send the data to other servers
- Update the counters using point-wise max



An Example of Lattices: Maintaining LIKE Counts on Cloud

Synchronize:
— Send the data to other servers
— Update the counters using point-wise max

	Server Blue			Server Red			Server Green		
Page 1	1	1	0	1	1	0	1	1	0
Page 2	0	0	0	0	0	0	0	0	0
Page 3	2	1	0	2	1	0	2	1	1

An Example of Lattices: Maintaining LIKE Counts on Cloud

Synchronize:
– Send the data to other servers
– Update the counters using point-wise max

	Server Blue			Server Red			Server Green		
Page 1	1	1	0	1	1	0	1	1	0
Page 2	0	0	0	0	0	0	0	0	0
Page 3	2	1	1	2	1	0	2	1	1

An Example of Lattices: Maintaining LIKE Counts on Cloud

Synchronize:
– Send the data to other servers
– Update the counters using point-wise max

	Server Blue			Server Red			Server Green		
Page 1	1	1	0	1	1	0	1	1	0
Page 2	0	0	0	0	0	0	0	0	0
Page 3	2	1	1	2	1	0	2	1	1

An Example of Lattices: Maintaining LIKE Counts on Cloud

Synchronize:
— Send the data to other servers
— Update the counters using point-wise max

	Server Blue			Server Red			Server Green		
Page 1	1	1	0	1	1	0	1	1	0
Page 2	0	0	0	0	0	0	0	0	0
Page 3	2	1	1	2	1	1	2	1	1

An Example of Lattices: Maintaining LIKE Counts on Cloud

After synchronization, all servers have the same data
Count for a page:
– Take sum of all counts at any server for the page

	Server Blue			Server Red			Server Green		
Page 1	1	1	0	1	1	0	1	1	0
Page 2	0	0	0	0	0	0	0	0	0
Page 3	2	1	1	2	1	1	2	1	1

Constructing Lattices

- Powerset construction with subset or superset relation
- Products of lattices
  - Cartesian product
  - Interval product
- Set of mappings as lattices

Variants of Powerset Lattices

Let the underlying set be S

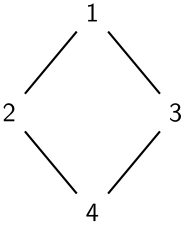
- The set 2^S with the partial order $\subseteq$ is a lattice

$$
\begin{aligned}
x \sqsubseteq y &\Leftrightarrow x \subseteq y \\
x \sqcap y &= x \cap y \\
x \sqcup y &= x \cup y
\end{aligned}
$$

- The set 2^S with the partial order $\supseteq$ is a lattice

$$
\begin{aligned}
x \sqsubseteq y &\Leftrightarrow x \supseteq y \\
x \sqcap y &= x \cup y \\
x \sqcup y &= x \cap y
\end{aligned}
$$

Cartesian Product of Lattice



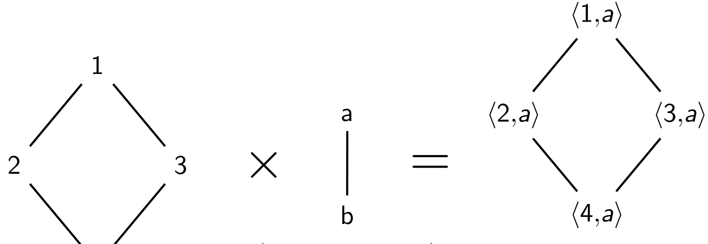
$\times$

$=$

$\langle L_A, \sqsubseteq_A, \sqcap_A, \sqcup_A \rangle$

$\langle L_N, \sqsubseteq_N, \sqcap_N, \sqcup_N \rangle$

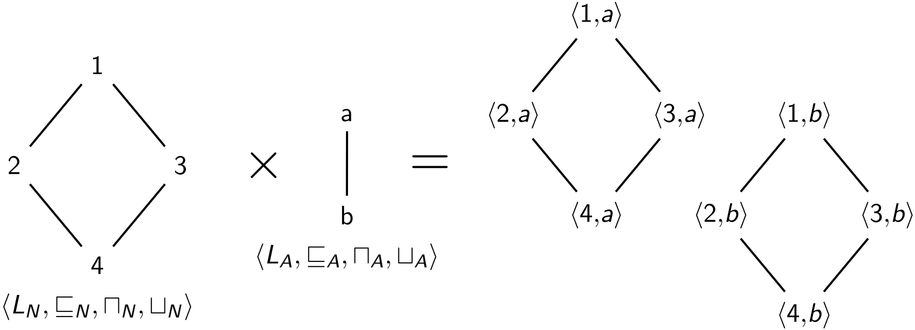
Cartesian Product of Lattice



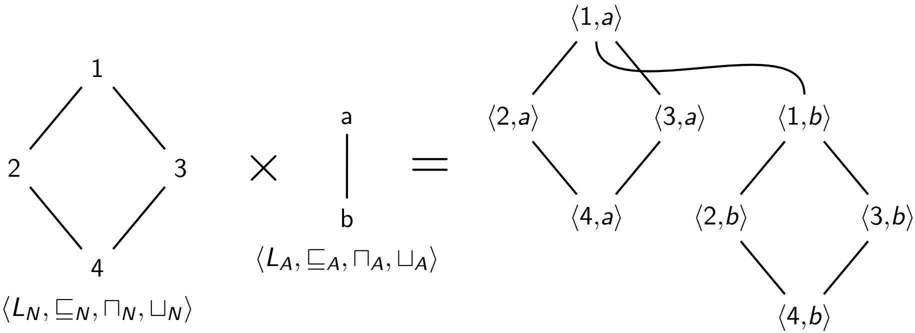
$\langle L_A, \sqsubseteq_A, \sqcap_A, \sqcup_A \rangle$

$\langle L_N, \sqsubseteq_N, \sqcap_N, \sqcup_N \rangle$

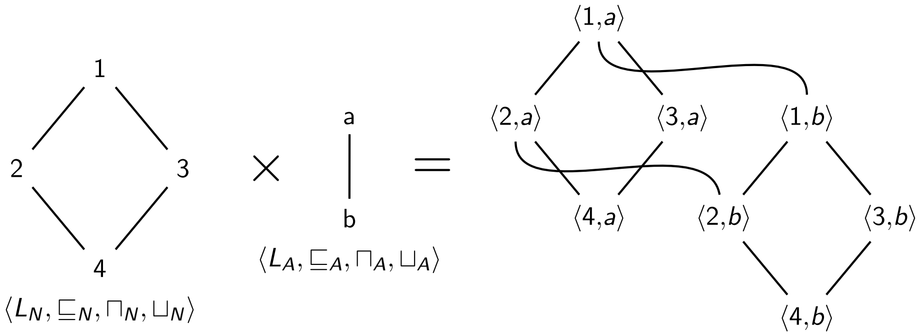
Cartesian Product of Lattice



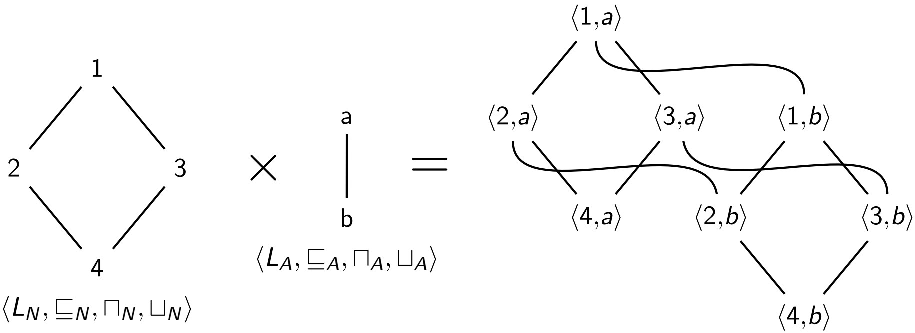
Cartesian Product of Lattice



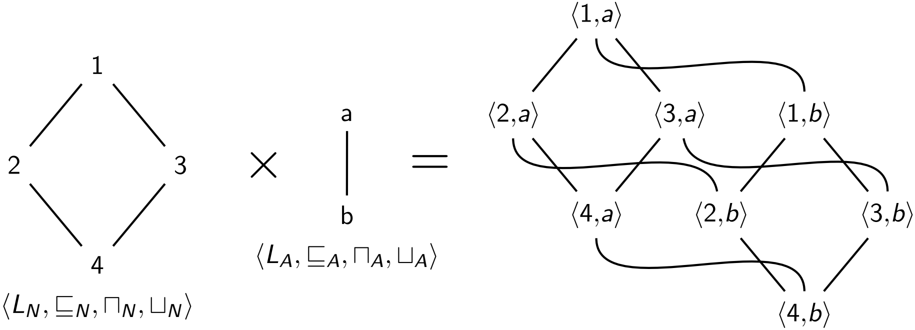
Cartesian Product of Lattice



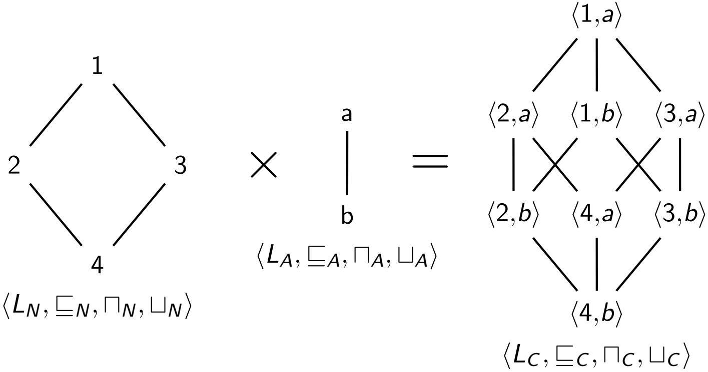
Cartesian Product of Lattice



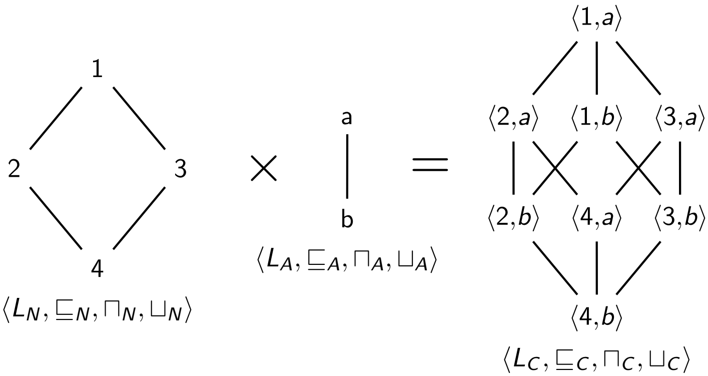
Cartesian Product of Lattice



Cartesian Product of Lattice



Cartesian Product of Lattice



$$
\begin{aligned}
\langle x_1, y_1 \rangle \sqsubseteq_C \langle x_2, y_2 \rangle \quad &\Leftrightarrow \quad x_1 \sqsubseteq_N x_2 \wedge y_1 \sqsubseteq_A y_2 \\
\langle x_1, y_1 \rangle \sqcap_C \langle x_2, y_2 \rangle \quad &= \quad \langle x_1 \sqcap_N x_2, y_1 \sqcap_A y_2 \rangle \\
\langle x_1, y_1 \rangle \sqcup_C \langle x_2, y_2 \rangle \quad &= \quad \langle x_1 \sqcup_N x_2, y_1 \sqcup_A y_2 \rangle
\end{aligned}
$$

Example of Cartesian Product: Concept Lattices

- *Context* of concepts. A collection of objects and their attributes
- *Concepts*. Sets of attributes as exhibited by specific objects
  - A concept C is a pair (O, A) where
    O is a set of objects exhibiting attributes in the set A
  - Every object in O has every attribute in A
- Partial order. $(O_2, A_2) \sqsubseteq (O_1, A_1) \Leftrightarrow O_2 \subseteq O_1$
  - Very few objects have all attributes
  - Since A is the set of attributes common to all objects in O,

$$O_2 \subseteq O_1 \Rightarrow A_2 \supseteq A_1$$

As the number of chosen objects decreases, the number of common attributes increases

Example of Concept Lattice (1)

From *Introduction to Lattices and Order* by Davey and Priestley [2002]

		Size			Distance from Sun		Moon?	
		Small (ss)	Medium (sm)	Large (sl)	Near (dn)	Far (df)	Yes (my)	No (mn)
Mercury	Me	x			x			x
Venus	V	x			x			x
Earth	E	x			x		x	
Mars	Ma	x			x		x	
Jupiter	J			x		x	x	
Saturn	S			x		x	x	
Uranus	U		x			x	x	
Neptune	N		x			x	x	
Pluto	P	x				x	x	

Example of Concept Lattice (2)

We write (O, A) as $\dfrac{O}{A}$

$$\frac{\{Me, V, E, Ma, J, S, U, N, P\}}{\{\}}$$

Example of Concept Lattice (2)

We write (O, A) as $\dfrac{O}{A}$

$$\frac{\{Me, V, E, Ma, J, S, U, N, P\}}{\{\}}$$

$$\frac{\{Me, V, E, Ma, P\}}{\{ss\}} \qquad \frac{\{E, Ma, J, S, U, N, P\}}{\{my\}}$$

Example of Concept Lattice (2)

We write (O, A) as $\dfrac{O}{A}$

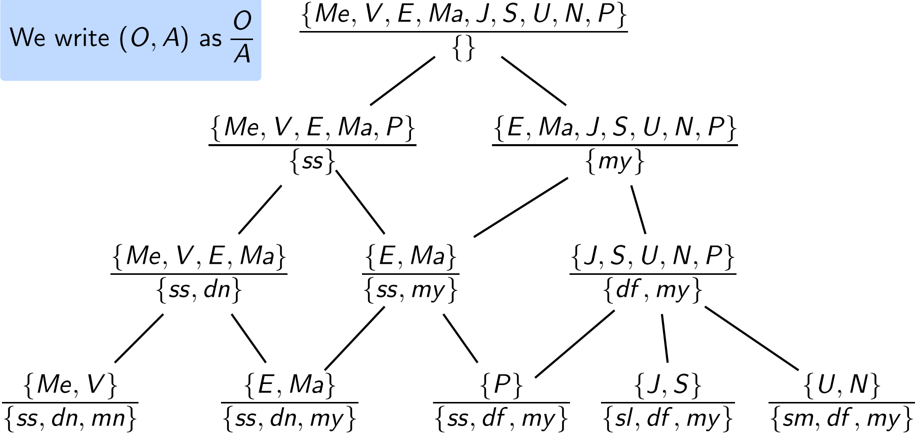
$$\frac{\{Me, V, E, Ma, J, S, U, N, P\}}{\{\}}$$

$$\frac{\{Me, V, E, Ma, P\}}{\{ss\}} \qquad \frac{\{E, Ma, J, S, U, N, P\}}{\{my\}}$$

$$\frac{\{Me, V, E, Ma\}}{\{ss, dn\}} \qquad \frac{\{E, Ma\}}{\{ss, my\}} \qquad \frac{\{J, S, U, N, P\}}{\{df, my\}}$$

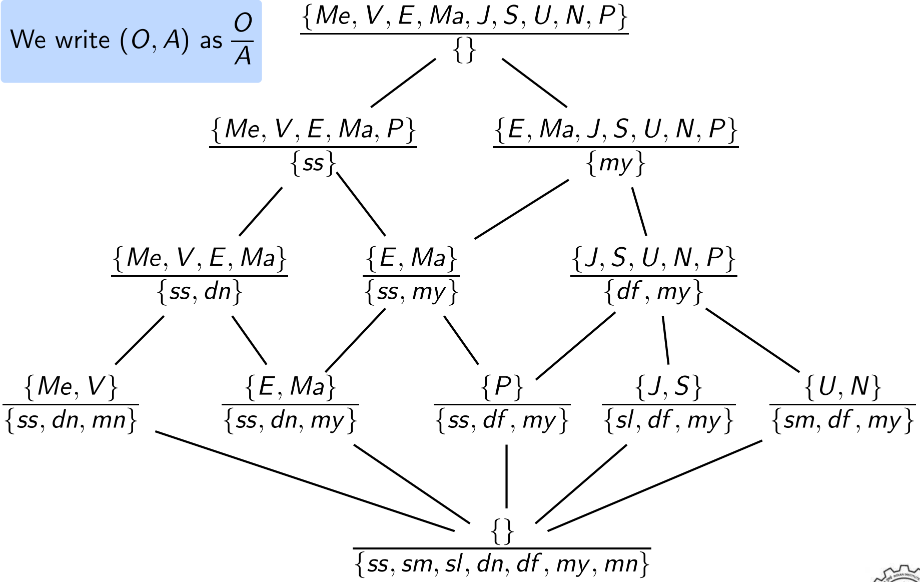
Example of Concept Lattice (2)

We write (O, A) as $\dfrac{O}{A}$



Example of Concept Lattice (2)

We write (O, A) as $\dfrac{O}{A}$



Variants of Product Lattices (1)

$L \subseteq L_1 \times L_2$, $\{(x_1, x_2), (y_1, y_2)\} \subseteq L$, $\{x_1, y_1\} \subseteq L_1$, and $\{x_2, y_2\} \subseteq L_2$

- *Cartesian Product*

$$
\begin{aligned}
(x_1, x_2) \sqsubseteq (y_1, y_2) &\Leftrightarrow x_1 \sqsubseteq_1 y_1 \wedge x_2 \sqsubseteq_2 y_2 \\
(x_1, x_2) \sqcap (y_1, y_2) &= x_1 \sqcap_1 y_1 \wedge x_2 \sqcap_2 y_2 \\
(x_1, x_2) \sqcup (y_1, y_2) &= x_1 \sqcup_1 y_1 \wedge x_2 \sqcup_2 y_2
\end{aligned}
$$

- *Interval Product*

Variants of Product Lattices (1)

$L \subseteq L_1 \times L_2$, $\{(x_1, x_2), (y_1, y_2)\} \subseteq L$, $\{x_1, y_1\} \subseteq L_1$, and $\{x_2, y_2\} \subseteq L_2$

- *Cartesian Product*

$$\begin{aligned}(x_1, x_2) \sqsubseteq (y_1, y_2) &\Leftrightarrow x_1 \sqsubseteq_1 y_1 \wedge x_2 \sqsubseteq_2 y_2 \\ (x_1, x_2) \sqcap (y_1, y_2) &= x_1 \sqcap_1 y_1 \wedge x_2 \sqcap_2 y_2 \\ (x_1, x_2) \sqcup (y_1, y_2) &= x_1 \sqcup_1 y_1 \wedge x_2 \sqcup_2 y_2\end{aligned}$$

- *Interval Product*

$$\begin{aligned}(x_1, x_2) \sqsubseteq (y_1, y_2) &\Leftrightarrow x_1 \sqsubseteq_1 y_1 \wedge x_2 \sqsupseteq_2 y_2 \\ (x_1, x_2) \sqcap (y_1, y_2) &= x_1 \sqcap_1 y_1 \wedge x_2 \sqcup_2 y_2 \\ (x_1, x_2) \sqcup (y_1, y_2) &= x_1 \sqcup_1 y_1 \wedge x_2 \sqcap_2 y_2\end{aligned}$$

Example: Integer lattices with $\sqsubseteq$ as $\leq$ and $\sqsupseteq$ as $\geq$

$(2, 10) \sqcap (5, 50) = (2, 50)$ and $(2, 10) \sqcup (5, 50) = (5, 10)$

- $\sqcap$ computes the *smallest* interval *containing* both the intervals
- $\sqcup$ computes the *largest* interval *contained* in both the intervals

Set of Mappings as a Lattice

Given a set A and a lattice L_1, the set of mappings $L = A \rightarrow L_1$ is a lattice

Let $X, Y \in L$, $\quad a \in A$, and $x, y \in L_1$

$$\begin{aligned} X \sqsubseteq Y &\Leftrightarrow \forall a \in S : (a, x) \in X \wedge (a, y) \in Y \wedge x \sqsubseteq_1 y \\ X \sqcap Y &= \{(a, x \sqcap_1 y) \mid a \in S, (a, x) \in X, (a, y) \in Y\} \\ X \sqcup Y &= \{(a, x \sqcup_1 y) \mid a \in S, (a, x) \in X, (a, y) \in Y\} \end{aligned}$$

Part 5

Data Flow Values: Details

The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- Requirement: glb must exist for all non-empty finite subsets

The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- Requirement: glb must exist for all non-empty finite subsets
- Corollary: $\perp$ must exist

  What guarantees the presence of $\perp$?

The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- Requirement: glb must exist for all non-empty finite subsets
- Corollary: $\perp$ must exist

  What guarantees the presence of $\perp$?

- $\top$ may not exist. Can be added artificially

The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- Requirement: glb must exist for all non-empty finite subsets
- Corollary: $\perp$ must exist

  What guarantees the presence of $\perp$?

  - Assume that two maximal descending chains terminate at two incomparable elements x_1 and x_2

- $\top$ may not exist. Can be added artificially

The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- Requirement: glb must exist for all non-empty finite subsets
- Corollary: $\perp$ must exist

  What guarantees the presence of $\perp$?

  - Assume that two maximal descending chains terminate at two incomparable elements x_1 and x_2
  - Since this is a meet semilattice, glb of $\{x_1, x_2\}$ must exist (say z)

- $\top$ may not exist. Can be added artificially

The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- Requirement: glb must exist for all non-empty finite subsets
- Corollary: $\perp$ must exist

  What guarantees the presence of $\perp$?

  - Assume that two maximal descending chains terminate at two incomparable elements x_1 and x_2
  - Since this is a meet semilattice, glb of $\{x_1, x_2\}$ must exist (say z)

    $\Rightarrow$ Neither of the chains is maximal

    Both of them can be extended to include z

- $\top$ may not exist. Can be added artificially

The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- Requirement: glb must exist for all non-empty finite subsets
- Corollary: $\bot$ must exist

  What guarantees the presence of $\bot$?

  - Assume that two maximal descending chains terminate at two incomparable elements x_1 and x_2
  - Since this is a meet semilattice, glb of $\{x_1, x_2\}$ must exist (say z)
    $\Rightarrow$ Neither of the chains is maximal
    Both of them can be extended to include z
  - Extending this argument to all strictly descending chains, it is easy to see that $\bot$ must exist

- $\top$ may not exist. Can be added artificially

The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- Requirement: glb must exist for all non-empty finite subsets
- Corollary: $\perp$ must exist

  What guarantees the presence of $\perp$?

  - Assume that two maximal descending chains terminate at two incomparable elements x_1 and x_2
  - Since this is a meet semilattice, glb of $\{x_1, x_2\}$ must exist (say z)
    $\Rightarrow$ Neither of the chains is maximal
    Both of them can be extended to include z
  - Extending this argument to all strictly descending chains, it is easy to see that $\perp$ must exist
- $\top$ may not exist. Can be added artificially
  - lub of arbitrary elements may not exist

The Set of Data Flow Values For Available Expressions Analysis

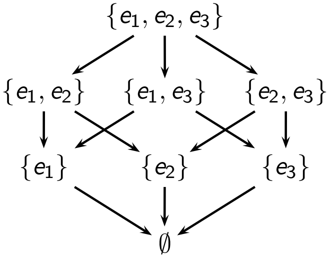
- The powerset of the universal set of expressions
- Partial order is the subset relation

$$
\begin{tikzcd}
 & \{e_1, e_2, e_3\} \arrow[dl] \arrow[d] \arrow[dr] & \\
\{e_1, e_2\} \arrow[d] \arrow[dr] & \{e_1, e_3\} \arrow[dl] \arrow[dr] & \{e_2, e_3\} \arrow[dl] \arrow[d] \\
\{e_1\} \arrow[dr] & \{e_2\} \arrow[d] & \{e_3\} \arrow[dl] \\
 & \emptyset &
\end{tikzcd}
$$

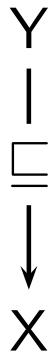
Set View of the Lattice

The Set of Data Flow Values For Available Expressions Analysis

- The powerset of the universal set of expressions
- Partial order is the subset relation

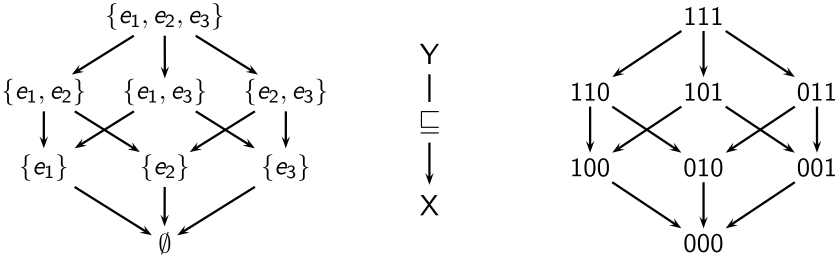


Set View of the Lattice



The Set of Data Flow Values For Available Expressions Analysis

- The powerset of the universal set of expressions
- Partial order is the subset relation



Set View of the Lattice

Bit Vector View

The Concept of Approximation

- x approximates y *iff*

  x can be used in place of y without causing any problems

- Validity of approximation is context specific

  x may be approximated by y in one context and by z in another

The Concept of Approximation

- x approximates y *iff*

  x can be used in place of y without causing any problems

- Validity of approximation is context specific

  x may be approximated by y in one context and by z in another

  - Approximating Money

    Earnings : Rs. 1050 can be safely approximated by Rs. 1000
    Expenses : Rs. 1050 can be safely approximated by Rs. 1100

The Concept of Approximation

- x approximates y *iff*

  x can be used in place of y without causing any problems

- Validity of approximation is context specific

  x may be approximated by y in one context and by z in another

  - Approximating Money

    Earnings : Rs. 1050 can be safely approximated by Rs. 1000
    Expenses : Rs. 1050 can be safely approximated by Rs. 1100

  - Approximating Time

    Travel time: 2 hours required can be safely approximated by 3 hours
    Study time: 3 available days can be safely assumed to be only 2 days

Two Important Objectives in Data Flow Analysis

- The discovered data flow information should be
  - *Exhaustive*. No optimization opportunity should be missed
  - *Safe*. Optimizations which do not preserve semantics should not be enabled

Two Important Objectives in Data Flow Analysis

- The discovered data flow information should be
  - *Exhaustive*. No optimization opportunity should be missed
  - *Safe*. Optimizations which do not preserve semantics should not be enabled
- Conservative approximations of these objectives are allowed

Two Important Objectives in Data Flow Analysis

- The discovered data flow information should be
  - *Exhaustive*. No optimization opportunity should be missed
  - *Safe*. Optimizations which do not preserve semantics should not be enabled
- Conservative approximations of these objectives are allowed
- The intended use of data flow information ($\equiv$ context) determines validity of approximations

Context Determines the Validity of Approximations

Will not do incorrect optimization
May prohibit correct optimization

Will not miss any correct optimization
May enable incorrect optimization

Analysis	Application	Safe Approximation	Exhaustive Approximation

Context Determines the Validity of Approximations

Will not do incorrect optimization
May prohibit correct optimization

Will not miss any correct optimization
May enable incorrect optimization

Analysis	Application	Safe Approximation	Exhaustive Approximation
Live variables	Dead code elimination	A dead variable is considered live	A live variable is considered dead

Context Determines the Validity of Approximations

Will not do incorrect optimization
May prohibit correct optimization

Will not miss any correct optimization
May enable incorrect optimization

Analysis	Application	Safe Approximation	Exhaustive Approximation
Live variables	Dead code elimination	A dead variable is considered live	A live variable is considered dead
Available expressions	Common subexpression elimination	An available expression is considered non-available	A non-available expression is considered available

Context Determines the Validity of Approximations

Will not do incorrect optimization
May prohibit correct optimization

Will not miss any correct optimization
May enable incorrect optimization

Analysis	Application	Safe Approximation	Exhaustive Approximation
Live variables	Dead code elimination	A dead variable is considered live	A live variable is considered dead
Available expressions	Common subexpression elimination	An available expression is considered non-available	A non-available expression is considered available

Spurious Inclusion

Spurious Exclusion

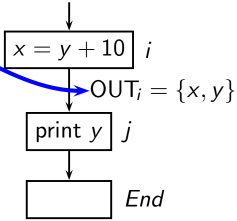
Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

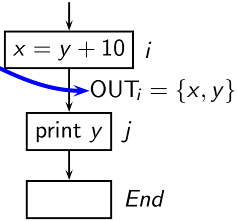
- Spurious inclusion of a non-live variable



Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

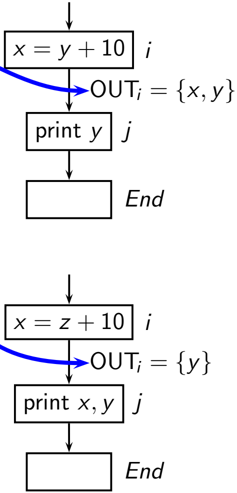
- Spurious inclusion of a non-live variable
  - A dead assignment may not be eliminated
  - Solution is sound but may be imprecise



Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

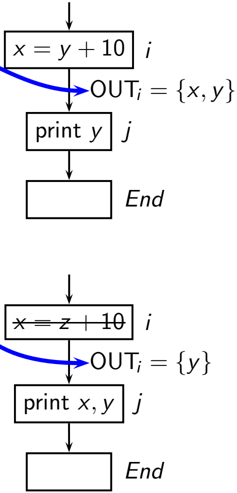
- Spurious inclusion of a non-live variable
  - A dead assignment may not be eliminated
  - Solution is sound but may be imprecise
- Spurious exclusion of a live variable



Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

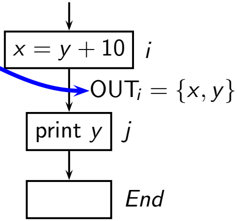
- Spurious inclusion of a non-live variable
  - A dead assignment may not be eliminated
  - Solution is sound but may be imprecise
- Spurious exclusion of a live variable
  - A useful assignment may be eliminated
  - Solution is unsound

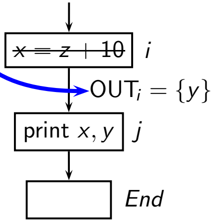


Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

- Spurious inclusion of a non-live variable
  - A dead assignment may not be eliminated
  - Solution is sound but may be imprecise
- Spurious exclusion of a live variable
  - A useful assignment may be eliminated
  - Solution is unsound
- Given $L_2 \supseteq L_1$ representing liveness information
  - Using L_2 in place of L_1 is sound
  - Using L_1 in place of L_2 may not be sound

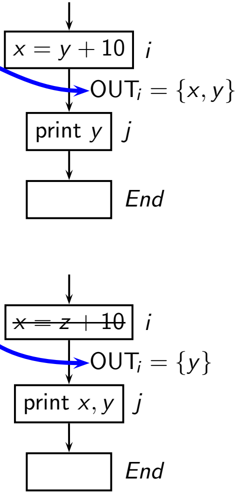




Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

- Spurious inclusion of a non-live variable
  - A dead assignment may not be eliminated
  - Solution is sound but may be imprecise
- Spurious exclusion of a live variable
  - A useful assignment may be eliminated
  - Solution is unsound
- Given $L_2 \supseteq L_1$ representing liveness information
  - Using L_2 in place of L_1 is sound
  - Using L_1 in place of L_2 may not be sound
- The smallest set of all live variables is most precise
  - Since liveness sets grow (confluence is $\cup$), we choose $\emptyset$ as the initial conservative value



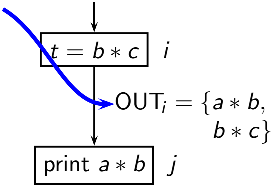
Soundness and Precision of Available Expressions Analysis

Consider common subexpression elimination based on availability information

Soundness and Precision of Available Expressions Analysis

Consider common subexpression elimination based on availability information

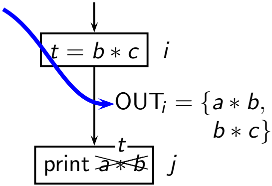
- Spurious inclusion of a non-available expression $a * b$



Soundness and Precision of Available Expressions Analysis

Consider common subexpression elimination based on availability information

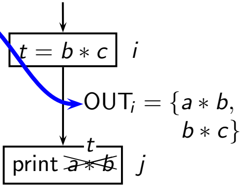
- Spurious inclusion of a non-available expression $a * b$
    - An occurrence of $a * b$ may be eliminated
    - Solution is unsound

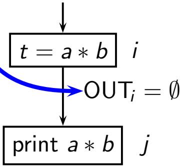


Soundness and Precision of Available Expressions Analysis

Consider common subexpression elimination based on availability information

- Spurious inclusion of a non-available expression $a * b$
    - An occurrence of $a * b$ may be eliminated
    - Solution is unsound
- Spurious exclusion of an available variable



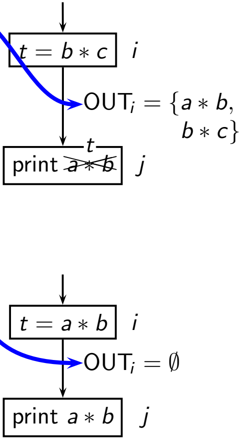


End

Soundness and Precision of Available Expressions Analysis

Consider common subexpression elimination based on availability information

- Spurious inclusion of a non-available expression $a * b$
  - An occurrence of $a * b$ may be eliminated
  - Solution is unsound
- Spurious exclusion of an available variable
  - An occurrence of $a * b$ may not be eliminated
  - Solution is sound but may be imprecise

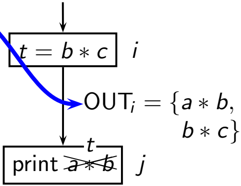


End

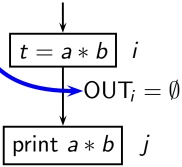
Soundness and Precision of Available Expressions Analysis

Consider common subexpression elimination based on availability information

- Spurious inclusion of a non-available expression $a * b$
  - An occurrence of $a * b$ may be eliminated
  - Solution is unsound



- Spurious exclusion of an available variable
  - An occurrence of $a * b$ may not be eliminated
  - Solution is sound but may be imprecise



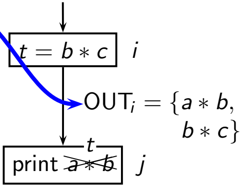
- Given $A_2 \supseteq A_1$ representing availability information
  - Using A_1 in place of A_2 is sound
  - Using A_2 in place of A_1 may not be sound

End

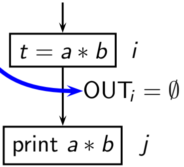
Soundness and Precision of Available Expressions Analysis

Consider common subexpression elimination based on availability information

- Spurious inclusion of a non-available expression $a * b$
  - An occurrence of $a * b$ may be eliminated
  - Solution is unsound



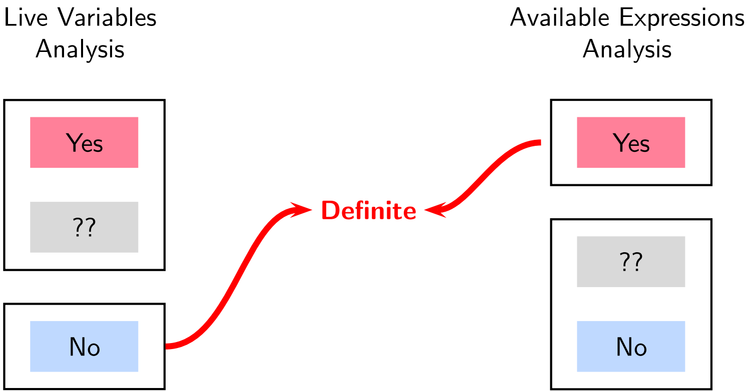
- Spurious exclusion of an available variable
  - An occurrence of $a * b$ may not be eliminated
  - Solution is sound but may be imprecise
- Given $A_2 \supseteq A_1$ representing availability information
  - Using A_1 in place of A_2 is sound
  - Using A_2 in place of A_1 may not be sound
- The largest set of available expressions is most precise
  - Since availability sets shrink (confluence is $\cap$), we choose $\mathbb{U}$ as the initial conservative value



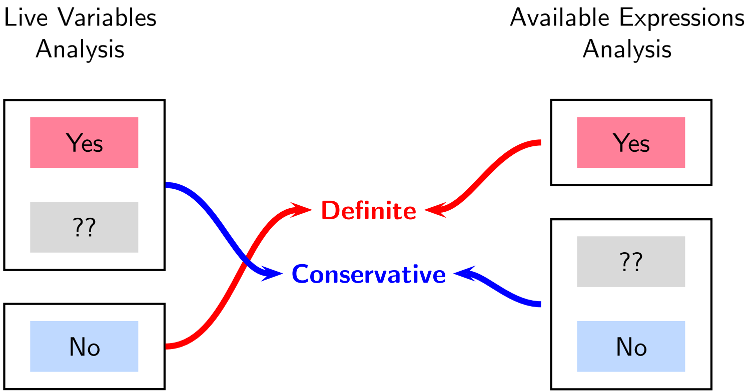
End

Conservative Approximation of Uncertain Information for Soundness

Conservative Approximation of Uncertain Information for Soundness



Conservative Approximation of Uncertain Information for Soundness



Partial Order Captures Approximation

- $\sqsubseteq$ captures valid approximations for safety

  $x \sqsubseteq y \;\Rightarrow\; x$ is *weaker than* y

  - The data flow information represented by x can be safely used in place of the data flow information represented by y
  - It may be imprecise, though

Partial Order Captures Approximation

- $\sqsubseteq$ captures valid approximations for safety

  $x \sqsubseteq y \;\Rightarrow\;$ x is *weaker than* y

  - The data flow information represented by x can be safely used in place of the data flow information represented by y
  - It may be imprecise, though

- $\sqsupseteq$ captures valid approximations for exhaustiveness

  $x \sqsupseteq y \;\Rightarrow\;$ x is *stronger than* y

  - The data flow information represented by x contains every value contained in the data flow information represented by y
    $x \sqcap y$ *will not compute a value weaker than* y
  - It may be unsafe, though

Partial Order Captures Approximation

- $\sqsubseteq$ captures valid approximations for safety

  $x \sqsubseteq y \;\Rightarrow\;$ x is *weaker than* y

  - The data flow information represented by x can be safely used in place of the data flow information represented by y
  - It may be imprecise, though

- $\sqsupseteq$ captures valid approximations for exhaustiveness

  $x \sqsupseteq y \;\Rightarrow\;$ x is *stronger than* y

  - The data flow information represented by x contains every value contained in the data flow information represented by y
    $x \sqcap y$ will not compute a value weaker than y
  - It may be unsafe, though

We want most exhaustive information which is also safe

Most Approximate Values in a Complete Lattice

- *Top.* $\forall x \in L,\ x \sqsubseteq \top$ Exhaustive approximation of all values

- *Bottom.* $\forall x \in L,\ \bot \sqsubseteq x$ Safe approximation of all values

Most Approximate Values in a Complete Lattice

- *Top.* $\forall x \in L,\ x \sqsubseteq \top$ Exhaustive approximation of all values
  - Using $\top$ in place of any data flow value will never miss out (or rule out) any possible value
- *Bottom.* $\forall x \in L,\ \bot \sqsubseteq x$ Safe approximation of all values

Most Approximate Values in a Complete Lattice

- *Top.* $\forall x \in L,\ x \sqsubseteq \top$ Exhaustive approximation of all values
  - Using $\top$ in place of any data flow value will never miss out (or rule out) any possible value
  - The consequences may be semantically *unsafe*, or *incorrect*
- *Bottom.* $\forall x \in L,\ \bot \sqsubseteq x$ Safe approximation of all values

Most Approximate Values in a Complete Lattice

- *Top.* $\forall x \in L,\ x \sqsubseteq \top$ Exhaustive approximation of all values
  - Using $\top$ in place of any data flow value will never miss out (or rule out) any possible value
  - The consequences may be semantically *unsafe*, or *incorrect*
- *Bottom.* $\forall x \in L,\ \bot \sqsubseteq x$ Safe approximation of all values
  - Using $\bot$ in place of any data flow value will never be *unsafe*, or *incorrect*

Most Approximate Values in a Complete Lattice

- *Top*. $\forall x \in L,\ x \sqsubseteq \top$ Exhaustive approximation of all values
  - Using $\top$ in place of any data flow value will never miss out (or rule out) any possible value
  - The consequences may be semantically *unsafe*, or *incorrect*
- *Bottom*. $\forall x \in L,\ \bot \sqsubseteq x$ Safe approximation of all values
  - Using $\bot$ in place of any data flow value will never be *unsafe*, or *incorrect*
  - The consequences may be *undefined* or *useless* because this replacement might miss out valid values

Most Approximate Values in a Complete Lattice

- *Top.* $\forall x \in L,\ x \sqsubseteq \top$ Exhaustive approximation of all values
  - Using $\top$ in place of any data flow value will never miss out (or rule out) any possible value
  - The consequences may be semantically *unsafe*, or *incorrect*
- *Bottom.* $\forall x \in L,\ \bot \sqsubseteq x$ Safe approximation of all values
  - Using $\bot$ in place of any data flow value will never be *unsafe*, or *incorrect*
  - The consequences may be *undefined* or *useless* because this replacement might miss out valid values

Appropriate orientation chosen by design

Setting Up Lattices

Available Expressions Analysis

$$
\begin{tikzcd}
 & \{e_1, e_2, e_3\} \arrow[dl] \arrow[d] \arrow[dr] & \\
\{e_1, e_2\} \arrow[d] \arrow[dr] & \{e_1, e_3\} \arrow[dl] \arrow[dr] & \{e_2, e_3\} \arrow[dl] \arrow[d] \\
\{e_1\} \arrow[dr] & \{e_2\} \arrow[d] & \{e_3\} \arrow[dl] \\
 & \emptyset &
\end{tikzcd}
$$

$\sqsubseteq$ is $\subseteq$

$\sqcap$ is $\cap$

Live Variables Analysis

$$
\begin{tikzcd}
 & \emptyset \arrow[dl] \arrow[d] \arrow[dr] & \\
\{v_1\} \arrow[d] \arrow[dr] & \{v_2\} \arrow[dl] \arrow[dr] & \{v_3\} \arrow[dl] \arrow[d] \\
\{v_1, v_2\} \arrow[dr] & \{v_1, v_3\} \arrow[d] & \{v_2, v_3\} \arrow[dl] \\
 & \{v_1, v_2, v_3\} &
\end{tikzcd}
$$

$\sqsubseteq$ is $\supseteq$

$\sqcap$ is $\cup$

Partial Order Relation

Reflexive	$x \sqsubseteq x$
Transitive	$x \sqsubseteq y, y \sqsubseteq z \Rightarrow x \sqsubseteq z$
Antisymmetric	$x \sqsubseteq y, y \sqsubseteq x \Leftrightarrow x = y$

Partial Order Relation

Reflexive	$x \sqsubseteq x$	x can be safely used in place of x
Transitive	$x \sqsubseteq y, y \sqsubseteq z \Rightarrow x \sqsubseteq z$	If x can be safely used in place of y and y can be safely used in place of z, then x can be safely used in place of z
Antisymmetric	$x \sqsubseteq y, y \sqsubseteq x \Leftrightarrow x = y$	If x can be safely used in place of y and y can be safely used in place of x, then x must be same as y

Merging Information

- $x \sqcap y$ computes the *greatest lower bound* of x and y i.e.
  largest z such that $z \sqsubseteq x$ and $z \sqsubseteq y$

  The largest safe approximation of combining data flow information x and y

Merging Information

- $x \sqcap y$ computes the *greatest lower bound* of x and y i.e. largest z such that $z \sqsubseteq x$ and $z \sqsubseteq y$

  The largest safe approximation of combining data flow information x and y

- Commutative $x \sqcap y = y \sqcap x$

  Associative $x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z$

  Idempotent $x \sqcap x = x$

Merging Information

- $x \sqcap y$ computes the *greatest lower bound* of x and y i.e.
  largest z such that $z \sqsubseteq x$ and $z \sqsubseteq y$

  The largest safe approximation of combining data flow information x and y

- Commutative $x \sqcap y = y \sqcap x$ — The order in which the data flow information is merged, does not matter

  Associative $x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z$ — Allow n-ary merging without any restriction on the order

  Idempotent $x \sqcap x = x$ — No loss of information if x is merged with itself

Merging Information

- $x \sqcap y$ computes the *greatest lower bound* of x and y i.e.
  largest z such that $z \sqsubseteq x$ and $z \sqsubseteq y$

  The largest safe approximation of combining data flow information x and y

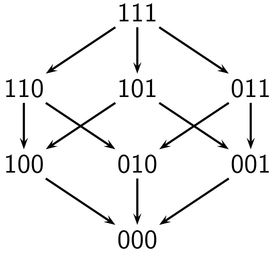
Commutative	$x \sqcap y = y \sqcap x$	The order in which the data flow information is merged, does not matter
Associative	$x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z$	Allow n-ary merging without any restriction on the order
Idempotent	$x \sqcap x = x$	No loss of information if x is merged with itself

- $\top$ is the identity of $\sqcap$
  - Presence of loops $\Rightarrow$ self dependence of data flow information
  - Using $\top$ as the initial value ensure exhaustiveness

More on Lattices in Data Flow Analysis

L = Lattice for all expressions	$\widehat{L}$ = Lattice for a single expression
$$\begin{tikzcd} & 111 \arrow[dl] \arrow[d] \arrow[dr] & \\ 110 \arrow[d] \arrow[dr] & 101 \arrow[dl] \arrow[dr] & 011 \arrow[dl] \arrow[d] \\ 100 \arrow[dr] & 010 \arrow[d] & 001 \arrow[dl] \\ & 000 & \end{tikzcd}$$	(Expression e is available) $$\begin{tikzcd} 1 \text{ or } \{e\} \arrow[d] \\ 0 \text{ or } \emptyset \end{tikzcd}$$ (Expressions e is not available)

More on Lattices in Data Flow Analysis

L = Lattice for all expressions	$\widehat{L}$ = Lattice for a single expression
	(Expression e is available) 1 or $\{e\}$ $\downarrow$ 0 or $\emptyset$ (Expressions e is not available)

Cartesian products if sets are used, vectors (or tuples) if bit are used

- $L = \widehat{L} \times \widehat{L} \times \widehat{L}$ and $x = \langle \widehat{x}_1, \widehat{x}_2, \widehat{x}_3 \rangle \in L$ where $\widehat{x}_i \in \widehat{L}$
- $\sqsubseteq = \widehat{\sqsubseteq} \times \widehat{\sqsubseteq} \times \widehat{\sqsubseteq}$ and $\sqcap = \widehat{\sqcap} \times \widehat{\sqcap} \times \widehat{\sqcap}$
- $\top = \widehat{\top} \times \widehat{\top} \times \widehat{\top}$ and $\bot = \widehat{\bot} \times \widehat{\bot} \times \widehat{\bot}$

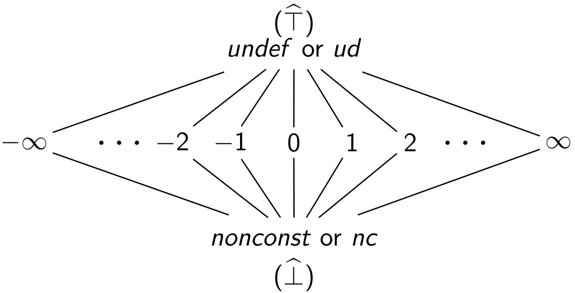
Component Lattice for Data Flow Information Represented By Bit Vectors

$$\begin{array}{c} (\widehat{\top}) \\ 1 \\ | \\ 0 \\ (\widehat{\bot}) \end{array} \qquad\qquad \begin{array}{c} (\widehat{\top}) \\ 0 \\ | \\ 1 \\ (\widehat{\bot}) \end{array}$$

$\sqcap$ is $\cap$ or Boolean AND

$\sqcap$ is $\cup$ or Boolean OR

Component Lattice for Integer Constant Propagation



- Overall lattice L is the set of mappings from variables to $\widehat{L}$
- $\sqcap$ and $\widehat{\sqcap}$ get defined by $\sqsubseteq$ and $\widehat{\sqsubseteq}$

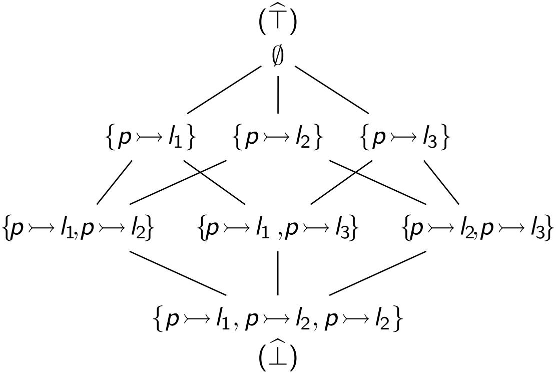
$\widehat{\sqcap}$	$\langle a, ud\rangle$	$\langle a, nc\rangle$	$\langle a, c_1\rangle$
$\langle a, ud\rangle$	$\langle a, ud\rangle$	$\langle a, nc\rangle$	$\langle a, c_1\rangle$
$\langle a, nc\rangle$	$\langle a, nc\rangle$	$\langle a, nc\rangle$	$\langle a, nc\rangle$
$\langle a, c_2\rangle$	$\langle a, c_2\rangle$	$\langle a, nc\rangle$	If $c_1 = c_2$ then $\langle a, c_1\rangle$ else $\langle a, nc\rangle$

Component Lattice for May Points-To Analysis

- Relation between pointer variables and locations in the memory

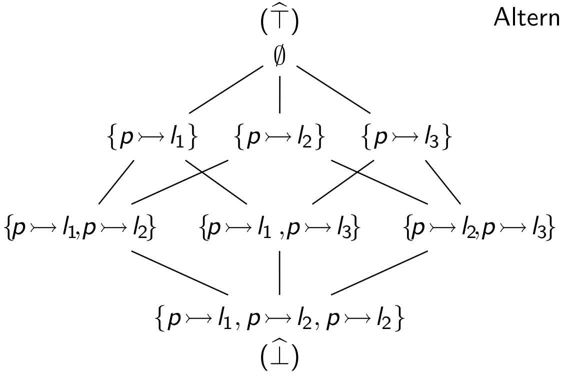
Component Lattice for May Points-To Analysis

- Relation between pointer variables and locations in the memory
- Assuming three locations l_1, l_2, and l_3, the component lattice for pointer p is

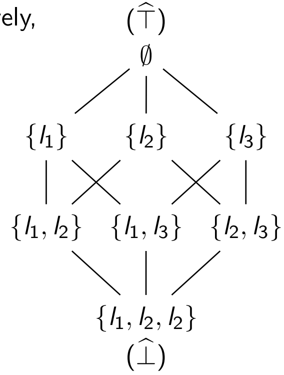


Component Lattice for May Points-To Analysis

- Relation between pointer variables and locations in the memory
- Assuming three locations l_1, l_2, and l_3, the component lattice for pointer p is

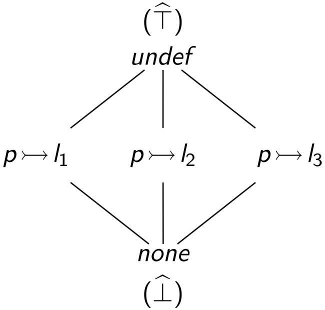


Alternatively,

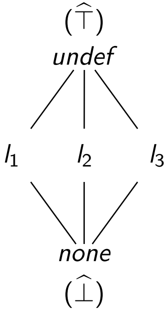


Component Lattice for Must Points-To Analysis

- A pointer can point to at most one location

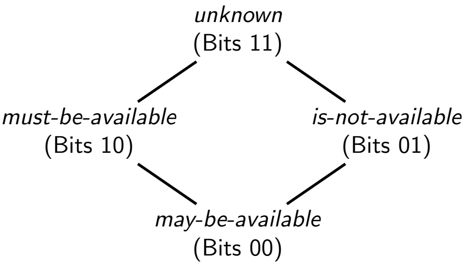


Alternatively,



Combined Total and Partial Availability Analysis

- Two bits per expression rather than one. Can be implemented using AND (as below) or using OR (reversed lattice)

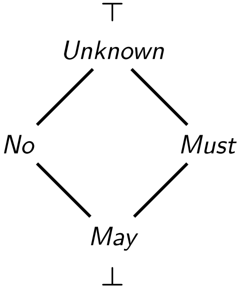


  Can also be implemented as a product of 1-0 and 0-1 lattice with AND for the first bit and OR for the second bit

- What approximation of safety does this lattice capture?

  Uncertain information (= no optimization) is guaranteed to be safe

General Lattice for May-Must Analysis



Interpreting data flow values

- *Unknown.* Nothing is known as yet
- *No.* Information does not hold along any path
- *Must.* Information must hold along all paths
- *May.* Information may hold along some path

Possible Applications

- Pointer Analysis : No need of separate of *May* and *Must* analyses
  eg. $(p \rightarrowtail l, May)$, $(p \rightarrowtail l, Must)$, $(p \rightarrowtail l, No)$, or $(p \rightarrowtail l, Unknown)$
- Type Inferencing for Dynamically Checked Languages

Part 6

Flow Functions

Flow Functions: An Outline of Our Discussion

- Defining flow functions
- Properties of flow functions

  (Some properties discussed in the context of solutions of data flow analysis)

The Set of Flow Functions

- F is the set of functions $f : L \to L$ such that
  - F contains an identity function

    To model "empty" statements, i.e. statements which do not influence the data flow information
  - F is closed under composition

    Cumulative effect of statements should generate data flow information from the same set
  - For every $x \in L$, there must be a finite set of flow functions $\{f_1, f_2, \ldots f_m\} \subseteq F$ such that

$$x = \prod_{1 \le i \le m} f_i(BI)$$

- Properties of f
  - Monotonicity and Distributivity
  - Loop Closure Boundedness and Separability

Flow Functions in Bit Vector Data Flow Frameworks

- Bit Vector Frameworks: Available Expressions Analysis, Reaching Definitions Analysis Live variable Analysis, Anticipable Expressions Analysis, Partial Redundancy Elimination etc
  - All functions can be defined in terms of constant Gen and Kill

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

  - Lattices are powersets with partial orders as $\subseteq$ or $\supseteq$ relations
  - Information is merged using $\cap$ or $\cup$

Flow Functions in Bit Vector Data Flow Frameworks

- Bit Vector Frameworks: Available Expressions Analysis, Reaching Definitions Analysis Live variable Analysis, Anticipable Expressions Analysis, Partial Redundancy Elimination etc
  - All functions can be defined in terms of constant Gen and Kill

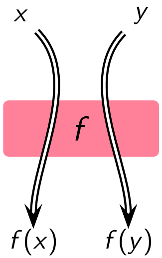
$$f(x) = \text{Gen} \cup (x - \text{Kill})$$

  - Lattices are powersets with partial orders as $\subseteq$ or $\supseteq$ relations
  - Information is merged using $\cap$ or $\cup$
- Flow functions in Strong Liveness Analysis, Pointer Analyses, Constant Propagation, Possibly Uninitialized Variables cannot be expressed using constant Gen and Kill

  Local context alone is not sufficient to describe the effect of statements fully

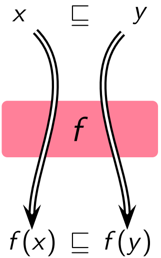
Monotonicity of Flow Functions

- Partial order is preserved: If x can be safely used in place of y then $f(x)$ can be safely used in place of $f(y)$



Monotonicity of Flow Functions

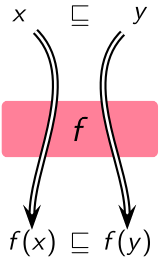
- Partial order is preserved: If x can be safely used in place of y then $f(x)$ can be safely used in place of $f(y)$



Monotonicity of Flow Functions

- Partial order is preserved: If x can be safely used in place of y then $f(x)$ can be safely used in place of $f(y)$

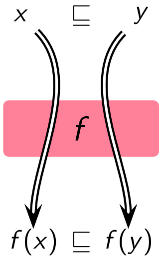
$$\forall x, y \in L, x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$$



Monotonicity of Flow Functions

- Partial order is preserved: If x can be safely used in place of y then $f(x)$ can be safely used in place of $f(y)$

$$\forall x, y \in L, x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$$



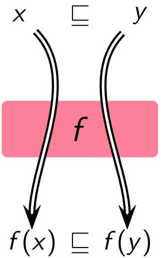
- Alternative definition

$$\forall x, y \in L, f(x \sqcap y) \sqsubseteq f(x) \sqcap f(y)$$

Monotonicity of Flow Functions

- Partial order is preserved: If x can be safely used in place of y then $f(x)$ can be safely used in place of $f(y)$

$$\forall x, y \in L, x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$$



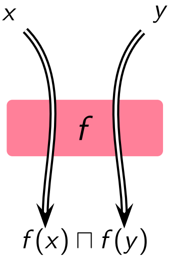
- Alternative definition

$$\forall x, y \in L, f(x \sqcap y) \sqsubseteq f(x) \sqcap f(y)$$

- Merging at intermediate points in shared segments of paths is safe (However, it may lead to imprecision)

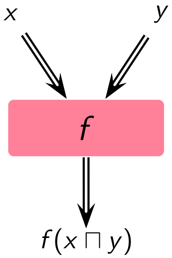
Distributivity of Flow Functions

- Merging distributes over function application



Distributivity of Flow Functions

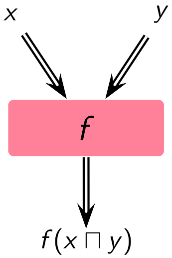
- Merging distributes over function application



Distributivity of Flow Functions

- Merging distributes over function application

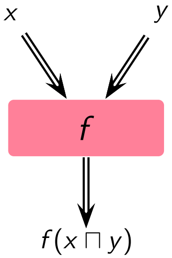
$\forall x, y \in L, f(x \sqcap y) = f(x) \sqcap f(y)$



Distributivity of Flow Functions

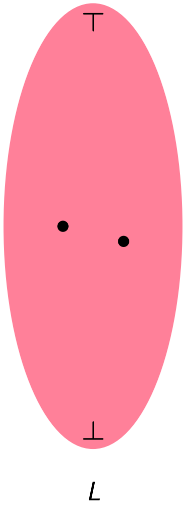
- Merging distributes over function application

$\forall x, y \in L, f(x \sqcap y) = f(x) \sqcap f(y)$

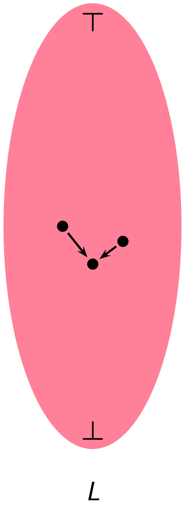


- Merging at intermediate points in shared segments of paths does not lead to imprecision

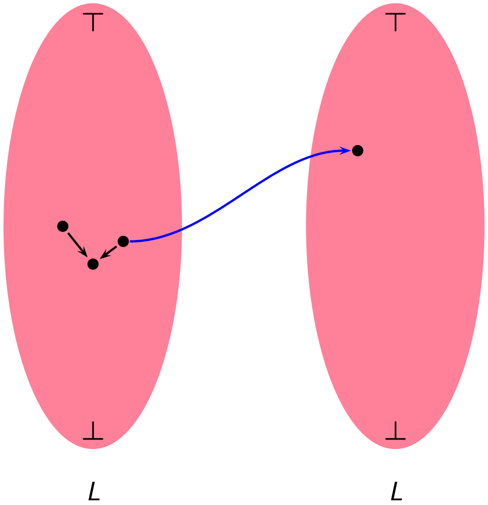
Monotonicity and Distributivity



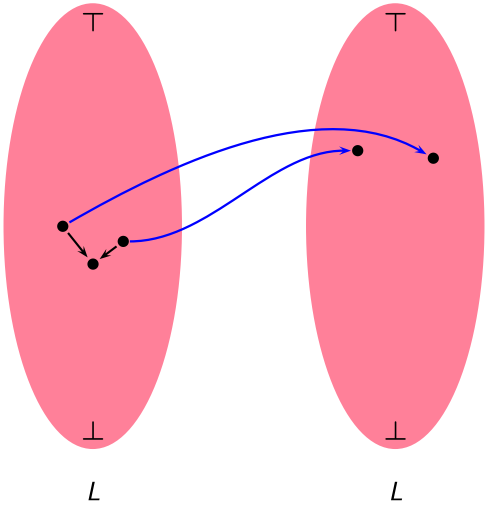
Monotonicity and Distributivity



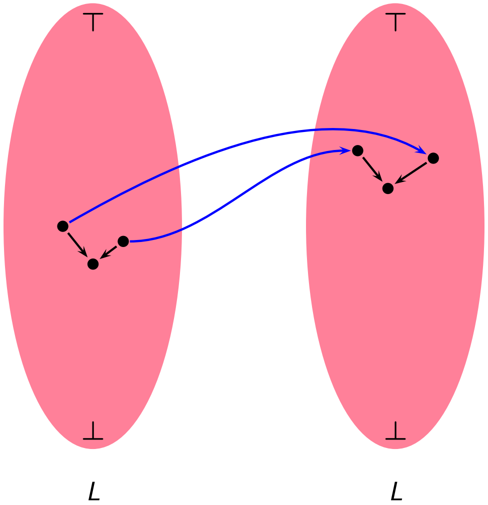
Monotonicity and Distributivity



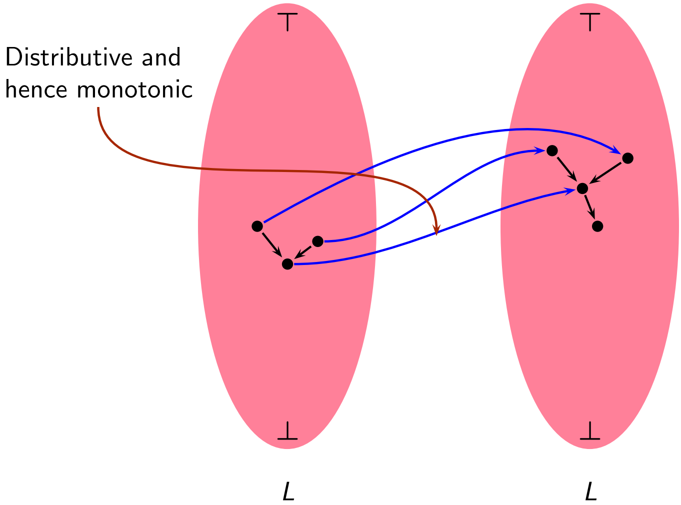
Monotonicity and Distributivity



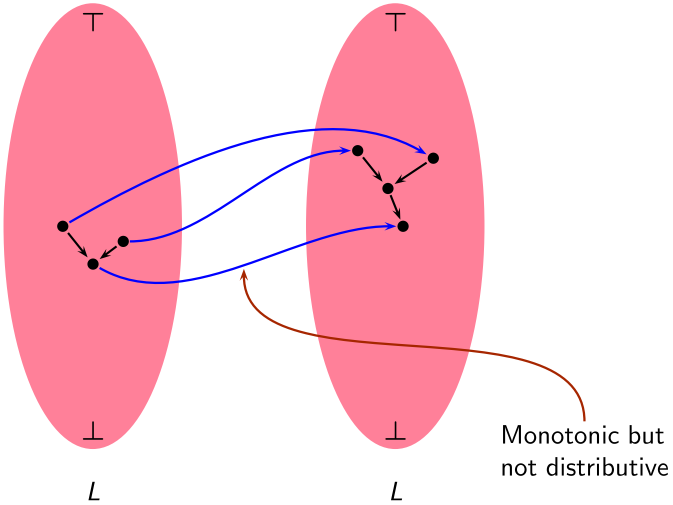
Monotonicity and Distributivity



Monotonicity and Distributivity



Monotonicity and Distributivity



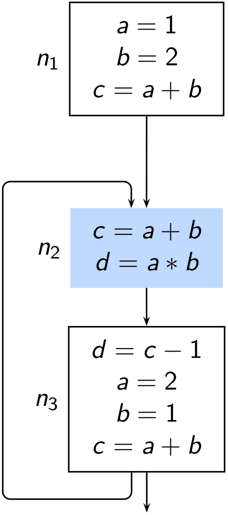
Distributivity of Bit Vector Frameworks

$$
\begin{aligned}
f(x) &= \text{Gen} \cup (x - \text{Kill}) \\
f(y) &= \text{Gen} \cup (y - \text{Kill})
\end{aligned}
$$

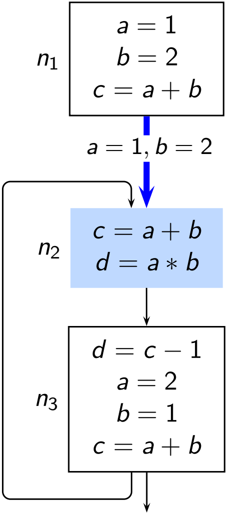
$$
\begin{aligned}
f(x \cup y) &= \text{Gen} \cup ((x \cup y) - \text{Kill}) \\
&= \text{Gen} \cup ((x - \text{Kill}) \cup (y - \text{Kill})) \\
&= (\text{Gen} \cup (x - \text{Kill}) \cup \text{Gen} \cup (y - \text{Kill})) \\
&= f(x) \cup f(y)
\end{aligned}
$$

$$
\begin{aligned}
f(x \cap y) &= \text{Gen} \cup ((x \cap y) - \text{Kill}) \\
&= \text{Gen} \cup ((x - \text{Kill}) \cap (y - \text{Kill})) \\
&= (\text{Gen} \cup (x - \text{Kill}) \cap \text{Gen} \cup (y - \text{Kill})) \\
&= f(x) \cap f(y)
\end{aligned}
$$

Non-Distributivity of Constant Propagation

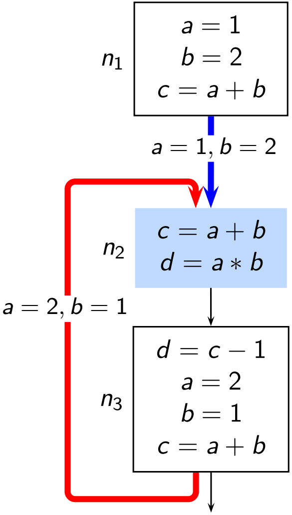


Non-Distributivity of Constant Propagation



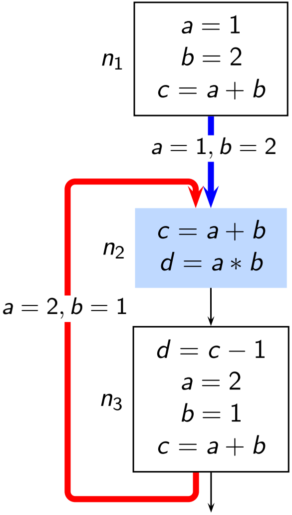
- $x = \langle 1, 2, 3, ud \rangle$ (Along $Out_{n_1} \rightarrow In_{n_2}$)

Non-Distributivity of Constant Propagation



- $x = \langle 1, 2, 3, ud \rangle$ (Along $Out_{n_1} \rightarrow In_{n_2}$)
- $y = \langle 2, 1, 3, 2 \rangle$ (Along $Out_{n_3} \rightarrow In_{n_2}$)

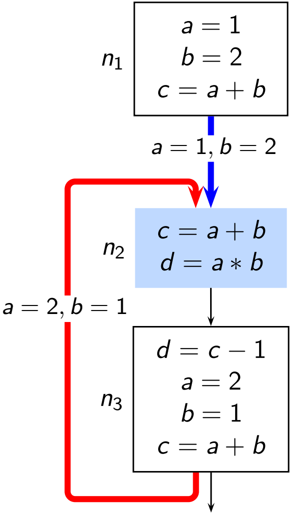
Non-Distributivity of Constant Propagation



- $x = \langle 1, 2, 3, ud \rangle$ (Along $Out_{n_1} \rightarrow In_{n_2}$)
- $y = \langle 2, 1, 3, 2 \rangle$ (Along $Out_{n_3} \rightarrow In_{n_2}$)
- Function application for block n_2 before merging

$$\begin{aligned} f(x) \sqcap f(y) &= f(\langle 1, 2, 3, ud \rangle) \sqcap f(\langle 2, 1, 3, 2 \rangle) \\ &= \langle 1, 2, 3, 2 \rangle \sqcap \langle 2, 1, 3, 2 \rangle \\ &= \langle \widehat{\perp}, \widehat{\perp}, 3, 2 \rangle \end{aligned}$$

Non-Distributivity of Constant Propagation



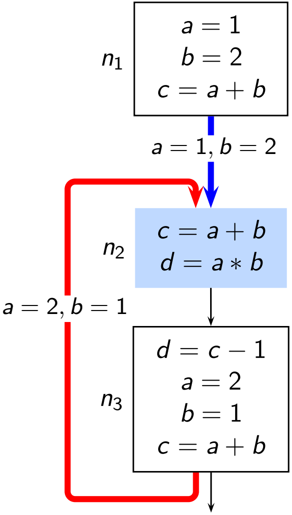
- $x = \langle 1, 2, 3, ud \rangle$ (Along $Out_{n_1} \rightarrow In_{n_2}$)
- $y = \langle 2, 1, 3, 2 \rangle$ (Along $Out_{n_3} \rightarrow In_{n_2}$)
- Function application for block n_2 before merging

$$\begin{aligned} f(x) \sqcap f(y) &= f(\langle 1, 2, 3, ud \rangle) \sqcap f(\langle 2, 1, 3, 2 \rangle) \\ &= \langle 1, 2, 3, 2 \rangle \sqcap \langle 2, 1, 3, 2 \rangle \\ &= \langle \widehat{\bot}, \widehat{\bot}, 3, 2 \rangle \end{aligned}$$

- Function application for block n_2 after merging

$$\begin{aligned} f(x \sqcap y) &= f(\langle 1, 2, 3, ud \rangle \sqcap \langle 2, 1, 3, 2 \rangle) \\ &= f(\langle \widehat{\bot}, \widehat{\bot}, 3, 2 \rangle) \\ &= \langle \widehat{\bot}, \widehat{\bot}, \widehat{\bot}, \widehat{\bot} \rangle \end{aligned}$$

Non-Distributivity of Constant Propagation



- $x = \langle 1, 2, 3, ud \rangle$ (Along $Out_{n_1} \rightarrow In_{n_2}$)
- $y = \langle 2, 1, 3, 2 \rangle$ (Along $Out_{n_3} \rightarrow In_{n_2}$)
- Function application for block n_2 before merging

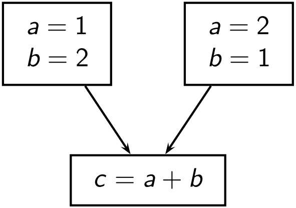
$$\begin{aligned} f(x) \sqcap f(y) &= f(\langle 1, 2, 3, ud \rangle) \sqcap f(\langle 2, 1, 3, 2 \rangle) \\ &= \langle 1, 2, 3, 2 \rangle \sqcap \langle 2, 1, 3, 2 \rangle \\ &= \langle \widehat{\bot}, \widehat{\bot}, 3, 2 \rangle \end{aligned}$$

- Function application for block n_2 after merging

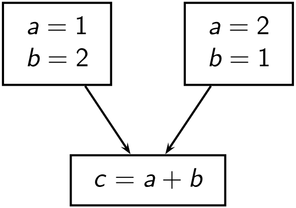
$$\begin{aligned} f(x \sqcap y) &= f(\langle 1, 2, 3, ud \rangle \sqcap \langle 2, 1, 3, 2 \rangle) \\ &= f(\langle \widehat{\bot}, \widehat{\bot}, 3, 2 \rangle) \\ &= \langle \widehat{\bot}, \widehat{\bot}, \widehat{\bot}, \widehat{\bot} \rangle \end{aligned}$$

- $f(x \sqcap y) \sqsubset f(x) \sqcap f(y)$

Why is Constant Propagation Non-Distributive?



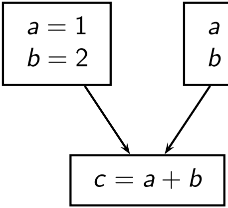
Why is Constant Propagation Non-Distributive?



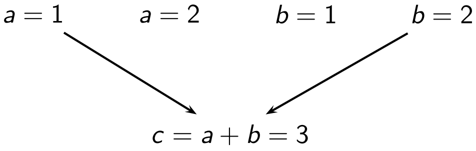
Possible combinations due to merging

$a = 1$ $\quad a = 2$ $\quad b = 1$ $\quad b = 2$

Why is Constant Propagation Non-Distributive?

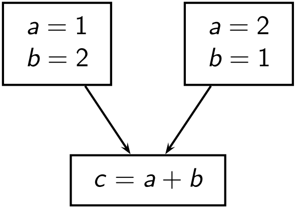


Possible combinations due to merging



- Correct combination

Why is Constant Propagation Non-Distributive?



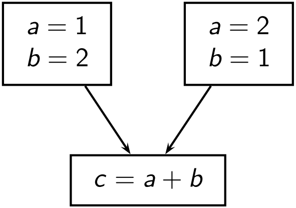
Possible combinations due to merging

$a = 1$ $\quad$ $a = 2$ $\quad$ $b = 1$ $\quad$ $b = 2$

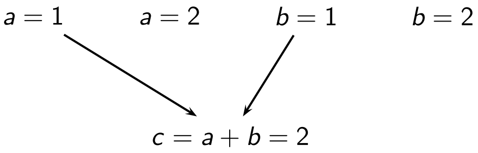
$c = a + b = 3$

- Correct combination

Why is Constant Propagation Non-Distributive?

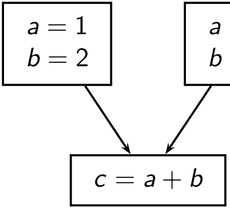


Possible combinations due to merging

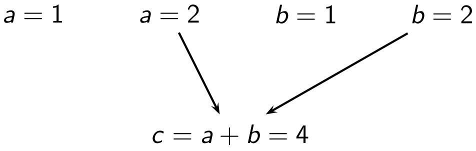


- Wrong combination
- Mutually exclusive information
- No execution path along which this information holds

Why is Constant Propagation Non-Distributive?



Possible combinations due to merging



- Wrong combination
- Mutually exclusive information
- No execution path along which this information holds

Part 7

Solutions of Data Flow Analysis

Solutions of Data Flow Analysis: An Outline of Our Discussion

- MoP and MFP assignments and their relationship
- Existence of MoP assignment
    - Boundedness of flow functions
- Existence and Computability of MFP assignment
    - Flow functions Vs. function computed by data flow equations
- Soundness of MFP solution

Solutions of Data Flow Analysis

- An assignment A associates data flow values with program points
  $A \sqsubseteq B$ if for all program points p, $A(p) \sqsubseteq B(p)$
- Performing data flow analysis

  Given

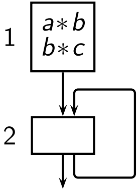
  - A set of flow functions, a lattice, and merge operation
  - A program flow graph with a mapping from nodes to flow functions

  Find out

  - An assignment A which is as exhaustive as possible and is safe

An Example For Available Expressions Analysis

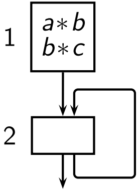
Program



Some Assignments							
	A_0	A_1	A_2	A_3	A_4	A_5	A_6
In_1	11	00	00	00	00	00	00
Out_1	11	11	00	11	11	11	11
In_2	11	11	00	00	10	01	01
Out_2	11	11	00	00	10	01	10

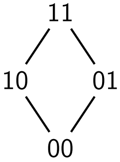
An Example For Available Expressions Analysis

Program



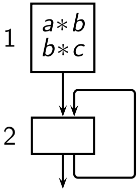
Some Assignments							
	A_0	A_1	A_2	A_3	A_4	A_5	A_6
In_1	11	00	00	00	00	00	00
Out_1	11	11	00	11	11	11	11
In_2	11	11	00	00	10	01	01
Out_2	11	11	00	00	10	01	10

Lattice L of data flow values at a node



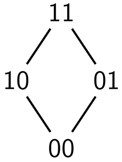
An Example For Available Expressions Analysis

Program

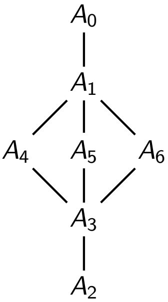


Some Assignments							
	A_0	A_1	A_2	A_3	A_4	A_5	A_6
In_1	11	00	00	00	00	00	00
Out_1	11	11	00	11	11	11	11
In_2	11	11	00	00	10	01	01
Out_2	11	11	00	00	10	01	10

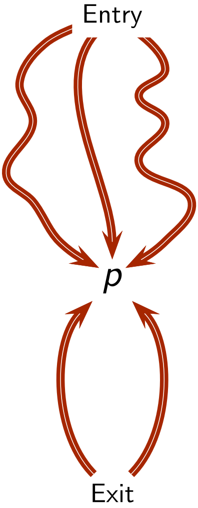
Lattice L of data flow values at a node



Lattice $L \times L \times L \times L$ for data flow values at all nodes



Meet Over Paths (MoP) Assignment

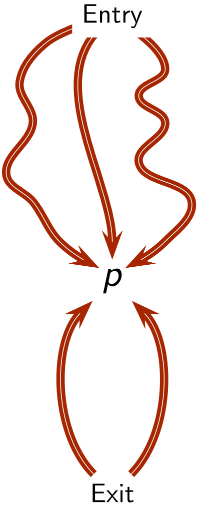


- The largest safe approximation of the information reaching a program point along all information flow paths

$$MoP(p) \quad = \bigsqcap_{\rho \in Paths(p)} f_\rho(BI)$$

  - f_ρ represents the compositions of flow functions along ρ
  - BI refers to the relevant information from the calling context
  - All execution paths are considered potentially executable by ignoring the results of conditionals

Meet Over Paths (MoP) Assignment



- The largest safe approximation of the information reaching a program point along all information flow paths

$$MoP(p) \quad = \prod_{\rho \,\in\, Paths(p)} f_\rho(BI)$$

  - f_ρ represents the compositions of flow functions along ρ
  - BI refers to the relevant information from the calling context
  - All execution paths are considered potentially executable by ignoring the results of conditionals

- Any $Info(p) \sqsubseteq MoP(p)$ is safe

Maximum Fixed Point (MFP) Assignment

- Difficulties in computing MoP assignment

Maximum Fixed Point (MFP) Assignment

- Difficulties in computing MoP assignment
  - In the presence of cycles there are infinite paths

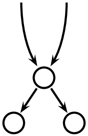
    If all paths need to be traversed $\Rightarrow$ Undecidability

Maximum Fixed Point (MFP) Assignment

- Difficulties in computing MoP assignment
  - In the presence of cycles there are infinite paths

    If all paths need to be traversed ⇒ Undecidability
  - Even if a program is acyclic, every conditional multiplies the number of paths by two

    If all paths need to be traversed ⇒ Intractability



Maximum Fixed Point (MFP) Assignment

- Difficulties in computing MoP assignment
  - In the presence of cycles there are infinite paths

    If all paths need to be traversed $\Rightarrow$ Undecidability
  - Even if a program is acyclic, every conditional multiplies the number of paths by two

    If all paths need to be traversed $\Rightarrow$ Intractability
- Why not merge information at intermediate points?
  - Merging is safe but may lead to imprecision
  - Computes fixed point solutions of data flow equations

Maximum Fixed Point (MFP) Assignment

- Difficulties in computing MoP assignment

  Path based specification

  - In the presence of cycles there are infinite paths

    If all paths need to be traversed $\Rightarrow$ Undecidability
  - Even if a program is acyclic, every conditional multiplies the number of paths by two

    If all paths need to be traversed $\Rightarrow$ Intractability
- Why not merge information at intermediate points?

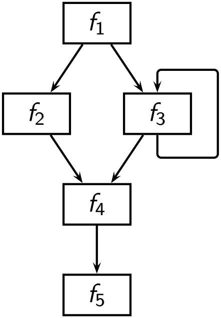
  Edge based specifications

  - Merging is safe but may lead to imprecision
  - Computes fixed point solutions of data flow equations

Computing MFP Vs. Computing MoP

Expression Tree for MFP

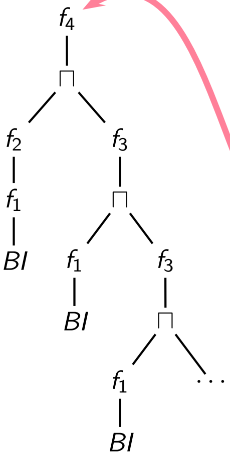
Program



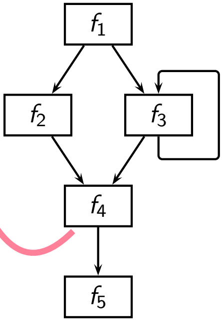
Expression Tree for MoP

Computing MFP Vs. Computing MoP

Expression Tree for MFP

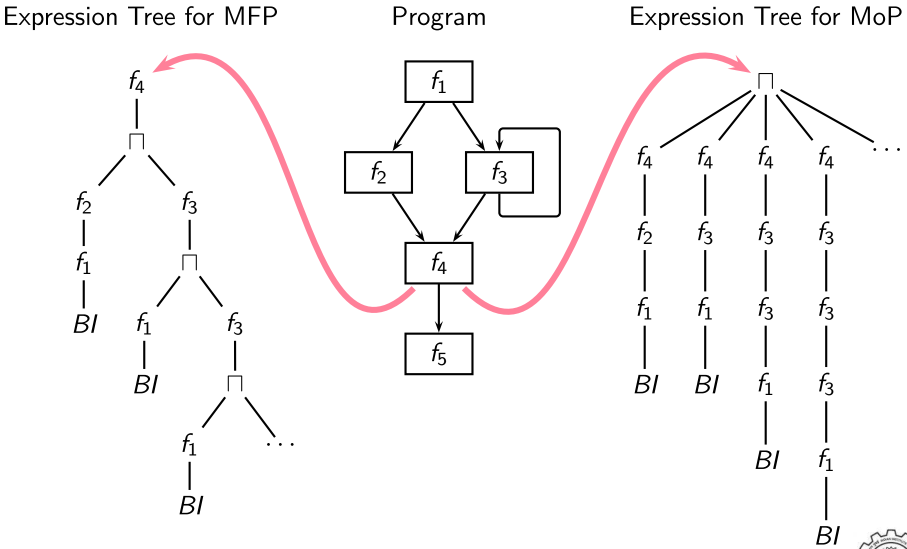


Program

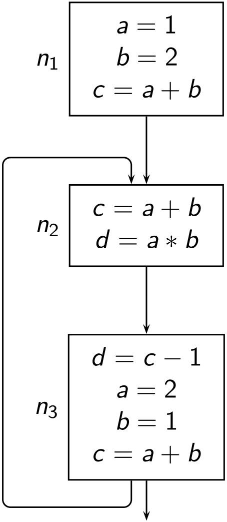


Expression Tree for MoP

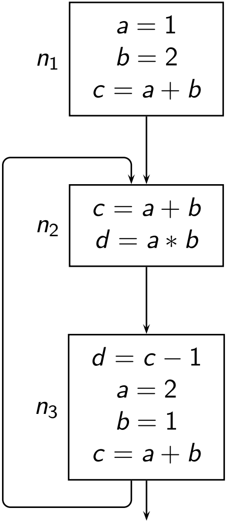
Computing MFP Vs. Computing MoP



Assignments for Constant Propagation Example



Assignments for Constant Propagation Example



MoP

$\langle \widehat{\top}, \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle$

$\langle 1, 2, 3, \widehat{\top} \rangle$

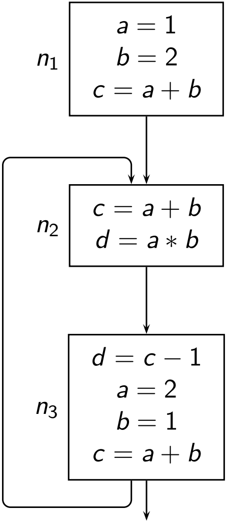
$\langle \widehat{\bot}, \widehat{\bot}, 3, 2 \rangle$

$\langle \widehat{\bot}, \widehat{\bot}, 3, 2 \rangle$

$\langle \widehat{\bot}, \widehat{\bot}, 3, 2 \rangle$

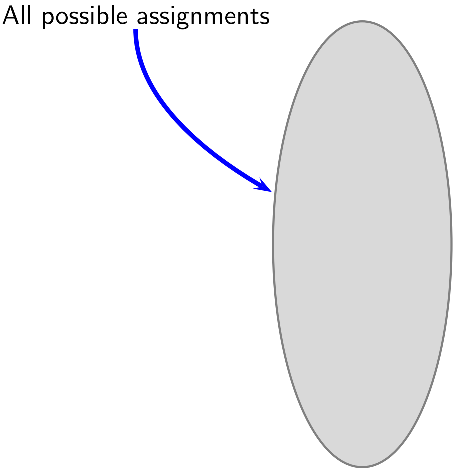
$\langle 2, 1, 3, 2 \rangle$

Assignments for Constant Propagation Example

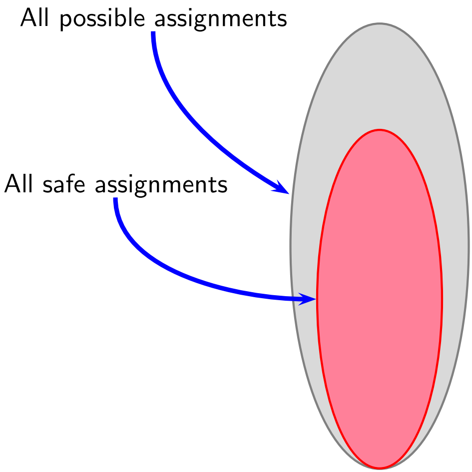


	MoP	MFP
n_1 entry	$\langle \widehat{\top}, \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle$	$\langle \widehat{\top}, \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle$
n_1 exit	$\langle 1, 2, 3, \widehat{\top} \rangle$	$\langle 1, 2, 3, \widehat{\top} \rangle$
n_2 entry	$\langle \widehat{\bot}, \widehat{\bot}, 3, 2 \rangle$	$\langle \widehat{\bot}, \widehat{\bot}, 3, \widehat{\bot} \rangle$
n_2 exit	$\langle \widehat{\bot}, \widehat{\bot}, 3, 2 \rangle$	$\langle \widehat{\bot}, \widehat{\bot}, \widehat{\bot}, \widehat{\bot} \rangle$
n_3 entry	$\langle \widehat{\bot}, \widehat{\bot}, 3, 2 \rangle$	$\langle \widehat{\bot}, \widehat{\bot}, \widehat{\bot}, \widehat{\bot} \rangle$
n_3 exit	$\langle 2, 1, 3, 2 \rangle$	$\langle 2, 1, 3, \widehat{\bot} \rangle$

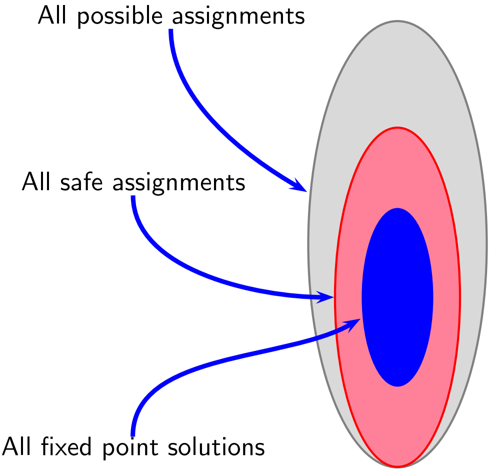
Possible Assignments as Solutions of Data Flow Analyses



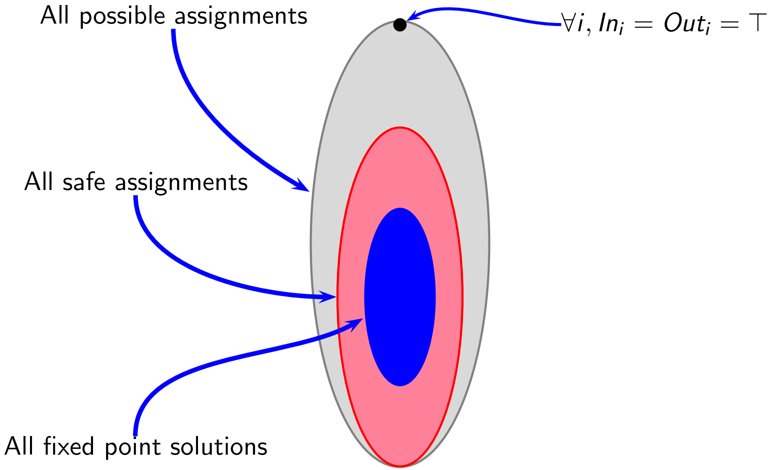
Possible Assignments as Solutions of Data Flow Analyses



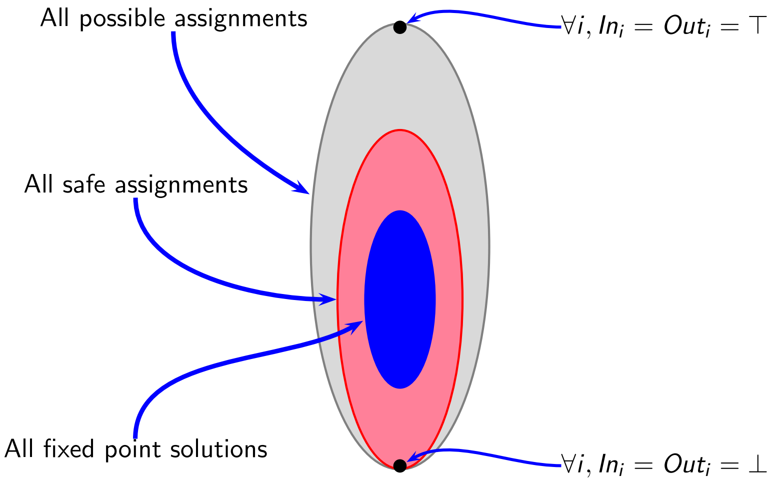
Possible Assignments as Solutions of Data Flow Analyses



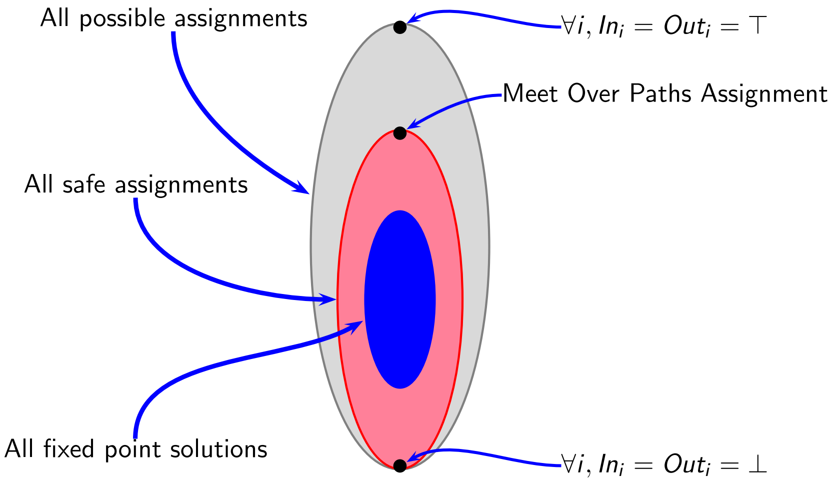
Possible Assignments as Solutions of Data Flow Analyses



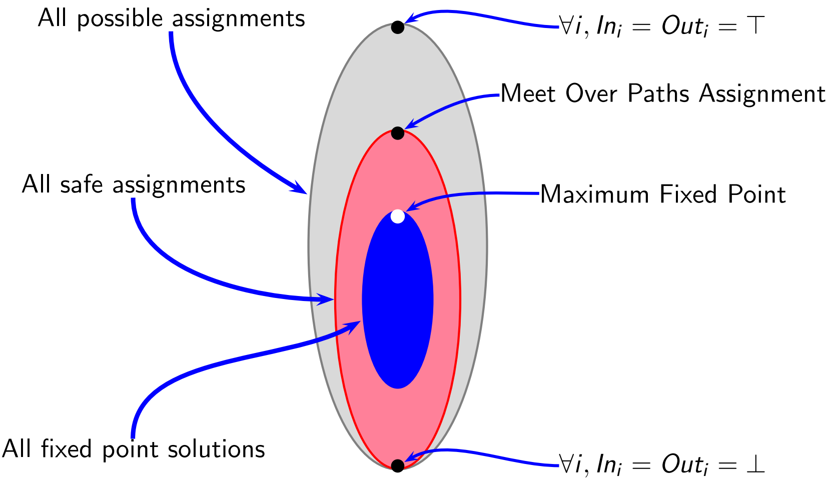
Possible Assignments as Solutions of Data Flow Analyses



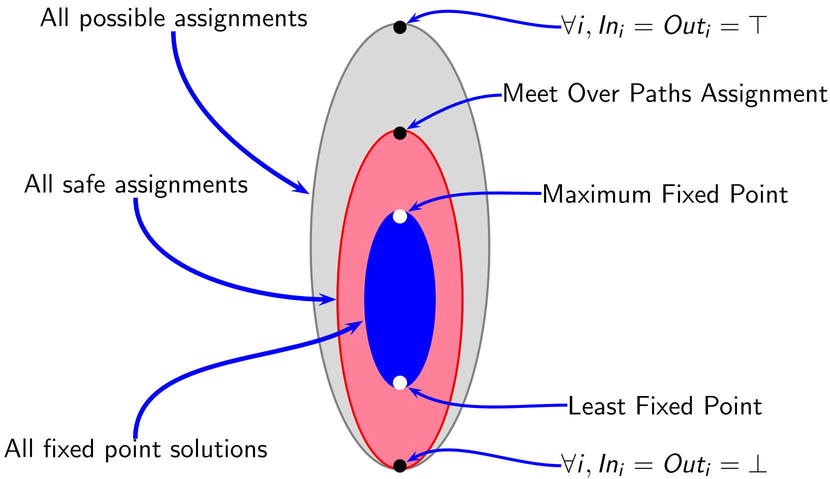
Possible Assignments as Solutions of Data Flow Analyses



Possible Assignments as Solutions of Data Flow Analyses

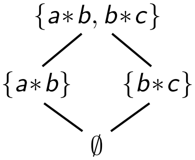


Possible Assignments as Solutions of Data Flow Analyses



An Instance of Available Expressions Analysis

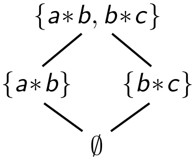
Lattice



Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
$f_\top$	$\{a*b, b*c\}$	f_{id}	x
$f_\perp$	$\emptyset$	f_c	$x \cup \{a*b\}$
f_a	$\{a*b\}$	f_d	$x \cup \{b*c\}$
f_b	$\{b*c\}$	f_e	$x - \{a*b\}$
		f_f	$x - \{b*c\}$

An Instance of Available Expressions Analysis

Lattice

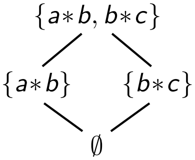


Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
$f_\top$	$\{a*b, b*c\}$	f_{id}	x
$f_\bot$	$\emptyset$	f_c	$x \cup \{a*b\}$
f_a	$\{a*b\}$	f_d	$x \cup \{b*c\}$
f_b	$\{b*c\}$	f_e	$x - \{a*b\}$
		f_f	$x - \{b*c\}$

- Is the lattice a meet semilattice?

An Instance of Available Expressions Analysis

Lattice

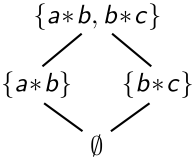


Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
$f_\top$	$\{a*b, b*c\}$	f_{id}	x
$f_\perp$	$\emptyset$	f_c	$x \cup \{a*b\}$
f_a	$\{a*b\}$	f_d	$x \cup \{b*c\}$
f_b	$\{b*c\}$	f_e	$x - \{a*b\}$
		f_f	$x - \{b*c\}$

- Is the lattice a meet semilattice?
- What is the meet operation that computes glb?

An Instance of Available Expressions Analysis

Lattice

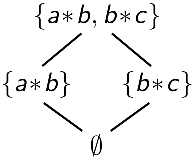


Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
$f_\top$	$\{a*b, b*c\}$	f_{id}	x
$f_\bot$	$\emptyset$	f_c	$x \cup \{a*b\}$
f_a	$\{a*b\}$	f_d	$x \cup \{b*c\}$
f_b	$\{b*c\}$	f_e	$x - \{a*b\}$
		f_f	$x - \{b*c\}$

- Is the lattice a meet semilattice?
- What is the meet operation that computes glb?
- Are all strictly descending chains finite?

An Instance of Available Expressions Analysis

Lattice



Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
$f_\top$	$\{a*b, b*c\}$	f_{id}	x
$f_\bot$	$\emptyset$	f_c	$x \cup \{a*b\}$
f_a	$\{a*b\}$	f_d	$x \cup \{b*c\}$
f_b	$\{b*c\}$	f_e	$x - \{a*b\}$
		f_f	$x - \{b*c\}$

- Is the lattice a meet semilattice?
- What is the meet operation that computes glb?
- Are all strictly descending chains finite?
- Does the function space have an identity function?

An Instance of Available Expressions Analysis

Lattice

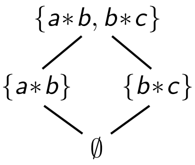
$$
\begin{tikzcd}[column sep=small]
 & \{a*b, b*c\} \arrow[dl, dash] \arrow[dr, dash] & \\
\{a*b\} \arrow[dr, dash] & & \{b*c\} \arrow[dl, dash] \\
 & \emptyset &
\end{tikzcd}
$$

Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
$f_\top$	$\{a*b, b*c\}$	f_{id}	x
$f_\bot$	$\emptyset$	f_c	$x \cup \{a*b\}$
f_a	$\{a*b\}$	f_d	$x \cup \{b*c\}$
f_b	$\{b*c\}$	f_e	$x - \{a*b\}$
		f_f	$x - \{b*c\}$

- Is the lattice a meet semilattice?
- What is the meet operation that computes glb?
- Are all strictly descending chains finite?
- Does the function space have an identity function?
- Are all values in the lattice computable from a finite merge of flow functions?

An Instance of Available Expressions Analysis

Lattice

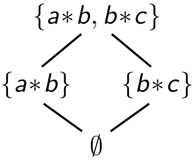


Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
$f_\top$	$\{a*b, b*c\}$	f_{id}	x
$f_\bot$	$\emptyset$	f_c	$x \cup \{a*b\}$
f_a	$\{a*b\}$	f_d	$x \cup \{b*c\}$
f_b	$\{b*c\}$	f_e	$x - \{a*b\}$
		f_f	$x - \{b*c\}$

- Is the lattice a meet semilattice?
- What is the meet operation that computes glb?
- Are all strictly descending chains finite?
- Does the function space have an identity function?
- Are all values in the lattice computable from a finite merge of flow functions?
- Is the function space closed under composition?

An Instance of Available Expressions Analysis

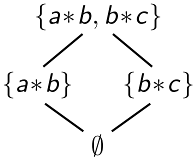
Lattice



Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
$f_\top$	$\{a*b, b*c\}$	f_{id}	x
$f_\perp$	$\emptyset$	f_c	$x \cup \{a*b\}$
f_a	$\{a*b\}$	f_d	$x \cup \{b*c\}$
f_b	$\{b*c\}$	f_e	$x - \{a*b\}$
		f_f	$x - \{b*c\}$

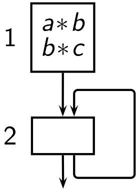
An Instance of Available Expressions Analysis

Lattice



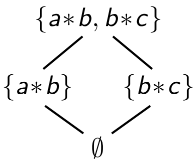
Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
$f_\top$	$\{a*b, b*c\}$	f_{id}	x
$f_\bot$	$\emptyset$	f_c	$x \cup \{a*b\}$
f_a	$\{a*b\}$	f_d	$x \cup \{b*c\}$
f_b	$\{b*c\}$	f_e	$x - \{a*b\}$
		f_f	$x - \{b*c\}$

Program



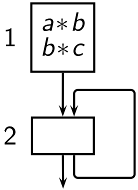
An Instance of Available Expressions Analysis

Lattice



Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
$f_\top$	$\{a*b, b*c\}$	f_{id}	x
$f_\perp$	$\emptyset$	f_c	$x \cup \{a*b\}$
f_a	$\{a*b\}$	f_d	$x \cup \{b*c\}$
f_b	$\{b*c\}$	f_e	$x - \{a*b\}$
		f_f	$x - \{b*c\}$

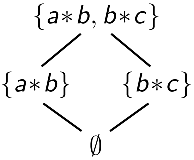
Program



Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

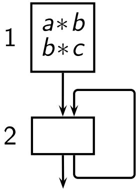
An Instance of Available Expressions Analysis

Lattice



Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
$f_\top$	$\{a*b, b*c\}$	f_{id}	x
$f_\bot$	$\emptyset$	f_c	$x \cup \{a*b\}$
f_a	$\{a*b\}$	f_d	$x \cup \{b*c\}$
f_b	$\{b*c\}$	f_e	$x - \{a*b\}$
		f_f	$x - \{b*c\}$

Program

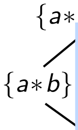


Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

Some Possible Assignments						
	A_1	A_2	A_3	A_4	A_5	A_6
In_1	00	00	00	00	00	00
Out_1	11	00	11	11	11	11
In_2	11	00	00	10	01	01
Out_2	11	00	00	10	01	10

An Instance of Available Expressions Analysis

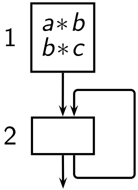
Lattice



Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
$f_\top$	$\{a*b, b*c\}$	f_{id}	x
			$x \cup \{a*b\}$
			$x \cup \{b*c\}$
			$x - \{a*b\}$
			$x - \{b*c\}$

- Maximum fixed point assignment
- Initialization for round robin iterative method: 11
- Safe assignment

Program

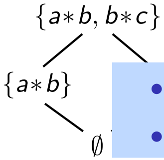


Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

Some Possible Assignments						
	A_1	A_2	A_3	A_4	A_5	A_6
In_1	00	00	00	00	00	00
Out_1	11	00	11	11	11	11
In_2	11	00	00	10	01	01
Out_2	11	00	00	10	01	10

An Instance of Available Expressions Analysis

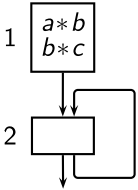
Lattice



Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
$f_\top$	$\{a*b, b*c\}$	f_{id}	x
$f_\bot$	$\emptyset$	f_c	$x \cup \{a*b\}$
		f_d	$x \cup \{b*c\}$
		f_e	$x - \{a*b\}$
		f_f	$x - \{b*c\}$

- Not a fixed point assignment
- Safe assignment

Program

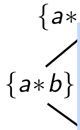


Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

Some Possible Assignments						
	A_1	A_2	A_3	A_4	A_5	A_6
In_1	00	00	00	00	00	00
Out_1	11	00	11	11	11	11
In_2	11	00	00	10	01	01
Out_2	11	00	00	10	01	10

An Instance of Available Expressions Analysis

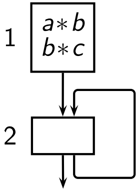
Lattice



Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
$f_\top$	$\{a*b, b*c\}$	f_{id}	x
			$x \cup \{a*b\}$
			$x \cup \{b*c\}$
			$x - \{a*b\}$
			$x - \{b*c\}$

- Minimum fixed point assignment
- Initialization for round robin iterative method: 00
- Safe assignment

Program



Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

Some Possible Assignments						
	A_1	A_2	A_3	A_4	A_5	A_6
In_1	00	00	00	00	00	00
Out_1	11	00	11	11	11	11
In_2	11	00	00	10	01	01
Out_2	11	00	00	10	01	10

An Instance of Available Expressions Analysis

Lattice

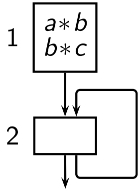
$\{a*$

$\{a*b\}$

Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
			x
			$x \cup \{a*b\}$
			$x \cup \{b*c\}$
			$x - \{a*b\}$
			$x - \{b*c\}$

- Fixed point assignment which is neither maximum nor minimum
- Initialization for round robin iterative method: 10
- Safe assignment

Program

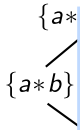


Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

Some Possible Assignments						
	A_1	A_2	A_3	A_4	A_5	A_6
In_1	00	00	00	00	00	00
Out_1	11	00	11	11	11	11
In_2	11	00	00	10	01	01
Out_2	11	00	00	10	01	10

An Instance of Available Expressions Analysis

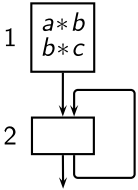
Lattice



Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
			x
			$x \cup \{a*b\}$
			$x \cup \{b*c\}$
			$x - \{a*b\}$
			$x - \{b*c\}$

- Fixed point assignment which is neither maximum nor minimum
- Initialization for round robin iterative method: 01
- Safe assignment

Program

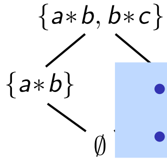


Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

Some Possible Assignments						
	A_1	A_2	A_3	A_4	A_5	A_6
In_1	00	00	00	00	00	00
Out_1	11	00	11	11	11	11
In_2	11	00	00	10	01	01
Out_2	11	00	00	10	01	10

An Instance of Available Expressions Analysis

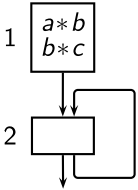
Lattice



Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
$f_\top$	$\{a*b, b*c\}$	f_{id}	x
$f_\bot$	$\emptyset$	f_c	$x \cup \{a*b\}$
		f_d	$x \cup \{b*c\}$
		f_e	$x - \{a*b\}$
		f_f	$x - \{b*c\}$

- Not a fixed point assignment
- Safe assignment

Program

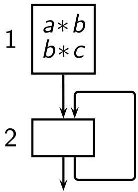


Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

Some Possible Assignments						
	A_1	A_2	A_3	A_4	A_5	A_6
In_1	00	00	00	00	00	00
Out_1	11	00	11	11	11	11
In_2	11	00	00	10	01	01
Out_2	11	00	00	10	01	10

Lattice of Assignments for Available Expressions Analysis

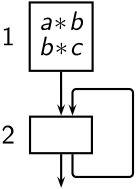
Program



Some Assignments							
	A_0	A_1	A_2	A_3	A_4	A_5	A_6
In_1	11	00	00	00	00	00	00
Out_1	11	11	00	11	11	11	11
In_2	11	11	00	00	10	01	01
Out_2	11	11	00	00	10	01	10

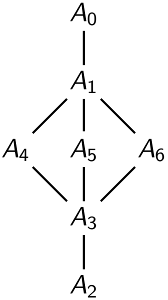
Lattice of Assignments for Available Expressions Analysis

Program



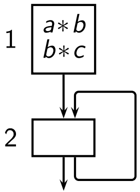
Some Assignments							
	A_0	A_1	A_2	A_3	A_4	A_5	A_6
In_1	11	00	00	00	00	00	00
Out_1	11	11	00	11	11	11	11
In_2	11	11	00	00	10	01	01
Out_2	11	11	00	00	10	01	10

Lattice $L \times L \times L \times L$ for all assignments (many assignments omitted, e.g. node 1 could have data flow values 10 and 01)



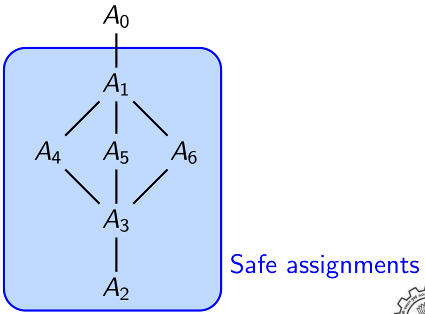
Lattice of Assignments for Available Expressions Analysis

Program



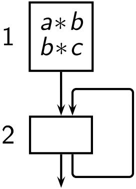
Some Assignments							
	A_0	A_1	A_2	A_3	A_4	A_5	A_6
In_1	11	00	00	00	00	00	00
Out_1	11	11	00	11	11	11	11
In_2	11	11	00	00	10	01	01
Out_2	11	11	00	00	10	01	10

Lattice $L \times L \times L \times L$ for all assignments (many assignments omitted, e.g. node 1 could have data flow values 10 and 01)



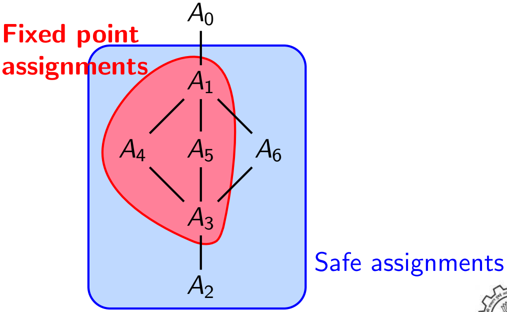
Lattice of Assignments for Available Expressions Analysis

Program



Some Assignments							
	A_0	A_1	A_2	A_3	A_4	A_5	A_6
In_1	11	00	00	00	00	00	00
Out_1	11	11	00	11	11	11	11
In_2	11	11	00	00	10	01	01
Out_2	11	11	00	00	10	01	10

Lattice $L \times L \times L \times L$ for all assignments (many assignments omitted, e.g. node 1 could have data flow values 10 and 01)



Existence of an MoP Assignment (1)

$$MoP(p) \quad = \prod_{\rho \,\in\, Paths(p)} f_\rho(BI)$$

- If a finite number of paths reach p, then existence of solution trivially follows
  - Function space is closed under composition
  - glb exists for all non-empty finite subsets of the lattice
    (Assuming that the data flow values form a meet semilattice)

Existence of an MoP Assignment (2)

$$MoP(p) \quad = \bigsqcap_{\rho \in Paths(p)} f_\rho(BI)$$

- If an infinite number of paths reach p then,

$$MoP(p) = f_{\rho_1}(BI) \sqcap f_{\rho_2}(BI) \sqcap f_{\rho_3}(BI) \sqcap \ldots$$

Existence of an MoP Assignment (2)

$$MoP(p) \quad = \bigsqcap_{\rho \in Paths(p)} f_\rho(BI)$$

- If an infinite number of paths reach p then,

$$MoP(p) = \underbrace{f_{\rho_1}(BI)}_{X_1} \sqcap f_{\rho_2}(BI) \sqcap f_{\rho_3}(BI) \sqcap \ldots$$

Existence of an MoP Assignment (2)

$$MoP(p) \quad = \prod_{\rho \in Paths(p)} f_\rho(BI)$$

- If an infinite number of paths reach p then,

$$MoP(p) = \underbrace{\underbrace{f_{\rho_1}(BI)}_{X_1} \sqcap f_{\rho_2}(BI)}_{X_2} \sqcap f_{\rho_3}(BI) \sqcap \ldots$$

- Every meet results in a weaker value

Existence of an MoP Assignment (2)

$$MoP(p) \quad = \prod_{\rho \,\in\, Paths(p)} f_\rho(BI)$$

- If an infinite number of paths reach p then,

$$MoP(p) = \underbrace{\underbrace{\underbrace{f_{\rho_1}(BI)}_{X_1} \sqcap f_{\rho_2}(BI)}_{X_2} \sqcap f_{\rho_3}(BI)}_{X_3} \sqcap \ldots$$

- Every meet results in a weaker value

Existence of an MoP Assignment (2)

$$MoP(p) \quad = \bigsqcap_{\rho \in Paths(p)} f_\rho(BI)$$

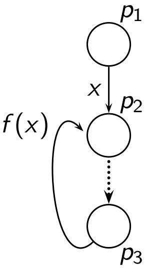
- If an infinite number of paths reach p then,

$$MoP(p) = \underbrace{\underbrace{\underbrace{f_{\rho_1}(BI)}_{X_1} \sqcap f_{\rho_2}(BI)}_{X_2} \sqcap f_{\rho_3}(BI)}_{X_3} \sqcap \ldots$$

- Every meet results in a weaker value
- The sequence $X_1, X_2, X_3, \ldots$ follows a descending chain
- Since all strictly descending chains are finite, MoP exists
(Assuming that our meet semilattice satisfies DCC)

Computability of MoP

Does existence of MoP imply it is computable?



Paths reaching the entry of p_2	Data Flow Value
p_1, p_2	x
p_1, p_2, p_3, p_2	$f(x)$
$p_1, p_2, p_3, p_2, p_3, p_2$	$f(f(x)) = f^2(x)$
$p_1, p_2, p_3, p_2, p_3, p_2, p_3, p_2$	$f(f(f(x))) = f^3(x)$
$\ldots$	$\ldots$

$$MoP(p_2) = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap f^4(x) \sqcap \ldots$$

MoP Computation is Undecidable

There does not exist any algorithm that can compute MoP assignment for every possible instance of every possible monotone data flow framework

- Reducing MPCP (Modified Post's Correspondence Problem) to constant propagation
- MPCP is known to be undecidable
- If an algorithm exists for detecting all constants

  $\Rightarrow$ MPCP would be decidable
- Since MPCP is undecidable

  $\Rightarrow$ There does not exist an algorithm for detecting all constants

  $\Rightarrow$ Static analysis is undecidable

Post's Correspondence Problem (PCP)

- Given strings $u_i, v_i \in \Sigma^+$ for some alphabet Σ, and two k-tuples,

$$\begin{aligned} U &= (u_1, u_2, \ldots, u_k) \\ V &= (v_1, v_2, \ldots, v_k) \end{aligned}$$

Is there a sequence $i_1, i_2, \ldots, i_m$ of one or more integers such that

$$u_{i_1} u_{i_2} \ldots u_{i_m} = v_{i_1} v_{i_2} \ldots v_{i_m}$$

Post's Correspondence Problem (PCP)

- Given strings $u_i, v_i \in \Sigma^+$ for some alphabet Σ, and two k-tuples,

$$\begin{aligned} U &= (u_1, u_2, \ldots, u_k) \\ V &= (v_1, v_2, \ldots, v_k) \end{aligned}$$

Is there a sequence $i_1, i_2, \ldots, i_m$ of one or more integers such that

$$u_{i_1} u_{i_2} \ldots u_{i_m} = v_{i_1} v_{i_2} \ldots v_{i_m}$$

- For $U = (101, 11, 100)$ and $V = (01, 1, 11001)$ the solution is $2, 3, 2$

$$\begin{aligned} u_2 u_3 u_2 &= 1110011 \\ v_2 v_3 v_2 &= 1110011 \end{aligned}$$

Post's Correspondence Problem (PCP)

- Given strings $u_i, v_i \in \Sigma^+$ for some alphabet Σ, and two k-tuples,

$$\begin{aligned} U &= (u_1, u_2, \ldots, u_k) \\ V &= (v_1, v_2, \ldots, v_k) \end{aligned}$$

Is there a sequence $i_1, i_2, \ldots, i_m$ of one or more integers such that

$$u_{i_1} u_{i_2} \ldots u_{i_m} = v_{i_1} v_{i_2} \ldots v_{i_m}$$

- For $U = (101, 11, 100)$ and $V = (01, 1, 11001)$ the solution is $2, 3, 2$

$$\begin{aligned} u_2 u_3 u_2 &= 1110011 \\ v_2 v_3 v_2 &= 1110011 \end{aligned}$$

- For $U = (1, 10111, 10)$, $V = (111, 10, 0)$, the solution is $2, 1, 1, 3$

Post's Correspondence Problem (PCP)

- Given strings $u_i, v_i \in \Sigma^+$ for some alphabet Σ, and two k-tuples,

$$\begin{aligned} U &= (u_1, u_2, \ldots, u_k) \\ V &= (v_1, v_2, \ldots, v_k) \end{aligned}$$

  Is there a sequence $i_1, i_2, \ldots, i_m$ of one or more integers such that

$$u_{i_1} u_{i_2} \ldots u_{i_m} = v_{i_1} v_{i_2} \ldots v_{i_m}$$

- For $U = (101, 11, 100)$ and $V = (01, 1, 11001)$ the solution is $2, 3, 2$

$$\begin{aligned} u_2 u_3 u_2 &= 1110011 \\ v_2 v_3 v_2 &= 1110011 \end{aligned}$$

- For $U = (1, 10111, 10)$, $V = (111, 10, 0)$, the solution is $2, 1, 1, 3$
- For $U = (01, 110)$, $V = (00, 11)$, there is no solution

Post's Correspondence Problem (PCP)

- Given strings $u_i, v_i \in \Sigma^+$ for some alphabet Σ, and two k-tuples,

$$\begin{aligned} U &= (u_1, u_2, \ldots, u_k) \\ V &= (v_1, v_2, \ldots, v_k) \end{aligned}$$

Is there a sequence $i_1, i_2, \ldots, i_m$ of one or more integers such that

$$u_{i_1} u_{i_2} \ldots u_{i_m} = v_{i_1} v_{i_2} \ldots v_{i_m}$$

- Sets U and V are finite and contain the same number of strings
- The strings in U and V are finite and are of varying lengths
- For constructing the new strings using the strings in U and V
  - The strings at the same the index of must be used
  - There is no limit on the length of the new string

Indices could repeat without any bound

Modified Post's Correspondence Problem (MPCP)

- The first string in the correspondence relation should be the first string from the k-tuple

$$u_1 u_{i_1} u_{i_2} \dots u_{i_m} = v_1 v_{i_1} v_{i_2} \dots v_{i_m}$$

Modified Post's Correspondence Problem (MPCP)

- The first string in the correspondence relation should be the first string from the k-tuple

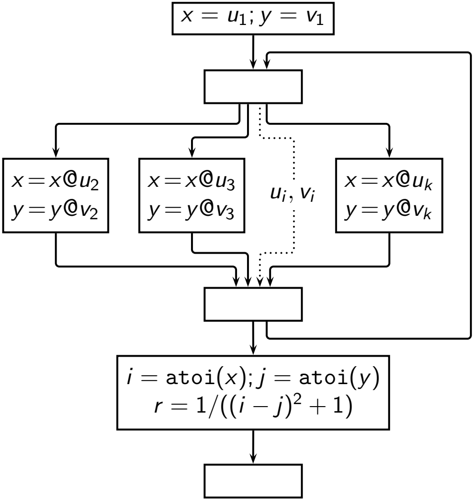
$$u_1 u_{i_1} u_{i_2} \ldots u_{i_m} = v_1 v_{i_1} v_{i_2} \ldots v_{i_m}$$

- For $U = (11, 1, 0111, 10)$, $V = (1, 111, 10, 0)$, the solution is $3, 2, 2, 4$

$$\begin{aligned} u_1 u_3 u_2 u_2 u_4 &= 1101111110 \\ v_1 v_3 v_2 v_2 v_4 &= 1101111110 \end{aligned}$$

Hecht's Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$

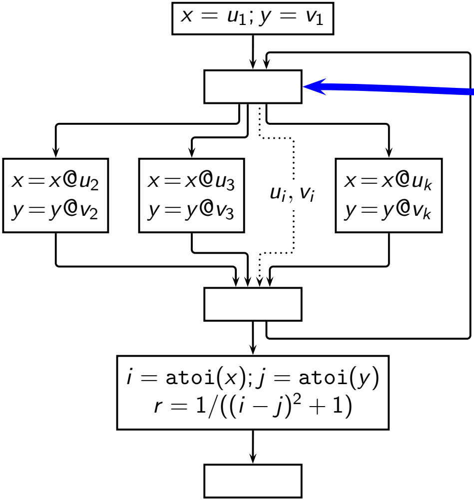


Hecht's Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$

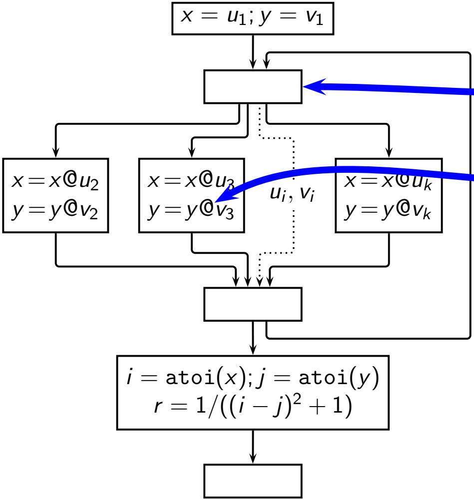
Each block in the loop corresponds to a particular index

Random branching for random selection of index



Hecht's Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$



Each block in the loop corresponds to a particular index

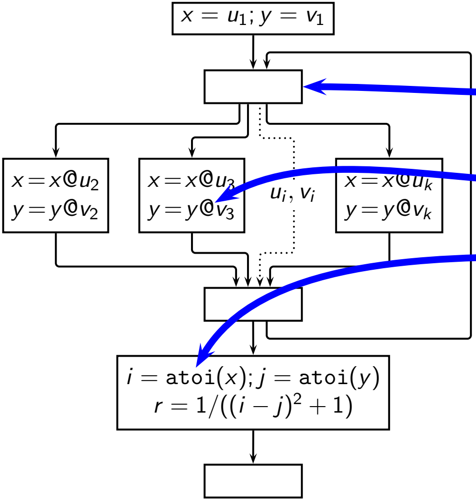
Random branching for random selection of index

String append

Hecht's Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$

Each block in the loop corresponds to a particular index



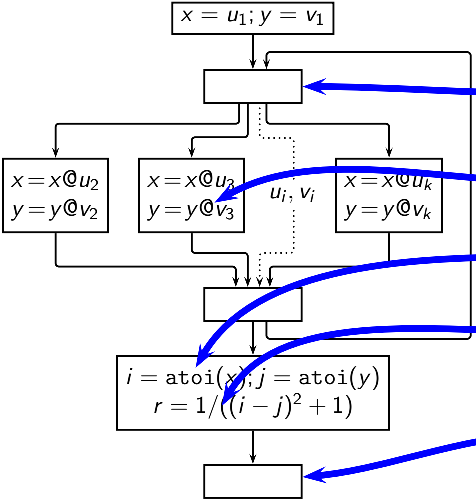
Random branching for random selection of index

String append

String to integer conversion

Hecht's Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$



Each block in the loop corresponds to a particular index

Random branching for random selection of index

String append

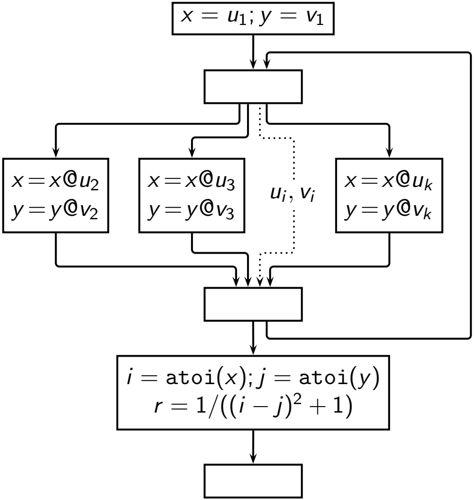
String to integer conversion

Integer division

MoP computation. No merge at intermediate points. Merge only at the point of interest

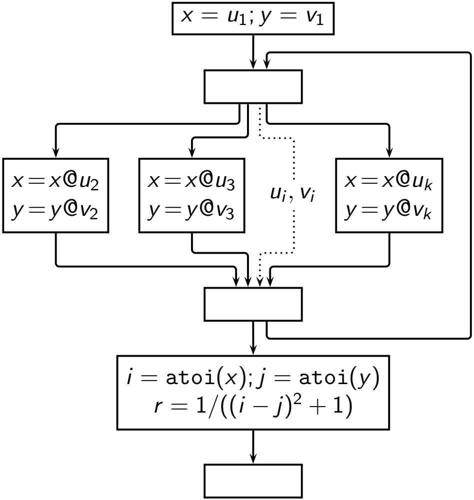
Hecht's Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$



Hecht's Reduction of MPCP to Constant Propagation

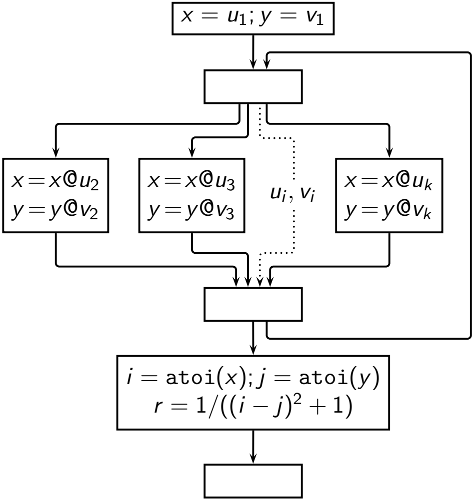
Given: An instance of MPCP with $\Sigma = \{0, 1\}$



- $i = j \Rightarrow r = 1$
  $i \neq j \Rightarrow r = 0$

Hecht's Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$



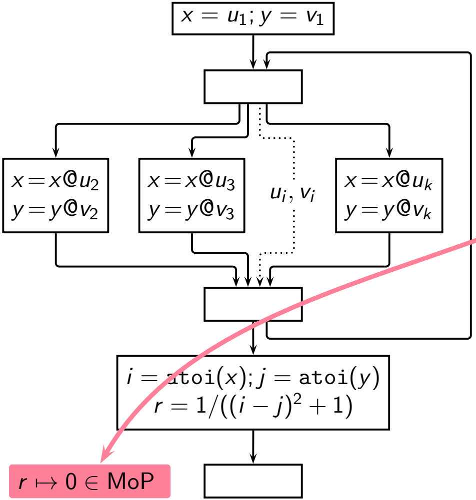
- $i = j \Rightarrow r = 1$
  $i \neq j \Rightarrow r = 0$
- **If** there exists an algorithm which can determine that

  $\{$

  $\}$

Hecht's Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$

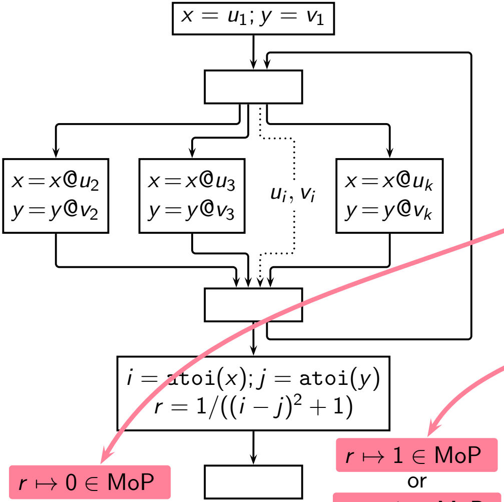


- $i = j \Rightarrow r = 1$
  $i \neq j \Rightarrow r = 0$
- **If** there exists an algorithm which can determine that
  {
  - $r = 0$ along every path
    (x is never equal to y, MPCP instance does not have a solution)

  }

Hecht's Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$



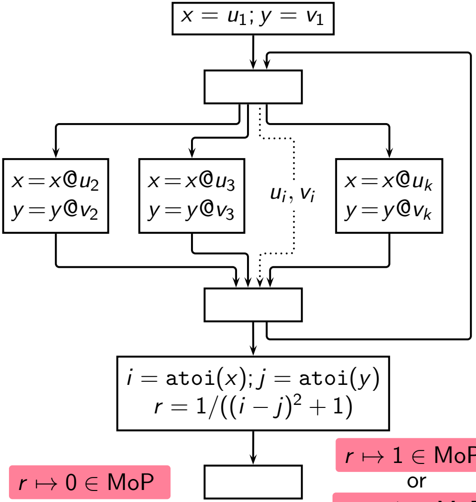
- $i = j \Rightarrow r = 1$
  $i \neq j \Rightarrow r = 0$
- **If** there exists an algorithm which can determine that
  {
  - $r = 0$ along every path (x is never equal to y, MPCP instance does not have a solution)
  - $r = 1$ along some path (some x is equal to y, MPCP instance has a solution)

  }

  Then MPCP is decidable

Hecht's Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$



$r \mapsto 0 \in$ MoP

$r \mapsto 1 \in$ MoP
or
$r \mapsto \perp \in$ MoP

The tricky part!!

- $i = j \Rightarrow r = 1$
  $i \neq j \Rightarrow r = 0$
- **If** there exists an algorithm which can determine that

  {
  - $r = 0$ along **every** path (x is never equal to y, MPCP instance does not have a solution)
  - $r = 1$ along **some** path (some x is equal to y, MPCP instance has a solution)

  }

Then MPCP is decidable

Hecht's Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$

- Asserting that no x is equal to y requires us to examine infinitely many (x, y) pairs
- If we keep finding x and y that are unequal, how long do we wait to decide that there is no x that is equal to y?
- In a lucky case we may find an x that is equal to y, but there is no guarantee

The tricky part!!

$i = j \Rightarrow r = 1$

$i \neq j \Rightarrow r = 0$

If there exists an algorithm which can determine that

{

- $r = 0$ along every path (x is never equal to y, MPCP instance does not have a solution)
- $r = 1$ along some path (some x is equal to y, MPCP instance has a solution)

}

Then MPCP is decidable

Hecht's Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$

- Asserting that no x is equal to y requires us to examine infinitely many (x, y) pairs
- If we keep finding x and y that are unequal, how long do we wait to decide that there is no x that is equal to y?
- In a lucky case we may find an x that is equal to y, but there is no guarantee

MPCP is not decidable
$\Rightarrow$ Constant Propagation is not decidable

The tricky part!!

$i = j \Rightarrow r = 1$
$i \neq j \Rightarrow r = 0$

If there exists an algorithm which can determine that

{
- $r = 0$ along every path (x is never equal to y, MPCP instance does not have a solution)
- $r = 1$ along some path (some x is equal to y, MPCP instance has a solution)

}

Then MPCP is decidable

Hecht's Reduction of MPCP to Constant Propagation

Given: An instance of MPCP with $\Sigma = \{0, 1\}$

- Asserting that no x is equal to y requires us to examine infinitely many (x, y) pairs
- If we keep finding x and y that are unequal, how long do we wait to decide that there is no x that is equal to y?
- In a lucky case we may find an x that is equal to y, but there is no guarantee

MPCP is not decidable
$\Rightarrow$ *Constant Propagation is not decidable*

- Descending chains consist of sets of pairs (x, y) with $\top$ as $\emptyset$

  Since there is no bound on the length of x and y, the number of these sets is infinite

  $\Rightarrow$ DCC is violated

The tricky part!!

$i = j \Rightarrow r = 1$

$i \neq j \Rightarrow r = 0$

If there exists an algorithm which can determine that

{

- $r = 0$ along every path (x is never equal to y, MPCP instance does not have a solution)
- $r = 1$ along some path (some x is equal to y, MPCP instance has a solution)

}

Then MPCP is decidable

Is MFP Always Computable?

MFP assignment may not be computable

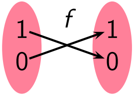
- if the flow functions are non-monotonic, or
- if some strictly descending chain is not finite

Computability of MFP

- If f is not monotonic, the computation may not converge

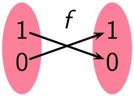
Computability of MFP

- If f is not monotonic, the computation may not converge



Computability of MFP

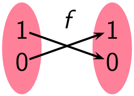
- If f is not monotonic, the computation may not converge



x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\ldots$
1	0	1	0	1	$\ldots$

Computability of MFP

- If f is not monotonic, the computation may not converge

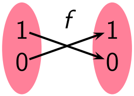


x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\ldots$
1	0	1	0	1	$\ldots$

$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots \; = \; 0$$

Computability of MFP

- If f is not monotonic, the computation may not converge



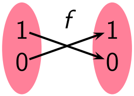
x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\ldots$
1	0	1	0	1	$\ldots$

$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots \ = \ 0$$

- Computing MFP iteratively

Computability of MFP

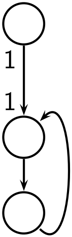
- If f is not monotonic, the computation may not converge



x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\ldots$
1	0	1	0	1	$\ldots$

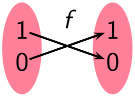
$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots \;=\; 0$$

- Computing MFP iteratively



Computability of MFP

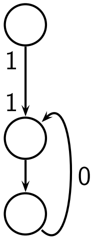
- If f is not monotonic, the computation may not converge



x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\ldots$
1	0	1	0	1	$\ldots$

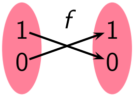
$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots = 0$$

- Computing MFP iteratively



Computability of MFP

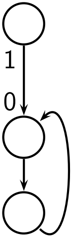
- If f is not monotonic, the computation may not converge



x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\dots$
1	0	1	0	1	$\dots$

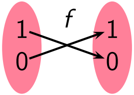
$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \dots = 0$$

- Computing MFP iteratively



Computability of MFP

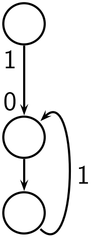
- If f is not monotonic, the computation may not converge



x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\ldots$
1	0	1	0	1	$\ldots$

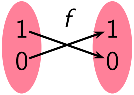
$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots \;=\; 0$$

- Computing MFP iteratively



Computability of MFP

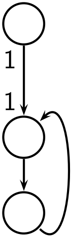
- If f is not monotonic, the computation may not converge



x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\dots$
1	0	1	0	1	$\dots$

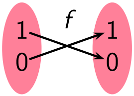
$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \dots \;=\; 0$$

- Computing MFP iteratively



Computability of MFP

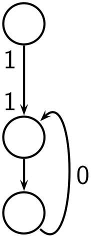
- If f is not monotonic, the computation may not converge



x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\ldots$
1	0	1	0	1	$\ldots$

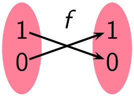
$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots \; = \; 0$$

- Computing MFP iteratively



Computability of MFP

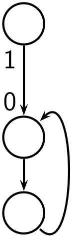
- If f is not monotonic, the computation may not converge



x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\dots$
1	0	1	0	1	$\dots$

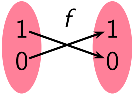
$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \dots \;=\; 0$$

- Computing MFP iteratively



Computability of MFP

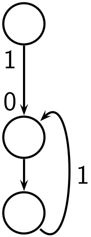
- If f is not monotonic, the computation may not converge



x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\dots$
1	0	1	0	1	$\dots$

$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \dots \; = \; 0$$

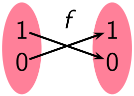
- Computing MFP iteratively



MFP does not exist and is not computable

Computability of MFP

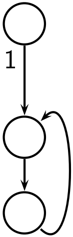
- If f is not monotonic, the computation may not converge



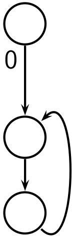
x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\dots$
1	0	1	0	1	$\dots$

$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \dots \; = \; 0$$

- Computing MFP iteratively

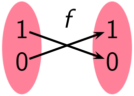


MFP does not exist and is not computable



Computability of MFP

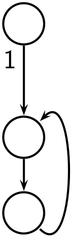
- If f is not monotonic, the computation may not converge



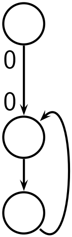
x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\dots$
1	0	1	0	1	$\dots$

$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \dots \; = \; 0$$

- Computing MFP iteratively

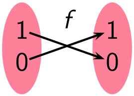


MFP does not exist and is not computable



Computability of MFP

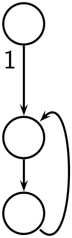
- If f is not monotonic, the computation may not converge



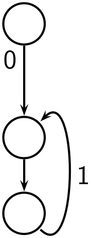
x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\ldots$
1	0	1	0	1	$\ldots$

$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots \; = \; 0$$

- Computing MFP iteratively

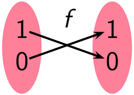


MFP does not exist and is not computable



Computability of MFP

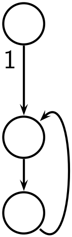
- If f is not monotonic, the computation may not converge



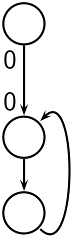
x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\dots$
1	0	1	0	1	$\dots$

$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \dots \; = \; 0$$

- Computing MFP iteratively

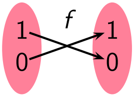


MFP does not exist and is not computable



Computability of MFP

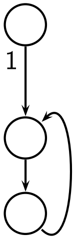
- If f is not monotonic, the computation may not converge



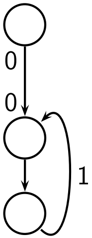
x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\ldots$
1	0	1	0	1	$\ldots$

$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots \ = \ 0$$

- Computing MFP iteratively

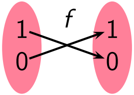


MFP does not exist and is not computable



Computability of MFP

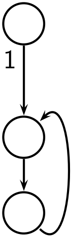
- If f is not monotonic, the computation may not converge



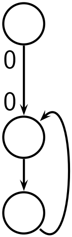
x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\ldots$
1	0	1	0	1	$\ldots$

$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots \; = \; 0$$

- Computing MFP iteratively

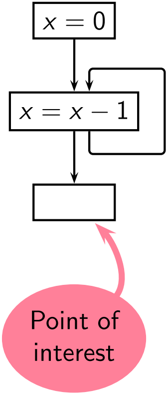


MFP does not exist and is not computable



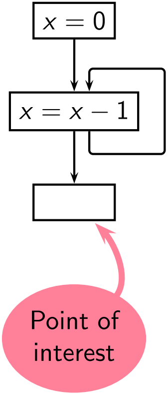
MFP exist and is computable

Computability of MFP



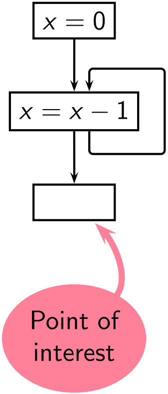
$\sqsubseteq$	$\leq$	$\leq$
$\sqcap$	min	min
Hasse diagram	0 | -1 | -2 | -3 | . . .	0 | -1 | -2 | -3 | . . . | $-\infty$
MFP exists?		
MFP computable?		
MoP exists?		

Computability of MFP



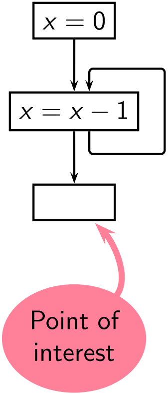
$\sqsubseteq$	$\leq$	$\leq$
$\sqcap$	min	min
Hasse diagram	0 | -1 | -2 | -3 | ...	0 | -1 | -2 | -3 | ... | $-\infty$
MFP exists?	No	
MFP computable?	No	
MoP exists?	No	

Computability of MFP



$\sqsubseteq$	$\leq$	$\leq$
$\sqcap$	min	min
Hasse diagram	0 -1 -2 -3 . . .	0 -1 -2 -3 . . . $-\infty$
MFP exists?	No	Yes
MFP computable?	No	No
MoP exists?	No	Yes

Computability of MFP



$\sqsubseteq$	$\leq$	$\leq$
$\sqcap$	min	min
	0 | -1 | . . .	0 | -1 | . . . | $-\infty$
MFP exists?	No	Yes
MFP computable?	No	No
MoP exists?	No	Yes

- Flow functions are monotonic
- Strictly descending chains are not finite

Existence and Computation of the Maximum Fixed Point

If L is a meet semilattice satisfying DCC, $f : L \rightarrow L$ is monotonic, then
$MFP(f) = f^{k+1}(\top) = f^{k}(\top)$ such that $f^{j+1}(\top) \neq f^{j}(\top)$, $j < k$

Existence and Computation of the Maximum Fixed Point

If L is a meet semilattice satisfying DCC, $f : L \to L$ is monotonic, then $MFP(f) = f^{k+1}(\top) = f^{k}(\top)$ such that $f^{j+1}(\top) \neq f^{j}(\top)$, $j < k$

Claims being made:

- $\exists k \; s.t. \; f^{k+1}(\top) = f^{k}(\top)$
- Since k is finite, $f^{k}(\top)$ exists and is computable
- $f^{k}(\top)$ is a fixed point
- $f^{k}(\top)$ is a the *maximum* fixed point

Existence and Computation of the Maximum Fixed Point

If L is a meet semilattice satisfying DCC, $f : L \rightarrow L$ is monotonic, then $MFP(f) = f^{k+1}(\top) = f^{k}(\top)$ such that $f^{j+1}(\top) \neq f^{j}(\top)$, $j < k$

Claims being made:

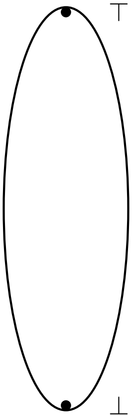
- $\exists k \; s.t. \; f^{k+1}(\top) = f^{k}(\top)$
- Since k is finite, $f^{k}(\top)$ exists and is computable
- $f^{k}(\top)$ is a fixed point
- $f^{k}(\top)$ is a the *maximum* fixed point

The proof depends on:

- The existence of glb for every pair of values in L
- Finiteness of strictly descending chains
- Monotonicity of f

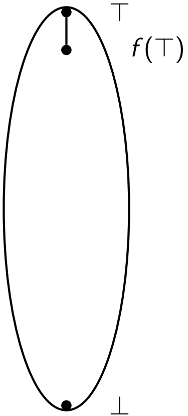
Existence and Computation of the Maximum Fixed Point

If L is a meet semilattice satisfying DCC, $f : L \rightarrow L$ is monotonic, then
$MFP(f) = f^{k+1}(\top) = f^{k}(\top)$ such that $f^{j+1}(\top) \neq f^{j}(\top)$, $j < k$



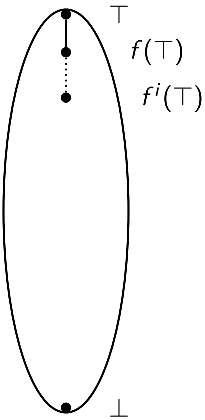
Existence and Computation of the Maximum Fixed Point

If L is a meet semilattice satisfying DCC, $f : L \to L$ is monotonic, then
$MFP(f) = f^{k+1}(\top) = f^k(\top)$ such that $f^{j+1}(\top) \neq f^j(\top)$, $j < k$



Existence and Computation of the Maximum Fixed Point

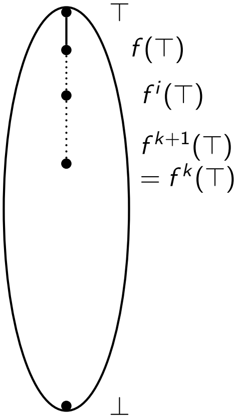
If L is a meet semilattice satisfying DCC, $f : L \to L$ is monotonic, then $MFP(f) = f^{k+1}(\top) = f^{k}(\top)$ such that $f^{j+1}(\top) \neq f^{j}(\top)$, $j < k$



- $\top \sqsupseteq f(\top) \sqsupseteq f^2(\top) \sqsupseteq f^3(\top) \sqsupseteq f^4(\top) \sqsupseteq \ldots$

Existence and Computation of the Maximum Fixed Point

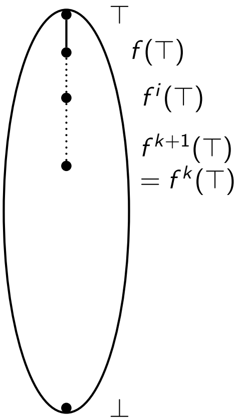
If L is a meet semilattice satisfying DCC, $f : L \rightarrow L$ is monotonic, then $MFP(f) = f^{k+1}(\top) = f^{k}(\top)$ such that $f^{j+1}(\top) \neq f^{j}(\top)$, $j < k$



- $\top \sqsupseteq f(\top) \sqsupseteq f^2(\top) \sqsupseteq f^3(\top) \sqsupseteq f^4(\top) \sqsupseteq \ldots$
- Since strictly descending chains are finite, there must exist $f^{k}(\top)$ such that $f^{k+1}(\top) = f^{k}(\top)$ and $f^{j+1}(\top) \neq f^{j}(\top)$, $j < k$

Existence and Computation of the Maximum Fixed Point

If L is a meet semilattice satisfying DCC, $f : L \rightarrow L$ is monotonic, then $MFP(f) = f^{k+1}(\top) = f^{k}(\top)$ such that $f^{j+1}(\top) \neq f^{j}(\top)$, $j < k$

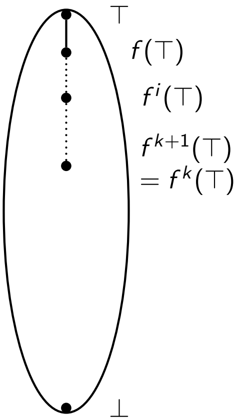


- $\top \sqsupseteq f(\top) \sqsupseteq f^2(\top) \sqsupseteq f^3(\top) \sqsupseteq f^4(\top) \sqsupseteq \dots$
- Since strictly descending chains are finite, there must exist $f^{k}(\top)$ such that $f^{k+1}(\top) = f^{k}(\top)$ and $f^{j+1}(\top) \neq f^{j}(\top)$, $j < k$
- If p is a fixed point of f then $p \sqsubseteq f^{k}(\top)$

  Proof strategy: Induction on i for $f^{i}(\top)$

Existence and Computation of the Maximum Fixed Point

If L is a meet semilattice satisfying DCC, $f : L \rightarrow L$ is monotonic, then $MFP(f) = f^{k+1}(\top) = f^k(\top)$ such that $f^{j+1}(\top) \neq f^j(\top),\ j < k$



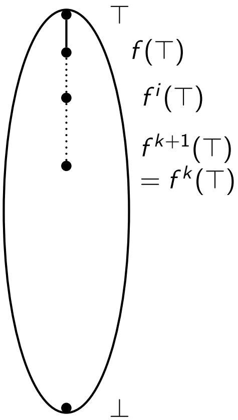
- $\top \sqsupseteq f(\top) \sqsupseteq f^2(\top) \sqsupseteq f^3(\top) \sqsupseteq f^4(\top) \sqsupseteq \dots$
- Since strictly descending chains are finite, there must exist $f^k(\top)$ such that $f^{k+1}(\top) = f^k(\top)$ and $f^{j+1}(\top) \neq f^j(\top),\ j < k$
- If p is a fixed point of f then $p \sqsubseteq f^k(\top)$

  Proof strategy: Induction on i for $f^i(\top)$

  - Basis ($i = 0$): $p \sqsubseteq f^0(\top) = \top$
  - Inductive Hypothesis: Assume that $p \sqsubseteq f^i(\top)$

Existence and Computation of the Maximum Fixed Point

If L is a meet semilattice satisfying DCC, $f : L \to L$ is monotonic, then $MFP(f) = f^{k+1}(\top) = f^k(\top)$ such that $f^{j+1}(\top) \neq f^j(\top),\ j < k$



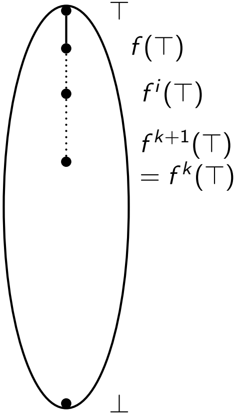
- $\top \sqsupseteq f(\top) \sqsupseteq f^2(\top) \sqsupseteq f^3(\top) \sqsupseteq f^4(\top) \sqsupseteq \ldots$
- Since strictly descending chains are finite, there must exist $f^k(\top)$ such that $f^{k+1}(\top) = f^k(\top)$ and $f^{j+1}(\top) \neq f^j(\top),\ j < k$
- If p is a fixed point of f then $p \sqsubseteq f^k(\top)$

  Proof strategy: Induction on i for $f^i(\top)$

  - Basis ($i = 0$): $p \sqsubseteq f^0(\top) = \top$
  - Inductive Hypothesis: Assume that $p \sqsubseteq f^i(\top)$
  - Proof: $f(p) \sqsubseteq f(f^i(\top))$ (f is monotonic)
    $\Rightarrow\ p \sqsubseteq f(f^i(\top))$ ($f(p) = p$)
    $\Rightarrow\ p \sqsubseteq f^{i+1}(\top)$

Existence and Computation of the Maximum Fixed Point

If L is a meet semilattice satisfying DCC, $f : L \to L$ is monotonic, then $MFP(f) = f^{k+1}(\top) = f^k(\top)$ such that $f^{j+1}(\top) \neq f^j(\top)$, $j < k$



- $\top \sqsupseteq f(\top) \sqsupseteq f^2(\top) \sqsupseteq f^3(\top) \sqsupseteq f^4(\top) \sqsupseteq \dots$
- Since strictly descending chains are finite, there must exist $f^k(\top)$ such that $f^{k+1}(\top) = f^k(\top)$ and $f^{j+1}(\top) \neq f^j(\top)$, $j < k$
- If p is a fixed point of f then $p \sqsubseteq f^k(\top)$

  Proof strategy: Induction on i for $f^i(\top)$

  - Basis ($i = 0$): $p \sqsubseteq f^0(\top) = \top$
  - Inductive Hypothesis: Assume that $p \sqsubseteq f^i(\top)$
  - Proof: $$\begin{aligned} & f(p) \sqsubseteq f(f^i(\top)) \ (f \text{ is monotonic}) \\ \Rightarrow \quad & p \sqsubseteq f(f^i(\top)) \ (f(p) = p) \\ \Rightarrow \quad & p \sqsubseteq f^{i+1}(\top) \end{aligned}$$
- Since this holds for every p that is a fixed point, $f^{k+1}(\top)$ must be the Maximum Fixed Point

Fixed Points Computation: Flow Functions Vs. Equations

- Recall that

$$MFP(f) = f^{k+1}(\top) = f^{k}(\top) \text{ such that } f^{j+1}(\top) \neq f^{j}(\top),\ j < k.$$

Fixed Points Computation: Flow Functions Vs. Equations

- Recall that

$$MFP(f) = f^{k+1}(\top) = f^{k}(\top) \text{ such that } f^{j+1}(\top) \neq f^{j}(\top),\ j < k.$$

  - What is f in the above?

Fixed Points Computation: Flow Functions Vs. Equations

- Recall that

$$MFP(f) = f^{k+1}(\top) = f^{k}(\top) \text{ such that } f^{j+1}(\top) \neq f^{j}(\top),\ j < k.$$

  - What is f in the above?
  - Flow function of a block? Which block?

Fixed Points Computation: Flow Functions Vs. Equations

- Recall that

$$MFP(f) = f^{k+1}(\top) = f^{k}(\top) \text{ such that } f^{j+1}(\top) \neq f^{j}(\top),\ j < k.$$

  - What is f in the above?
  - Flow function of a block? Which block?

- Our method computes the maximum fixed point of data flow equations!

Fixed Points Computation: Flow Functions Vs. Equations

- Recall that

$$MFP(f) = f^{k+1}(\top) = f^{k}(\top) \text{ such that } f^{j+1}(\top) \neq f^{j}(\top),\ j < k.$$

  - What is f in the above?
  - Flow function of a block? Which block?
- Our method computes the maximum fixed point of data flow equations!
- What is the relation between the maximum fixed point of data flow equations and the MFP defined above?

Fixed Points Computation: Flow Functions Vs. Equations

- Data flow equations for a CFG with N nodes can be written as

$$
\begin{aligned}
In_1 &= BI \\
Out_1 &= f_1(In_1) \\
In_2 &= Out_1 \sqcap \ldots \\
Out_2 &= f_2(In_2) \\
&\ldots \\
In_N &= Out_{N-1} \sqcap \ldots \\
Out_N &= f_N(In_N)
\end{aligned}
$$

Fixed Points Computation: Flow Functions Vs. Equations

- Data flow equations for a CFG with N nodes can be written as

$$
\begin{aligned}
In_1 &= f_{In_1}(\langle In_1, Out_1, \ldots, In_N, Out_N \rangle) \\
Out_1 &= f_{Out_1}(\langle In_1, Out_1, \ldots, In_N, Out_N \rangle) \\
In_2 &= f_{In_2}(\langle In_1, Out_1, \ldots, In_N, Out_N \rangle) \\
Out_2 &= f_{Out_2}(\langle In_1, Out_1, \ldots, In_N, Out_N \rangle) \\
& \ldots \\
In_N &= f_{In_N}(\langle In_1, Out_1, \ldots, In_N, Out_N \rangle) \\
Out_N &= f_{Out_N}(\langle In_1, Out_1, \ldots, In_N, Out_N \rangle)
\end{aligned}
$$

where each flow function is of the form $L \times L \times \ldots \times L \rightarrow L$

Fixed Points Computation: Flow Functions Vs. Equations

- Data flow equations for a CFG with N nodes can be written as

$$
\begin{array}{lcll}
\langle In_1, Out_1, \ldots, In_N, Out_N \rangle & = & \langle & f_{In_1}(\langle In_1, Out_1, \ldots, In_N, Out_N \rangle), \\
 & & & f_{Out_1}(\langle In_1, Out_1, \ldots, In_N, Out_N \rangle), \\
 & & & \ldots \\
 & & & f_{In_N}(\langle In_1, Out_1, \ldots, In_N, Out_N \rangle), \\
 & & & f_{Out_N}(\langle In_1, Out_1, \ldots, In_N, Out_N \rangle), \\
 & & \rangle &
\end{array}
$$

where each flow function is of the form $L \times L \times \ldots \times L \rightarrow L$

Fixed Points Computation: Flow Functions Vs. Equations

- Data flow equations for a CFG with N nodes can be written as

$$\begin{array}{rcll} \mathcal{X} & = & \langle & f_{In_1}(\mathcal{X}), \\ & & & f_{Out_1}(\mathcal{X}), \\ & & & \ldots \\ & & & f_{In_N}(\mathcal{X}), \\ & & & f_{Out_N}(\mathcal{X}), \\ & & \rangle & \end{array}$$

where $\mathcal{X} = \langle In_1, Out_1, \ldots, In_N, Out_N \rangle$

Fixed Points Computation: Flow Functions Vs. Equations

- Data flow equations for a CFG with N nodes can be written as

$$\mathcal{X} = \mathcal{F}(\mathcal{X})$$

where

$$\begin{aligned} \mathcal{X} &= \langle In_1, Out_1, \ldots, In_N, Out_N \rangle \\ \mathcal{F}(\mathcal{X}) &= \langle f_{In_1}(\mathcal{X}), f_{Out_1}(\mathcal{X}), \ldots, f_{In_N}(\mathcal{X}), f_{Out_N}(\mathcal{X}) \rangle \end{aligned}$$

Fixed Points Computation: Flow Functions Vs. Equations

- Data flow equations for a CFG with N nodes can be written as

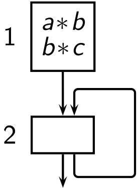
$$\mathcal{X} \quad = \quad \mathcal{F}(\mathcal{X})$$

$$\text{where} \qquad \begin{array}{rcl} \mathcal{X} & = & \langle In_1, Out_1, \ldots, In_N, Out_N \rangle \\ \mathcal{F}(\mathcal{X}) & = & \langle f_{In_1}(\mathcal{X}), f_{Out_1}(\mathcal{X}), \ldots, f_{In_N}(\mathcal{X}), f_{Out_N}(\mathcal{X}) \rangle \end{array}$$

We compute the fixed points of function $\mathcal{F}$ defined above

An Instance of Available Expressions Analysis

Program

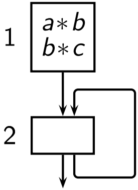


- Conventional data flow equations

$$
\begin{aligned}
In_1 &= 00 \\
Out_1 &= 11
\end{aligned}
$$

An Instance of Available Expressions Analysis

Program

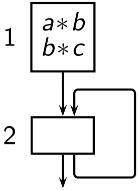


- Conventional data flow equations

$$
\begin{aligned}
In_1 &= 00 & \qquad In_2 &= Out_1 \cap Out_2 \\
Out_1 &= 11 & \qquad Out_2 &= In_2
\end{aligned}
$$

An Instance of Available Expressions Analysis

Program



- Conventional data flow equations

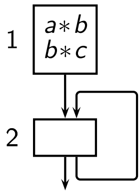
$$\begin{aligned} In_1 &= 00 & \qquad In_2 &= Out_1 \cap Out_2 \\ Out_1 &= 11 & \qquad Out_2 &= In_2 \end{aligned}$$

- Data Flow Equation $\mathcal{X} = \mathcal{F}(\mathcal{X})$ is

$$\mathcal{F}(\langle In_1, Out_1, In_2, Out_2 \rangle) = \langle 00, 11, Out_1 \cap Out_2, In_2 \rangle$$

An Instance of Available Expressions Analysis

Program



- Conventional data flow equations

$$In_1 = 00 \qquad In_2 = Out_1 \cap Out_2$$
$$Out_1 = 11 \qquad Out_2 = In_2$$

- Data Flow Equation $\mathcal{X} = \mathcal{F}(\mathcal{X})$ is

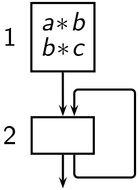
$$\mathcal{F}(\langle In_1, Out_1, In_2, Out_2\rangle) = \langle 00, 11, Out_1 \cap Out_2, In_2\rangle$$

- The maximum fixed point assignment is

$$\mathcal{F}(\langle 11, 11, 11, 11\rangle) = \langle 00, 11, 11, 11\rangle$$

An Instance of Available Expressions Analysis

Program



- Conventional data flow equations

$$\begin{aligned} In_1 &= 00 & \qquad In_2 &= Out_1 \cap Out_2 \\ Out_1 &= 11 & \qquad Out_2 &= In_2 \end{aligned}$$

- Data Flow Equation $\mathcal{X} = \mathcal{F}(\mathcal{X})$ is

$$\mathcal{F}(\langle In_1, Out_1, In_2, Out_2 \rangle) = \langle 00, 11, Out_1 \cap Out_2, In_2 \rangle$$

- The maximum fixed point assignment is

$$\mathcal{F}(\langle 11, 11, 11, 11 \rangle) = \langle 00, 11, 11, 11 \rangle$$

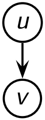
- The minimum fixed point assignment is

$$\mathcal{F}(\langle 00, 00, 00, 00 \rangle) = \langle 00, 11, 00, 00 \rangle$$

The Essential Difference Between MFP and MoP Values

- For all edges $u \rightarrow v$, $FP(v) \sqsubseteq f_{u\rightarrow v}(FP(u))$

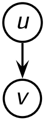
  because $FP(v) = \prod_{u \in pred(v)} f_{u\rightarrow v}(FP(u))$



The Essential Difference Between MFP and MoP Values

- For all edges $u \rightarrow v$, $FP(v) \sqsubseteq f_{u\rightarrow v}(FP(u))$

  because $FP(v) = \prod_{u \in pred(v)} f_{u\rightarrow v}(FP(u))$



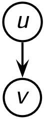
- Such a relationship does not exist for MoP

  because $MoP(v)$ is not computed from $MoP(u)$

The Essential Difference Between MFP and MoP Values

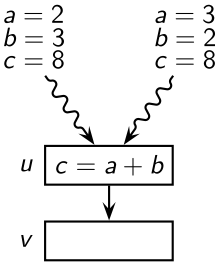
- For all edges $u \rightarrow v$, $FP(v) \sqsubseteq f_{u \rightarrow v}(FP(u))$

  because $FP(v) = \bigsqcap_{u \in pred(v)} f_{u \rightarrow v}(FP(u))$



- Such a relationship does not exist for MoP

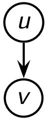
  because $MoP(v)$ is not computed from $MoP(u)$



The Essential Difference Between MFP and MoP Values

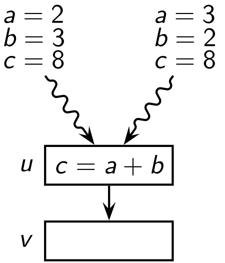
- For all edges $u \to v$, $FP(v) \sqsubseteq f_{u\to v}(FP(u))$

  because $FP(v) = \displaystyle\bigsqcap_{u \in pred(v)} f_{u\to v}(FP(u))$



- Such a relationship does not exist for MoP

  because $MoP(v)$ is not computed from $MoP(u)$



$$
\begin{aligned}
MoP(u) &= \langle \bot, \bot, 8 \rangle \\
f_{u\to v}(MoP(u)) &= \langle \bot, \bot, \bot \rangle \\
MoP(v) &= \langle \bot, \bot, 5 \rangle
\end{aligned}
$$

$$
\begin{aligned}
FP(u) &= \langle \bot, \bot, 8 \rangle \\
f_{u\to v}(FP(u)) &= \langle \bot, \bot, \bot \rangle \\
FP(v) &= \langle \bot, \bot, \bot \rangle
\end{aligned}
$$

The Essential Difference Between MFP and MoP Computation (1)

When we have a meet semilattice with DCC and monotonic flow functions

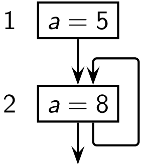
$$MoP(v) = \bigsqcap_{\rho_v \in Paths(v)} f_{\rho_v}(BI) \ = \ f_{\rho^0}(BI) \sqcap f_{\rho^1}(BI) \sqcap \ldots f_{\rho^i}(BI) \sqcap \ldots$$

$$MFP(v) = \bigsqcap_{u \in pred(v)} f_{u \rightarrow v}(MFP(u))$$

- $Mop(v)$ exists (because of DCC)
- $MoP(v)$ needs to iterate over all paths reaching v. For termination,
  - the meet across all paths upto some ρ_i should result in $\perp$ value, or
  - all paths reaching v should be exhausted
- $MFP(v)$ needs to iterate over the entire CFG repeatedly
  - In each iteration over the graph, it needs to iterate over all predecessors
  - Termination depends on finding two successively value that are same Guaranteed by DCC and monotonicity

The Essential Difference Between MFP and MoP Computation (2)

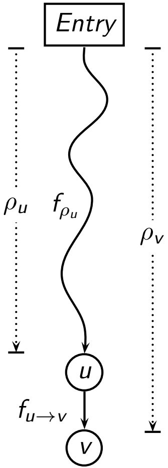
Consider a constant propagation example



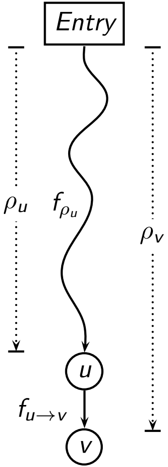
- An algorithm to compute $MoP(2)$ needs to consider the paths $(1),\ (1,2),\ (1,2,2),\ (1,2,2,2),\ \ldots$
- After how many paths should it terminate?

  Values being same across two successive paths cannot be the termination criterian for *MoP*

Soundness of FP Assignment: FP $\sqsubseteq$ MoP

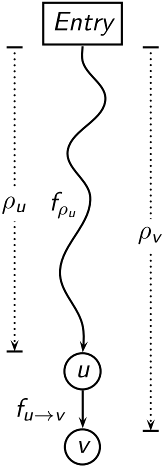


Soundness of FP Assignment: FP $\sqsubseteq$ MoP



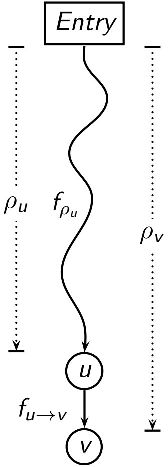
- $MoP(v) \quad = \bigsqcap_{\rho \in Paths(v)} f_\rho(BI)$

Soundness of FP Assignment: FP $\sqsubseteq$ MoP



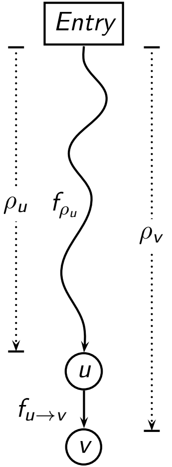
- $MoP(v) \;=\; \bigsqcap_{\rho \in Paths(v)} f_\rho(BI)$
- Proof Obligation: $\forall \rho_v \; FP(v) \sqsubseteq f_{\rho_v}(BI)$

Soundness of FP Assignment: FP $\sqsubseteq$ MoP



- $MoP(v) \quad = \bigsqcap_{\rho \in Paths(v)} f_\rho(BI)$
- Proof Obligation: $\forall \rho_v \; FP(v) \sqsubseteq f_{\rho_v}(BI)$
- Claim 1: $\forall u \to v, FP(v) \sqsubseteq f_{u\to v}(FP(u))$

Soundness of FP Assignment: FP $\sqsubseteq$ MoP



- $MoP(v) \quad = \bigsqcap\limits_{\rho \in Paths(v)} f_\rho(BI)$
- Proof Obligation: $\forall \rho_v \; FP(v) \sqsubseteq f_{\rho_v}(BI)$
- Claim 1: $\forall u \to v, FP(v) \sqsubseteq f_{u\to v}(FP(u))$
- Proof Outline: Induction on the length of the path

  Base case: Path of length 0

  $FP(Entry) = MoP(Entry) = BI$

  Inductive hypothesis: Assume it holds for paths consisting of k edges (say at u)

$$\begin{array}{lll} & FP(u) \sqsubseteq f_{\rho_u}(BI) & \text{(Inductive hypothesis)} \\ & FP(v) \sqsubseteq f_{u\to v}(FP(u)) & \text{(Claim 1)} \\ \Rightarrow & FP(v) \sqsubseteq f_{u\to v}(f_{\rho_u}(BI)) & \\ \Rightarrow & FP(v) \sqsubseteq f_{\rho_v}(BI) & \end{array}$$

This holds for every FP an hence for MFP also

Part 8

Theoretical Abstractions: A Summary

Theoretical Abstractions: A Summary

Necessary and sufficient conditions for designing a data flow framework

Theoretical Abstractions: A Summary

Necessary and sufficient conditions for designing a data flow framework

- A meet semilattice satisfying dcc

Theoretical Abstractions: A Summary

Necessary and sufficient conditions for designing a data flow framework

- A meet semilattice satisfying dcc
- A function space
  - Monotonic functions

Theoretical Abstractions: A Summary

Necessary and sufficient conditions for designing a data flow framework

- A meet semilattice satisfying dcc
  - Meet: commutative, associative, and idempotent
  - Partial order: reflexive, transitive, and antisymmetric
  - Existence of $\perp$
- A function space
  - Monotonic functions

Theoretical Abstractions: A Summary

Necessary and sufficient conditions for designing a data flow framework

- A meet semilattice satisfying dcc
  - Meet: commutative, associative, and idempotent
  - Partial order: reflexive, transitive, and antisymmetric
  - Existence of $\perp$
- A function space
  - Existence of the identity function
  - Closure under composition
  - Monotonic functions

Part 9

Performing Data Flow Analysis

Performing Data Flow Analysis

- Algorithms for computing MFP solution
- Complexity of data flow analysis
- Factor affecting the complexity of data flow analysis

Iterative Methods of Performing Data Flow Analysis

Successive recomputation after conservative initialization ($\top$)

- *Round Robin*. Repeated traversals over nodes in a fixed order

  Termination : After values stabilise

  + Simplest to understand and implement
  − May perform unnecessary computations

Iterative Methods of Performing Data Flow Analysis

Successive recomputation after conservative initialization ($\top$)

- *Round Robin*. Repeated traversals over nodes in a fixed order

  Termination : After values stabilise

  + Simplest to understand and implement
  − May perform unnecessary computations

Our examples use this method

Iterative Methods of Performing Data Flow Analysis

Successive recomputation after conservative initialization ($\top$)

- *Round Robin*. Repeated traversals over nodes in a fixed order

  Termination : After values stabilise

  + Simplest to understand and implement
  − May perform unnecessary computations

Our examples use this method

- *Work List*. Dynamic list of nodes which need recomputation

  Termination : When the list becomes empty

  + Demand driven. Avoid unnecessary computations
  − Overheads of maintaining work list

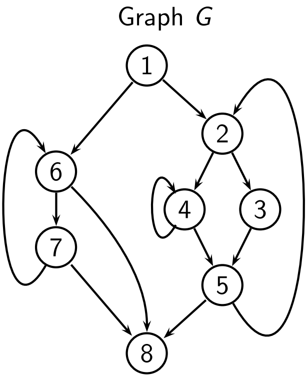
Elimination Methods of Performing Data Flow Analysis

Delayed computations of dependent data flow values of dependent nodes

Find suitable single-entry regions

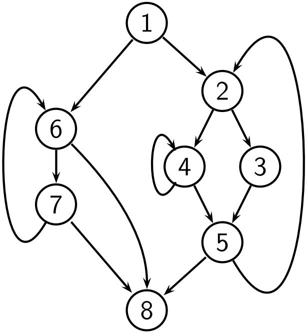
- *Interval Based Analysis*. Uses graph partitioning
- T_1, T_2 *Based Analysis*. Uses graph parsing

Classification of Edges in a Graph

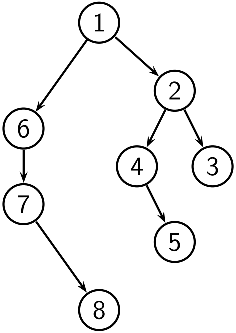


Classification of Edges in a Graph

Graph G

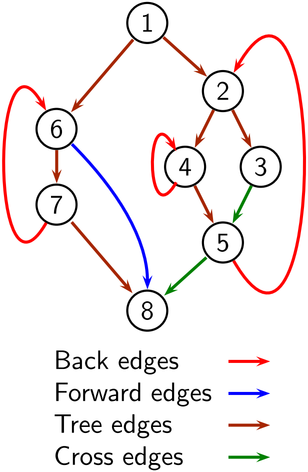


A depth first spanning tree of G

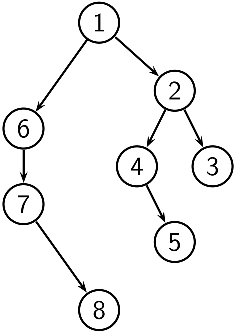


Classification of Edges in a Graph

Graph G

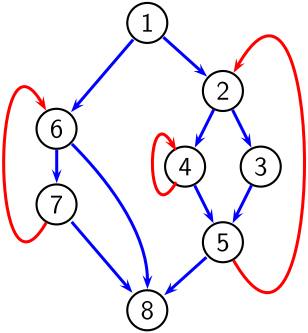


A depth first spanning tree of G



Classification of Edges in a Graph

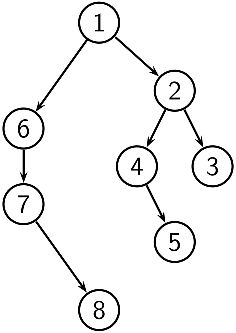
Graph G



Back edges

Forward edges

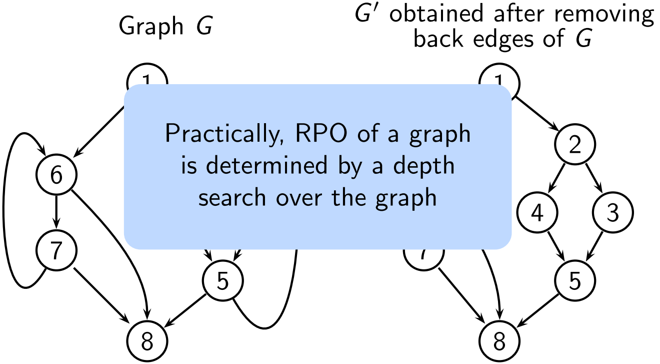
A depth first spanning tree of G



For data flow analysis, we club *tree*, *forward*, and *cross* edges into *forward* edges. Thus we have just forward or back edges in a control flow graph

Reverse Post Order Traversal

- A reverse post order (rpo) is a topological sort of the graph obtained after removing back edges



- Some possible RPOs for G are: $(1, 2, 3, 4, 5, 6, 7, 8)$, $(1, 6, 7, 2, 3, 4, 5, 8)$, $(1, 6, 2, 7, 4, 3, 5, 8)$, and $(1, 2, 6, 7, 3, 4, 5, 8)$

Round Robin Iterative Algorithm

```
In_0 = BI
for all j ≠ 0 do
    In_j = ⊤
change = true
while change do
{   change = false
    for j = 1 to N − 1 do
    {   temp = ⨅_{p ∈ pred(j)} f_p(In_p)
        if temp ≠ In_j then
        {   In_j = temp
            change = true
        }
    }
}
```

Round Robin Iterative Algorithm

$In_0 = BI$
for all $j \neq 0$ **do**
    $In_j = \top$
$change = true$
while $change$ **do**
$\{$ $change = false$
    for $j = 1$ to $N - 1$ **do**
    $\{$ $temp = \prod_{p \in pred(j)} f_p(In_p)$
        if $temp \neq In_j$ **then**
        $\{$ $In_j = temp$
            $change = true$
        $\}$
    $\}$
$\}$

- Computation of Out_j has been left implicit

  Works fine for unidirectional frameworks

Round Robin Iterative Algorithm

1 $In_0 = BI$
2 **for** all $j \neq 0$ **do**
3 $\quad In_j = \top$
4 $change = true$
5 **while** $change$ **do**
6 $\{ \quad change = false$
7 $\quad$ **for** $j = 1$ to $N-1$ **do**
8 $\quad \{ \quad temp = \displaystyle\bigsqcap_{p \in pred(j)} f_p(In_p)$
9 $\quad\quad$ **if** $temp \neq In_j$ **then**
10 $\quad\quad \{ \quad In_j = temp$
11 $\quad\quad\quad change = true$
12 $\quad\quad \}$
13 $\quad \}$
14 $\}$

- Computation of Out_j has been left implicit

  Works fine for unidirectional frameworks

- $\top$ is the identity of $\sqcap$ (line 3)

Round Robin Iterative Algorithm

1 $In_0 = BI$
2 **for** all $j \neq 0$ **do**
3     $In_j = \top$
4 $change = true$
5 **while** $change$ **do**
6 { $change = false$
7   **for** $j = 1$ to $N - 1$ **do**
8   { $temp = \displaystyle\prod_{p \in pred(j)} f_p(In_p)$
9   **if** $temp \neq In_j$ **then**
10   { $In_j = temp$
11             $change = true$
12   }
13   }
14 }

- Computation of Out_j has been left implicit

  Works fine for unidirectional frameworks
- $\top$ is the identity of $\sqcap$ (line 3)
- Reverse postorder (rpo) traversal for efficiency (line 7)

Round Robin Iterative Algorithm

```
1   In_0 = BI
2   for all j ≠ 0 do
3       In_j = ⊤
4   change = true
5   while change do
6   {   change = false
7       for j = 1 to N − 1 do
8       {   temp = ⊓_{p ∈ pred(j)} f_p(In_p)
9           if temp ≠ In_j then
10          {   In_j = temp
11              change = true
12          }
13      }
14  }
```

- Computation of Out_j has been left implicit

  Works fine for unidirectional frameworks
- $\top$ is the identity of $\sqcap$ (line 3)
- Reverse postorder (rpo) traversal for efficiency (line 7)
- rpo traversal AND no loops

  $\Rightarrow$ no need of initialization

Complexity of Round Robin Iterative Algorithm

- Unidirectional bit vector frameworks
  - Construct a spanning tree T of G to identify postorder traversal
  - Traverse G in reverse postorder for forward problems and
    Traverse G in postorder for backward problems
  - Depth $d(G, T)$: Maximum number of back edges in any acyclic path

Task	Number of iterations
First computation of *In* and *Out*	1
Convergence (until *change* remains true)	$d(G, T)$
Verifying convergence (*change* becomes false)	1

Complexity of Round Robin Iterative Algorithm

- Unidirectional bit vector frameworks
  - Construct a spanning tree T of G to identify postorder traversal
  - Traverse G in reverse postorder for forward problems and Traverse G in postorder for backward problems
  - Depth $d(G,T)$: Maximum number of back edges in any acyclic path

Task	Number of iterations
First computation of *In* and *Out*	1
Convergence (until *change* remains true)	$d(G,T)$
Verifying convergence (*change* becomes false)	1

- What about bidirectional bit vector frameworks?

Complexity of Round Robin Iterative Algorithm

- Unidirectional bit vector frameworks
  - Construct a spanning tree T of G to identify postorder traversal
  - Traverse G in reverse postorder for forward problems and Traverse G in postorder for backward problems
  - Depth $d(G, T)$: Maximum number of back edges in any acyclic path

Task	Number of iterations
First computation of *In* and *Out*	1
Convergence (until *change* remains true)	$d(G, T)$
Verifying convergence (*change* becomes false)	1

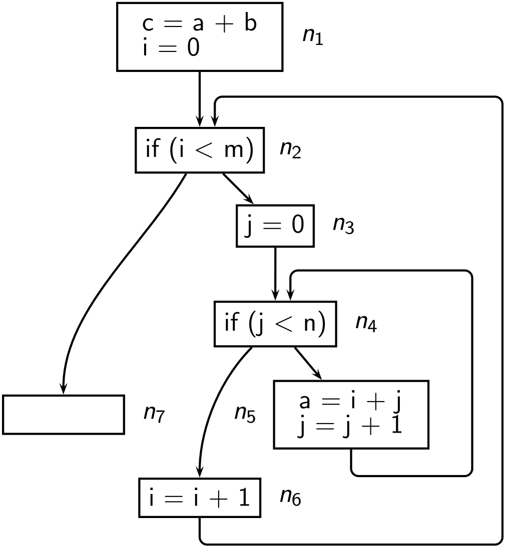
- What about bidirectional bit vector frameworks?
- What about other frameworks?

Example C Program with d(G,T) = 2

```
void fun(int m, int n)
{
   int i,j,a,b,c;
   c=a+b;
   i=0;
   while(i<m)
   {
         j=0;
         while(j<n)
         {
            a=i+j;
            j=j+1;
         }
         i=i+1;
      }
   }
```

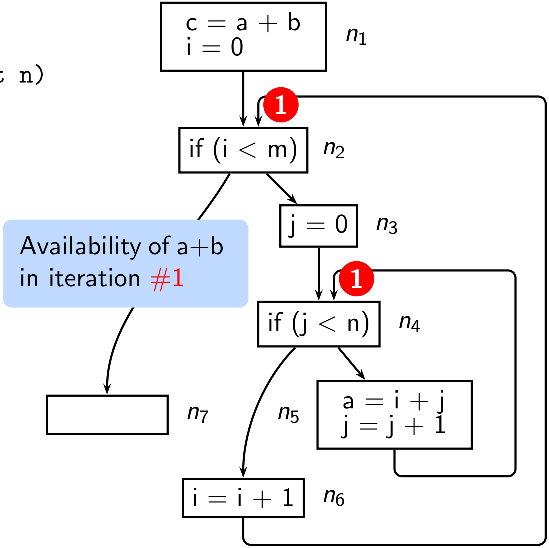
Example C Program with d(G,T) = 2

```
void fun(int m, int n)
{
   int i,j,a,b,c;
   c=a+b;
   i=0;
   while(i<m)
   {
         j=0;
         while(j<n)
         {
            a=i+j;
            j=j+1;
         }
         i=i+1;
      }
   }
```



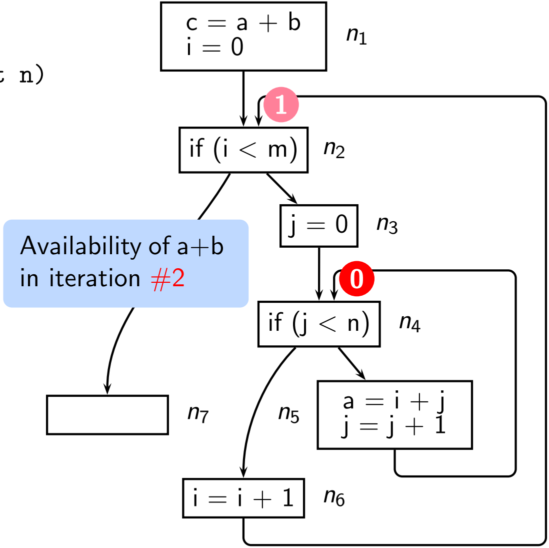
Example C Program with d(G,T) = 2

```c
void fun(int m, int n)
{
   int i,j,a,b,c;
   c=a+b;
   i=0;
   while(i<m)
   {
         j=0;
         while(j<n)
         {
            a=i+j;
            j=j+1;
         }
         i=i+1;
      }
   }
```



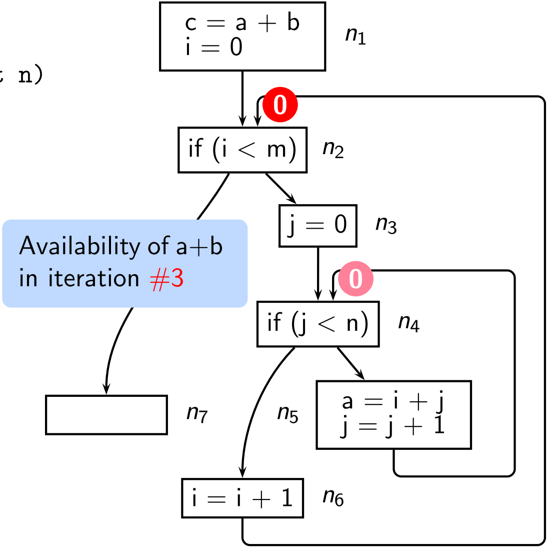
Example C Program with d(G,T) = 2

```
void fun(int m, int n)
{
    int i,j,a,b,c;
    c=a+b;
    i=0;
    while(i<m)
    {
        j=0;
        while(j<n)
        {
            a=i+j;
            j=j+1;
        }
        i=i+1;
    }
}
```



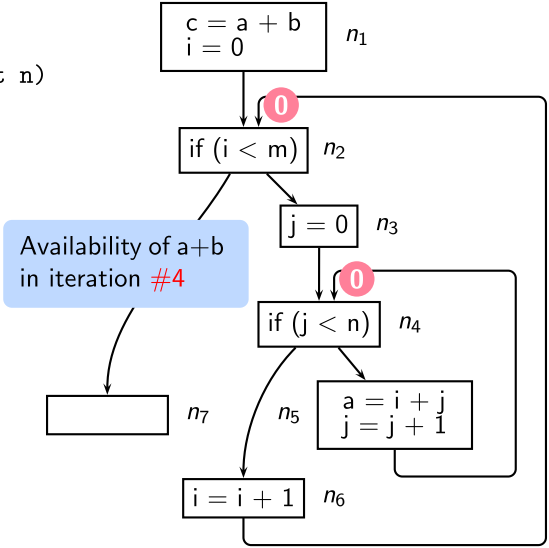
Example C Program with d(G,T) = 2

```
void fun(int m, int n)
{
    int i,j,a,b,c;
    c=a+b;
    i=0;
    while(i<m)
    {
        j=0;
        while(j<n)
        {
            a=i+j;
            j=j+1;
        }
        i=i+1;
    }
}
```



Example C Program with d(G,T) = 2

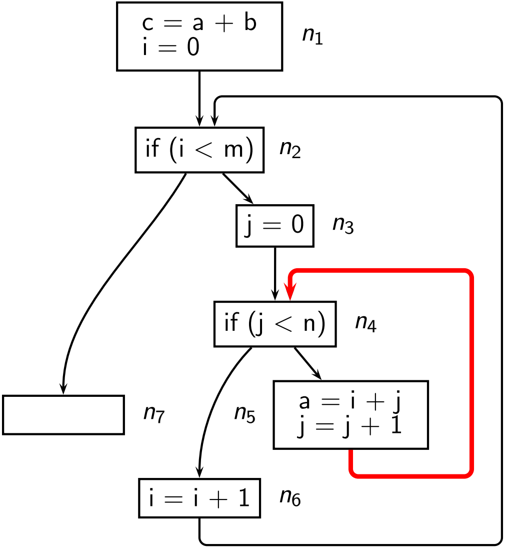
```c
void fun(int m, int n)
{
    int i,j,a,b,c;
    c=a+b;
    i=0;
    while(i<m)
    {
        j=0;
        while(j<n)
        {
            a=i+j;
            j=j+1;
        }
        i=i+1;
    }
}
```



3 + 1 iterations for available expressions analysis

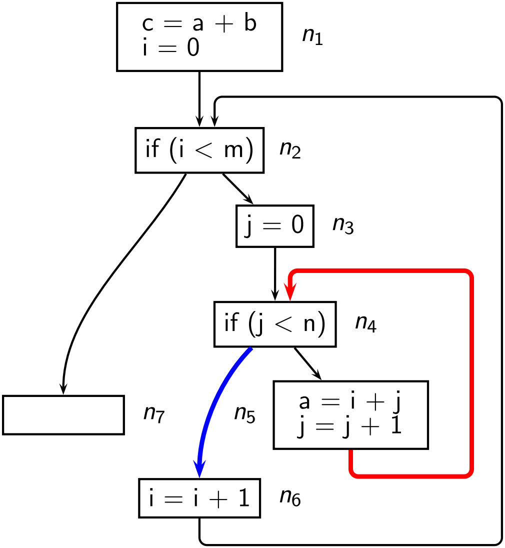
Example C Program with d(G,T) = 2

```
void fun(int m, int n)
{
   int i,j,a,b,c;
   c=a+b;
   i=0;
   while(i<m)
   {
         j=0;
         while(j<n)
         {
            a=i+j;
            j=j+1;
         }
         i=i+1;
      }
   }
```



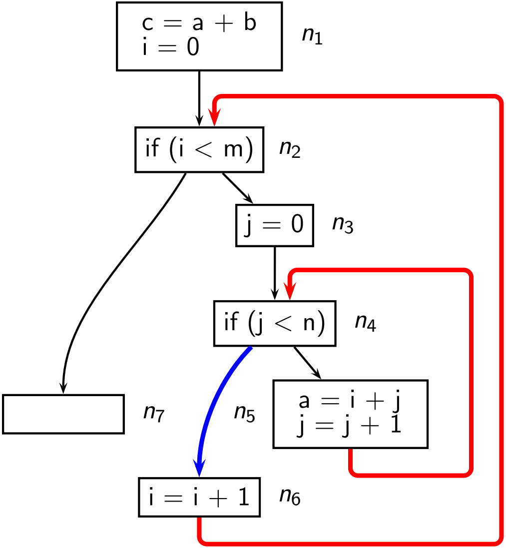
Example C Program with d(G,T) = 2

```
void fun(int m, int n)
{
   int i,j,a,b,c;
   c=a+b;
   i=0;
   while(i<m)
   {
         j=0;
         while(j<n)
         {
            a=i+j;
            j=j+1;
         }
         i=i+1;
      }
   }
```



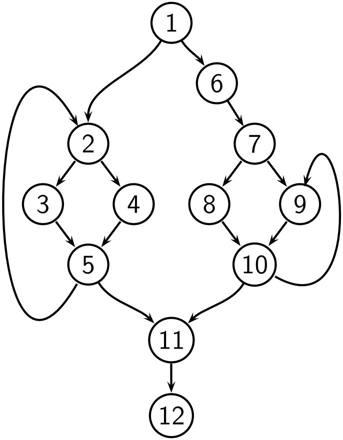
Example C Program with d(G,T) = 2

```c
void fun(int m, int n)
{
   int i,j,a,b,c;
   c=a+b;
   i=0;
   while(i<m)
   {
         j=0;
         while(j<n)
         {
            a=i+j;
            j=j+1;
         }
         i=i+1;
      }
   }
```



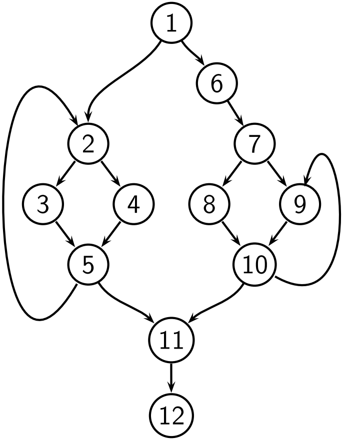
Complexity of Bidirectional Bit Vector Frameworks

Example: Consider the following CFG for PRE



Complexity of Bidirectional Bit Vector Frameworks

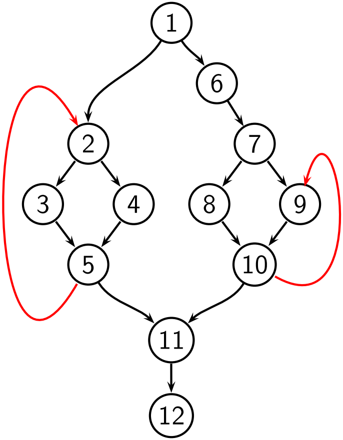
Example: Consider the following CFG for PRE



- Node numbers are in reverse post order

Complexity of Bidirectional Bit Vector Frameworks

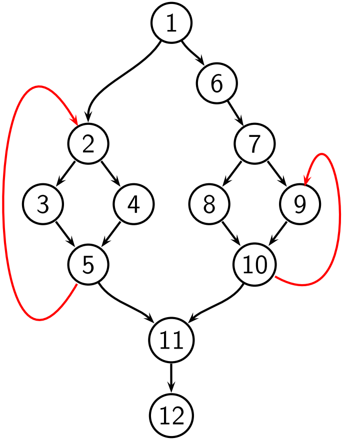
Example: Consider the following CFG for PRE



- Node numbers are in reverse post order
- Back edges in the graph are $n_5 \rightarrow n_2$ and $n_{10} \rightarrow n_9$

Complexity of Bidirectional Bit Vector Frameworks

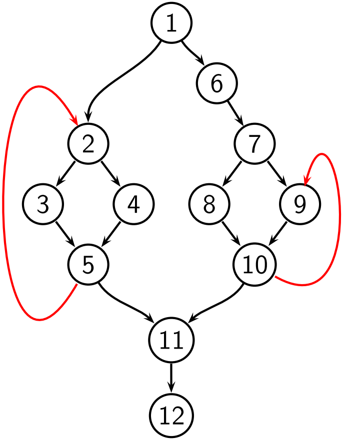
Example: Consider the following CFG for PRE



- Node numbers are in reverse post order
- Back edges in the graph are $n_5 \rightarrow n_2$ and $n_{10} \rightarrow n_9$
- $d(G, T) = 1$

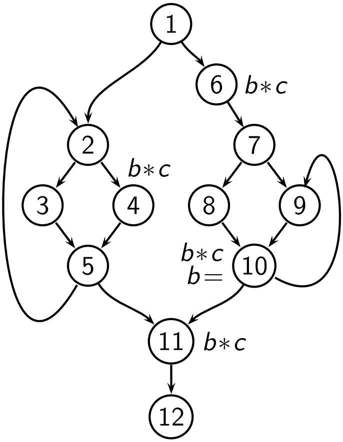
Complexity of Bidirectional Bit Vector Frameworks

Example: Consider the following CFG for PRE



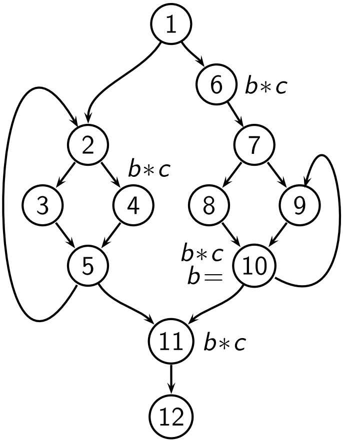
- Node numbers are in reverse post order
- Back edges in the graph are $n_5 \rightarrow n_2$ and $n_{10} \rightarrow n_9$
- $d(G, T) = 1$
- Actual iterations : 5

Complexity of Bidirectional Bit Vector Frameworks



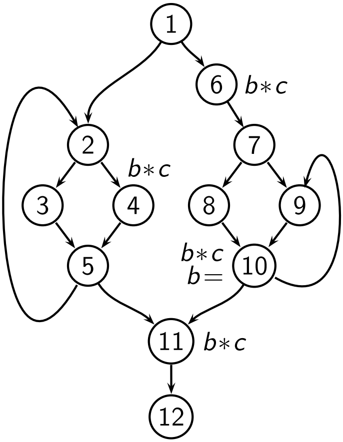
	Pairs of *Out*,*In* Values							
	Initia-lization	Changes in Iterations					Final values & transformation	
		#1	#2	#3	#4	#5		
	O,I	O,I	O,I	O,I	O,I	O,I	O,I	
12	0,1							
11	1,1							
10	1,1							
9	1,1							
8	1,1							
7	1,1							
6	1,1							
5	1,1							
4	1,1							
3	1,1							
2	1,1							
1	1,1							

Complexity of Bidirectional Bit Vector Frameworks



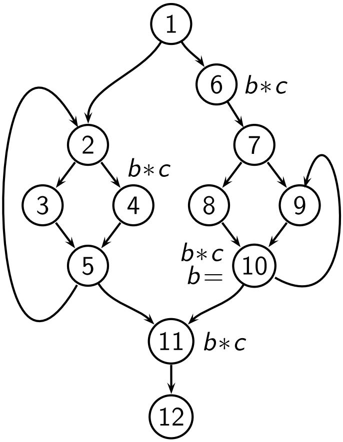
	Pairs of *Out*,*In* Values							
	Initia-lization	Changes in Iterations					Final values & transformation	
		#1	#2	#3	#4	#5		
	O,I	O,I	O,I	O,I	O,I	O,I	O,I	
12	0,1	0,0						
11	1,1	0,1						
10	1,1							
9	1,1							
8	1,1							
7	1,1							
6	1,1	1,0						
5	1,1							
4	1,1							
3	1,1							
2	1,1							
1	1,1	0,0						

Complexity of Bidirectional Bit Vector Frameworks



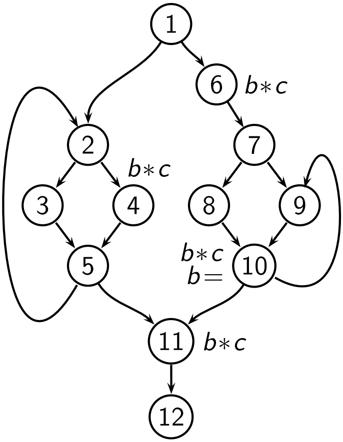
	Pairs of *Out*,*In* Values							
	Initia-lization	Changes in Iterations					Final values & transformation	
		#1	#2	#3	#4	#5		
	O,I	O,I	O,I	O,I	O,I	O,I	O,I	
12	0,1	0,0						
11	1,1	0,1						
10	1,1							
9	1,1							
8	1,1							
7	1,1							
6	1,1	1,0						
5	1,1							
4	1,1							
3	1,1							
2	1,1		1,0					
1	1,1	0,0						

Complexity of Bidirectional Bit Vector Frameworks



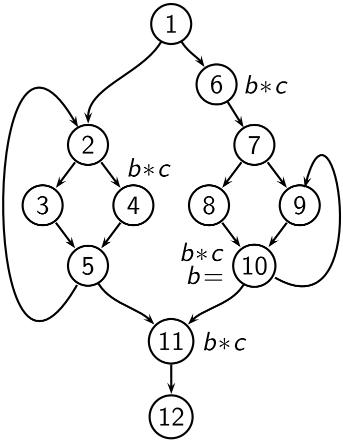
	Pairs of *Out*,*In* Values							
	Initia-lization	Changes in Iterations					Final values & transformation	
		#1	#2	#3	#4	#5		
	O,I	O,I	O,I	O,I	O,I	O,I	O,I	
12	0,1	0,0						
11	1,1	0,1						
10	1,1							
9	1,1							
8	1,1							
7	1,1							
6	1,1	1,0						
5	1,1			0,0				
4	1,1			0,1				
3	1,1			0,0				
2	1,1		1,0	0,0				
1	1,1	0,0						

Complexity of Bidirectional Bit Vector Frameworks



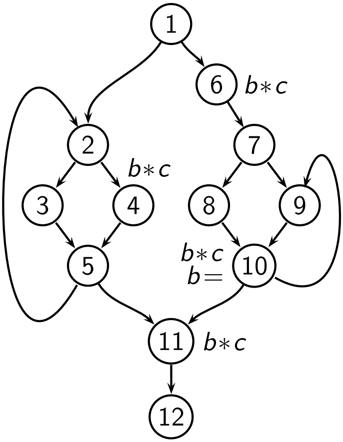
	Pairs of *Out*,*In* Values							
	Initia-lization	Changes in Iterations					Final values & transformation	
		#1	#2	#3	#4	#5		
	O,I	O,I	O,I	O,I	O,I	O,I	O,I	
12	0,1	0,0						
11	1,1	0,1			0,0			
10	1,1				0,1			
9	1,1				1,0			
8	1,1							
7	1,1				0,0			
6	1,1	1,0			0,0			
5	1,1			0,0				
4	1,1			0,1	0,0			
3	1,1			0,0				
2	1,1		1,0	0,0				
1	1,1	0,0						

Complexity of Bidirectional Bit Vector Frameworks



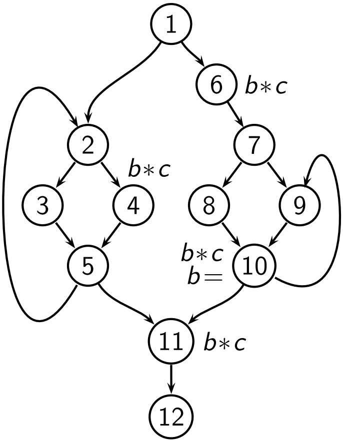
	Pairs of *Out*,*In* Values							
	Initia-lization	Changes in Iterations					Final values & transformation	
		#1	#2	#3	#4	#5		
	O,I	O,I	O,I	O,I	O,I	O,I	O,I	
12	0,1	0,0						
11	1,1	0,1			0,0			
10	1,1				0,1			
9	1,1				1,0			
8	1,1					1,0		
7	1,1				0,0			
6	1,1	1,0			0,0			
5	1,1			0,0				
4	1,1			0,1	0,0			
3	1,1			0,0				
2	1,1		1,0	0,0				
1	1,1	0,0						

Complexity of Bidirectional Bit Vector Frameworks



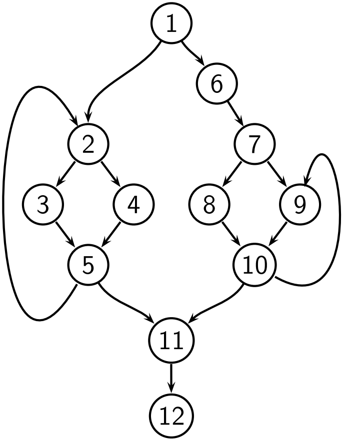
	Pairs of *Out*,*In* Values							
	Initialization	Changes in Iterations					Final values & transformation	
		#1	#2	#3	#4	#5		
	O,I	O,I	O,I	O,I	O,I	O,I	O,I	
12	0,1	0,0					0,0	
11	1,1	0,1			0,0		0,0	
10	1,1				0,1		0,1	
9	1,1				1,0		1,0	
8	1,1					1,0	1,0	
7	1,1				0,0		0,0	
6	1,1	1,0			0,0		0,0	
5	1,1			0,0			0,0	
4	1,1			0,1	0,0		0,0	
3	1,1			0,0			0,0	
2	1,1		1,0	0,0			0,0	
1	1,1	0,0					0,0	

Complexity of Bidirectional Bit Vector Frameworks



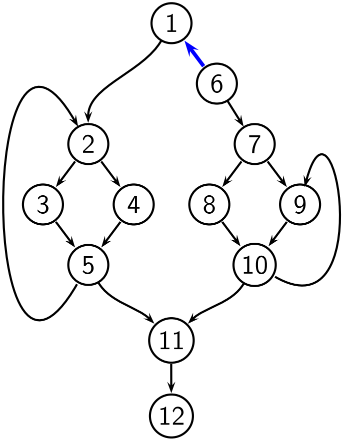
	Pairs of *Out*,*In* Values							
	Initia-lization	Changes in Iterations					Final values & transformation	
		#1	#2	#3	#4	#5		
	O,I	O,I	O,I	O,I	O,I	O,I	O,I	
12	0,1	0,0					0,0	
11	1,1	0,1			0,0		0,0	
10	1,1				0,1		0,1	Delete
9	1,1				1,0		1,0	Insert
8	1,1					1,0	1,0	Insert
7	1,1				0,0		0,0	
6	1,1	1,0			0,0		0,0	
5	1,1			0,0			0,0	
4	1,1			0,1	0,0		0,0	
3	1,1			0,0			0,0	
2	1,1		1,0	0,0			0,0	
1	1,1	0,0					0,0	

An Example of Information Flow in Our PRE Analysis



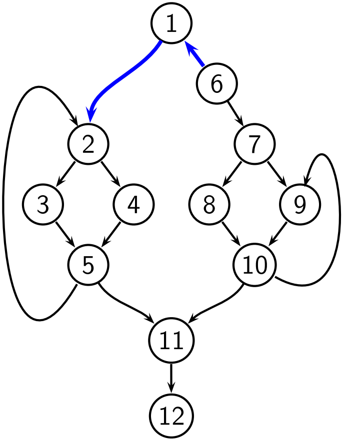
- $PavIn_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)

An Example of Information Flow in Our PRE Analysis



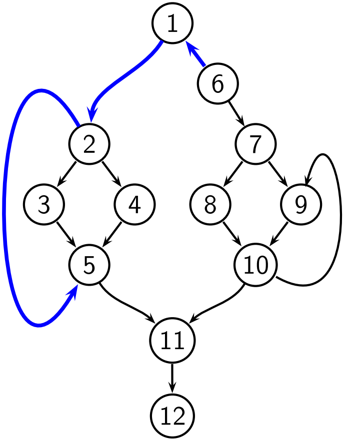
- $PavIn_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)

An Example of Information Flow in Our PRE Analysis



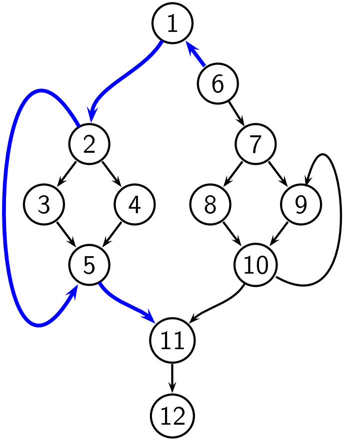
- *PavIn*$_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)

An Example of Information Flow in Our PRE Analysis



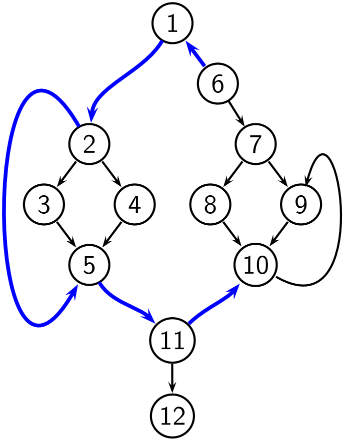
- *PavIn*$_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)

An Example of Information Flow in Our PRE Analysis



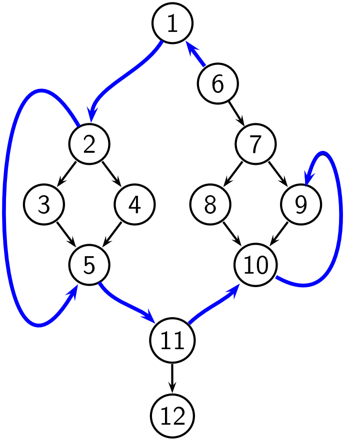
- $PavIn_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)

An Example of Information Flow in Our PRE Analysis



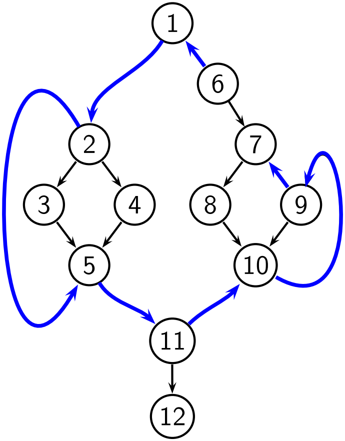
- $PavIn_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)

An Example of Information Flow in Our PRE Analysis



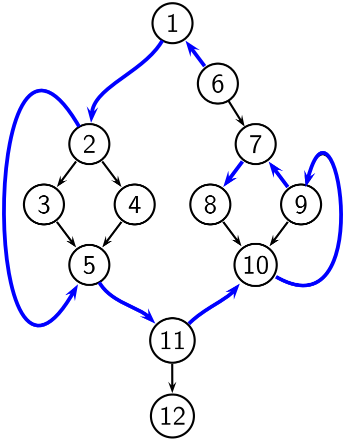
- $PavIn_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)

An Example of Information Flow in Our PRE Analysis



- $PavIn_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)

An Example of Information Flow in Our PRE Analysis



- $PavIn_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)

Information Flow and Information Flow Paths

- Default value at each program point: $\top$
- *Information flow path*

Information Flow and Information Flow Paths

- Default value at each program point: $\top$
- *Information flow path*

    Sequence of adjacent program points

Information Flow and Information Flow Paths

- Default value at each program point: $\top$
- *Information flow path*

  Sequence of adjacent program points
  along which data flow values change

Information Flow and Information Flow Paths

- Default value at each program point: $\top$
- *Information flow path*

  Sequence of adjacent program points
  along which data flow values change

- A change in the data flow at a program point could be
  - *Generation of information*
    Change from $\top$ to a non-$\top$ due to local effect (i.e. $f(\top) \neq \top$)
  - *Propagation of information*
    Change from x to y such that $y \sqsubseteq x$ due to global effect

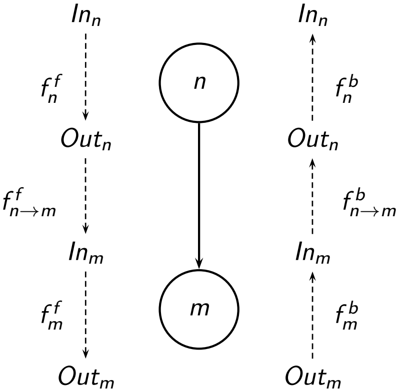
Information Flow and Information Flow Paths

- Default value at each program point: $\top$
- *Information flow path*

  Sequence of adjacent program points
  along which data flow values change

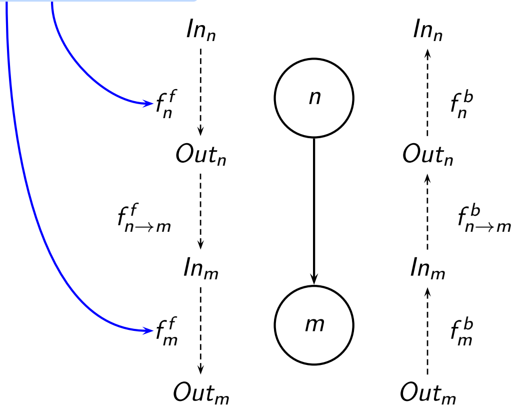
- A change in the data flow at a program point could be
  - *Generation of information*
    Change from $\top$ to a non-$\top$ due to local effect (i.e. $f(\top) \neq \top$)
  - *Propagation of information*
    Change from x to y such that $y \sqsubseteq x$ due to global effect
- Information flow path (ifp) need not be a graph theoretic path

Edge and Node Flow Functions

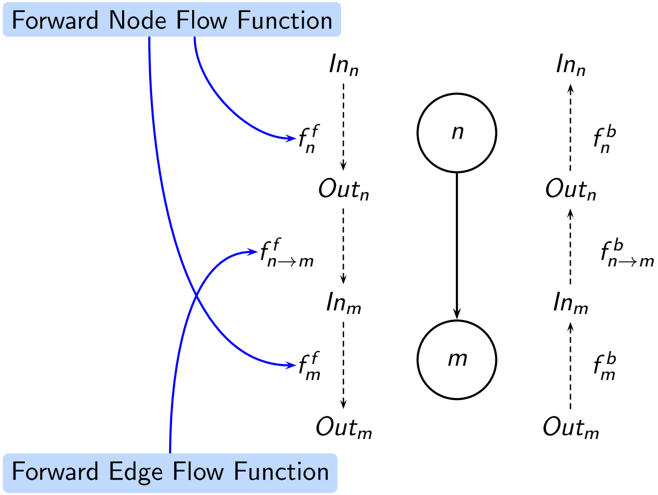


Edge and Node Flow Functions

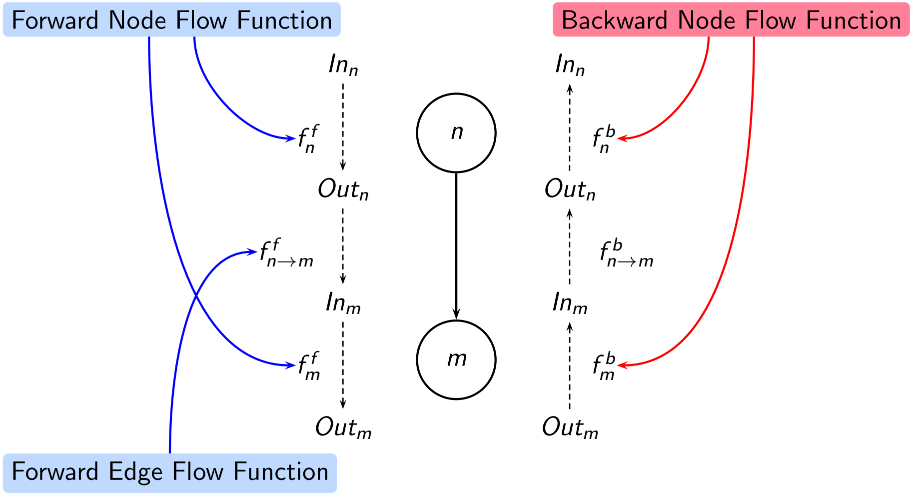
Forward Node Flow Function



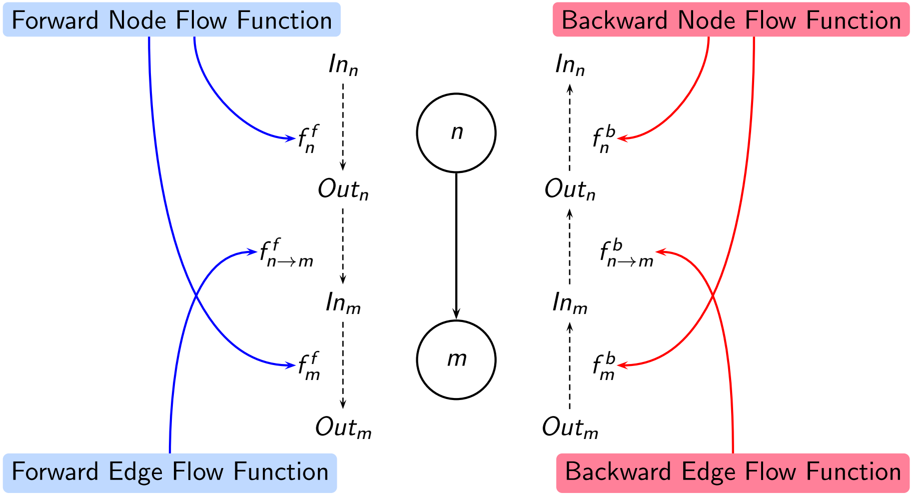
Edge and Node Flow Functions



Edge and Node Flow Functions



Edge and Node Flow Functions



General Data Flow Equations

$$In_n = \begin{cases} BI_{Start} \sqcap f_n^b(Out_n) & n = Start \\ \left(\bigsqcap_{m \in pred(n)} f_{m \to n}^f(Out_m) \right) \sqcap f_n^b(Out_n) & \text{otherwise} \end{cases}$$

$$Out_n = \begin{cases} BI_{End} \sqcap f_n^f(In_n) & n = End \\ \left(\bigsqcap_{m \in succ(n)} f_{m \to n}^b(In_m) \right) \sqcap f_n^f(In_n) & \text{otherwise} \end{cases}$$

- Edge flow functions are typically identity

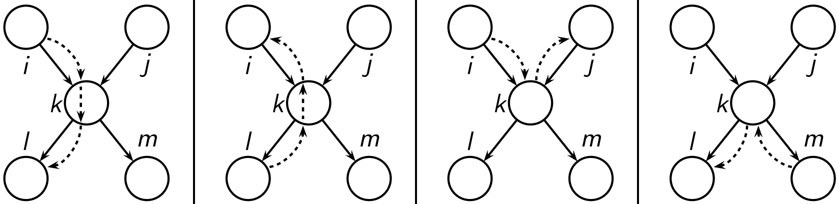
$$\forall x \in L,\ f(x) = x$$

- If particular flows are absent, the corresponding flow functions are

$$\forall x \in L,\ f(x) = \top$$

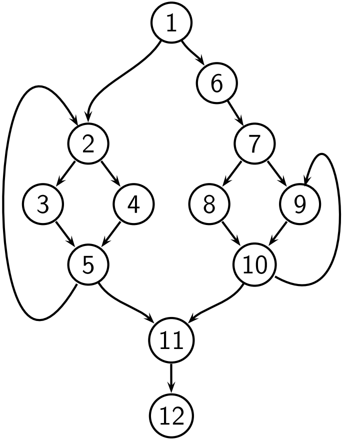
Modelling Information Flows Using Edge and Node Flow Functions

Forward	Backward	Bidirectional	Bidirectional



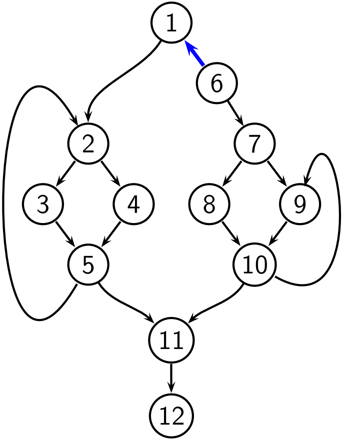
$f^f_{k\to l} \circ f^f_k \circ f^f_{i\to k}$	$f^b_{i\to k} \circ f^b_k \circ f^b_{k\to l}$	$f^b_{j\to k} \circ f^f_{i\to k}$	$f^f_{k\to l} \circ f^b_{k\to m}$

Information Flow Paths in PRE



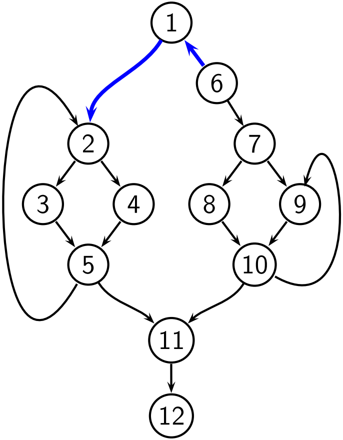
- Information could flow along arbitrary paths

Information Flow Paths in PRE



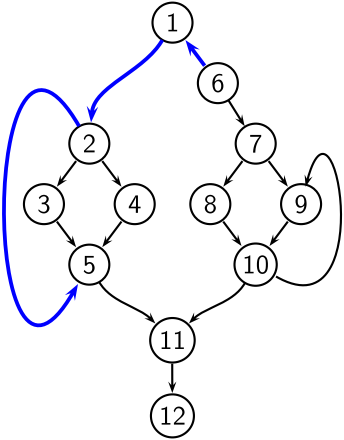
- Information could flow along arbitrary paths

Information Flow Paths in PRE



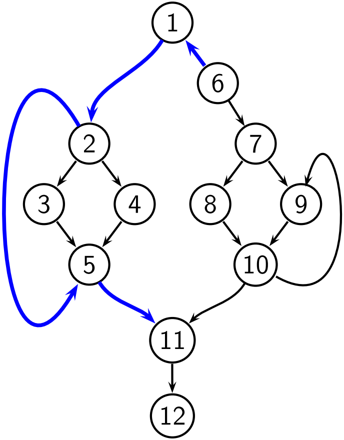
- Information could flow along arbitrary paths

Information Flow Paths in PRE



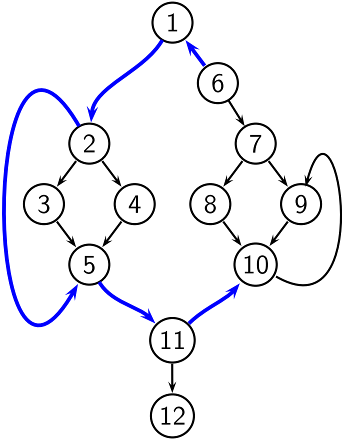
- Information could flow along arbitrary paths

Information Flow Paths in PRE



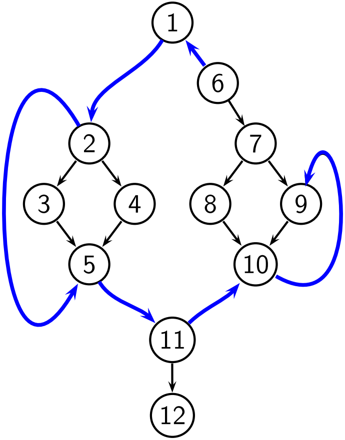
- Information could flow along arbitrary paths

Information Flow Paths in PRE



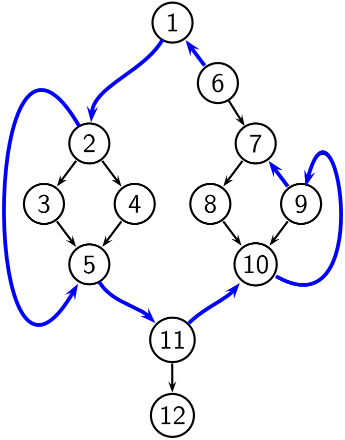
- Information could flow along arbitrary paths

Information Flow Paths in PRE



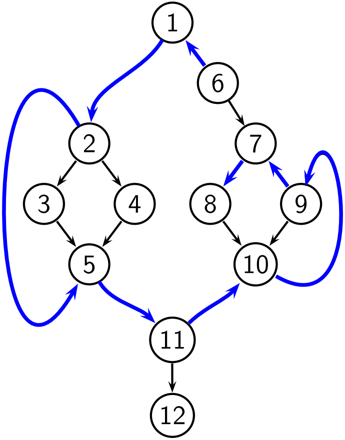
- Information could flow along arbitrary paths

Information Flow Paths in PRE



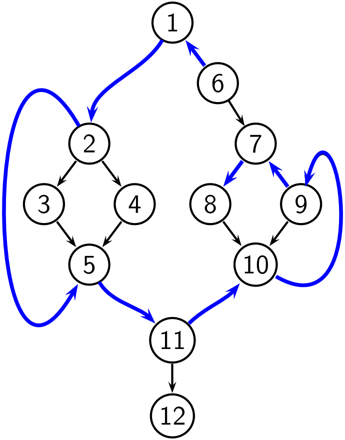
- Information could flow along arbitrary paths

Information Flow Paths in PRE



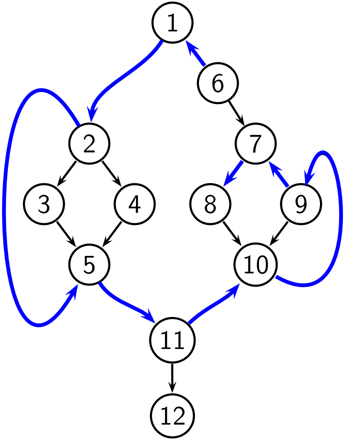
- Information could flow along arbitrary paths
- Theoretically predicted number : 144

Information Flow Paths in PRE



- Information could flow along arbitrary paths
- Theoretically predicted number : 144
- Actual iterations : 5

Information Flow Paths in PRE



- Information could flow along arbitrary paths
- Theoretically predicted number : 144
- Actual iterations : 5
- Not related to depth (1)

Complexity of Worklist Algorithms for Bit Vector Frameworks

- Assume n nodes and r entities
- Total number of data flow values $= 2 \cdot n \cdot r$
- A data flow value can change at most once
- Complexity is $\mathcal{O}(n \cdot r)$

Complexity of Worklist Algorithms for Bit Vector Frameworks

- Assume n nodes and r entities
- Total number of data flow values $= 2 \cdot n \cdot r$
- A data flow value can change at most once
- Complexity is $\mathcal{O}\left(n \cdot r\right)$
- *Must be same for both unidirectional and bidirectional frameworks*
  (Number of data flow values does not change!)

Lacuna with Older Estimates of PRE Complexity

- Lacuna with PRE : Complexity
  - r is typically $\mathcal{O}(n)$
  - Assuming that at most one data flow value changes in one traversal

Lacuna with Older Estimates of PRE Complexity

- Lacuna with PRE : Complexity
  - r is typically $\mathcal{O}(n)$
  - Assuming that at most one data flow value changes in one traversal
  - Worst case number of traversals $= \mathcal{O}\left(n^2\right)$

Lacuna with Older Estimates of PRE Complexity

- Lacuna with PRE : Complexity
  - r is typically $\mathcal{O}(n)$
  - Assuming that at most one data flow value changes in one traversal
  - Worst case number of traversals $= \mathcal{O}\left(n^2\right)$
- Practical graphs may have upto 50 nodes
  - Predicted number of traversals : 2,500
  - Practical number of traversals : ≤ 5

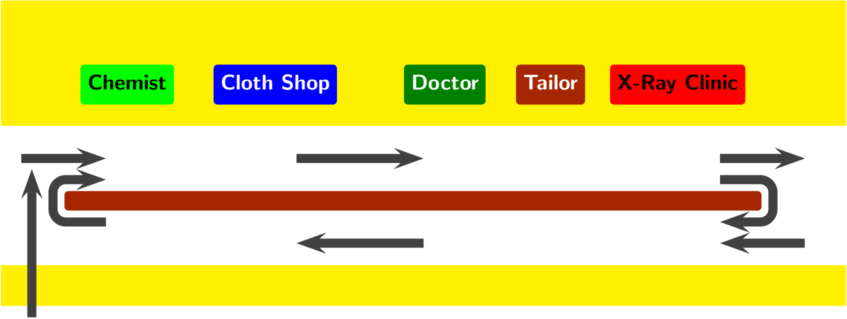
Lacuna with Older Estimates of PRE Complexity

- Lacuna with PRE : Complexity
  - r is typically $\mathcal{O}(n)$
  - Assuming that at most one data flow value changes in one traversal
  - Worst case number of traversals $= \mathcal{O}\left(n^2\right)$
- Practical graphs may have upto 50 nodes
  - Predicted number of traversals : 2,500
  - Practical number of traversals : ≤ 5
- No explanation for about 14 years despite dozens of efforts

Lacuna with Older Estimates of PRE Complexity

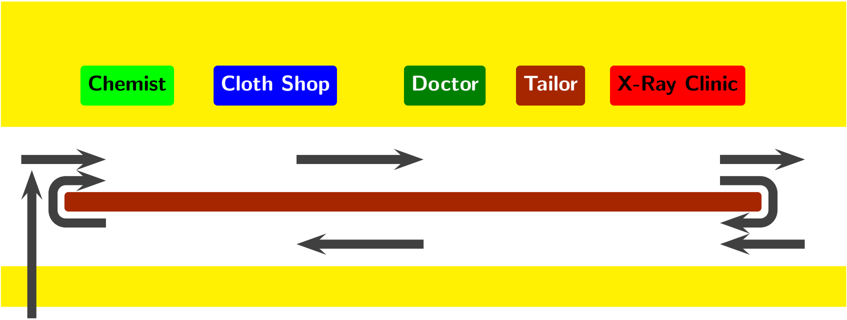
- Lacuna with PRE : Complexity
  - r is typically $\mathcal{O}(n)$
  - Assuming that at most one data flow value changes in one traversal
  - Worst case number of traversals $= \mathcal{O}\left(n^2\right)$
- Practical graphs may have upto 50 nodes
  - Predicted number of traversals : 2,500
  - Practical number of traversals : ≤ 5
- No explanation for about 14 years despite dozens of efforts
- Not much experimentation with performing advanced optimizations involving bidirectional dependency

Complexity of Round Robin Iterative Method



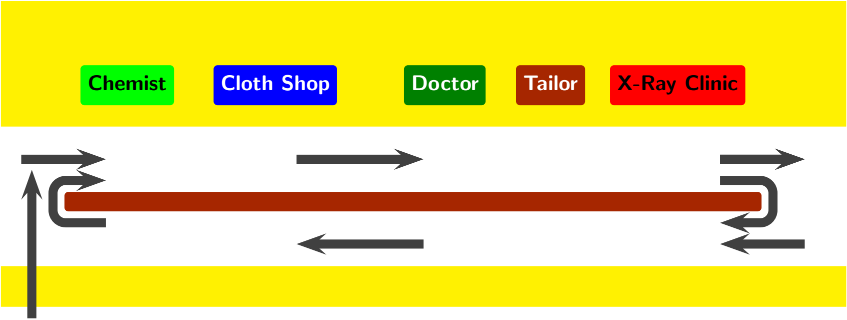
- Buy OTC (Over-The-Counter) medicine   No U-Turn   1 Trip

Complexity of Round Robin Iterative Method



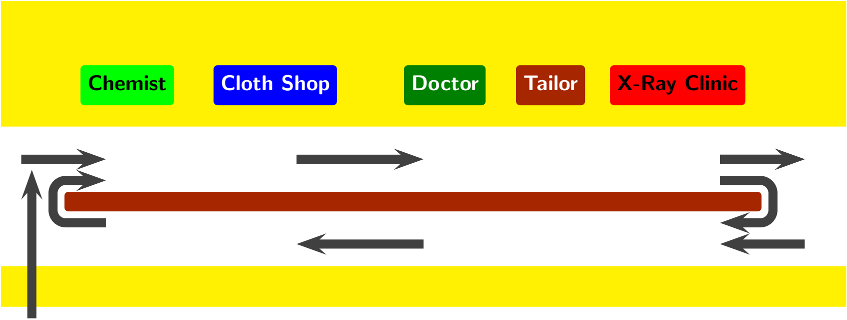
• Buy OTC (Over-The-Counter) medicine	No U-Turn	1 Trip
• Buy cloth. Give it to the tailor for stitching	No U-Turn	1 Trip

Complexity of Round Robin Iterative Method



• Buy OTC (Over-The-Counter) medicine	No U-Turn	1 Trip
• Buy cloth. Give it to the tailor for stitching	No U-Turn	1 Trip
• Buy medicine with doctor's prescription	1 U-Turn	2 Trips

Complexity of Round Robin Iterative Method



• Buy OTC (Over-The-Counter) medicine	No U-Turn	1 Trip
• Buy cloth. Give it to the tailor for stitching	No U-Turn	1 Trip
• Buy medicine with doctor's prescription	1 U-Turn	2 Trips
• Buy medicine with doctor's prescription The diagnosis requires X-Ray	2 U-Turns	3 Trips

Information Flow Paths and Width of a Graph

- A traversal $u \rightarrow v$ in an ifp is
  - *Compatible* if u is visited *before* v in the chosen graph traversal
  - *Incompatible* if u is visited *after* v in the chosen graph traversal

Information Flow Paths and Width of a Graph

- A traversal $u \rightarrow v$ in an ifp is
  - *Compatible* if u is visited *before* v in the chosen graph traversal
  - *Incompatible* if u is visited *after* v in the chosen graph traversal
- Every incompatible edge traversal requires one additional iteration

Information Flow Paths and Width of a Graph

- A traversal $u \rightarrow v$ in an ifp is
  - *Compatible* if u is visited *before* v in the chosen graph traversal
  - *Incompatible* if u is visited *after* v in the chosen graph traversal
- Every incompatible edge traversal requires one additional iteration
- Width of a program flow graph with respect to a data flow framework

  Maximum number of incompatible traversals in any ifp, no part of which is bypassed

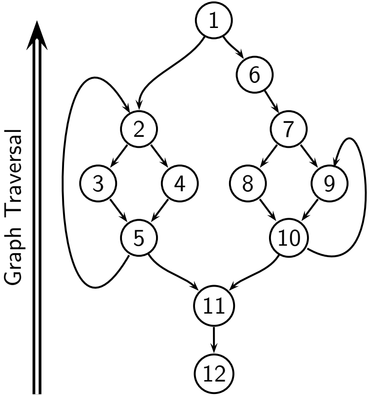
Information Flow Paths and Width of a Graph

- A traversal $u \rightarrow v$ in an ifp is
  - *Compatible* if u is visited *before* v in the chosen graph traversal
  - *Incompatible* if u is visited *after* v in the chosen graph traversal
- Every incompatible edge traversal requires one additional iteration
- Width of a program flow graph with respect to a data flow framework

  Maximum number of incompatible traversals in any ifp, no part of which is bypassed
- Width $+$ 1 iterations are sufficient to converge on MFP solution

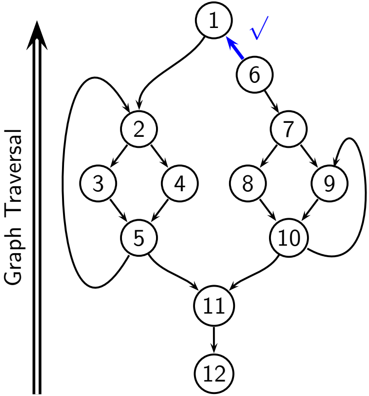
  (1 additional iteration may be required for verifying convergence)

Complexity of Bidirectional Bit Vector Frameworks



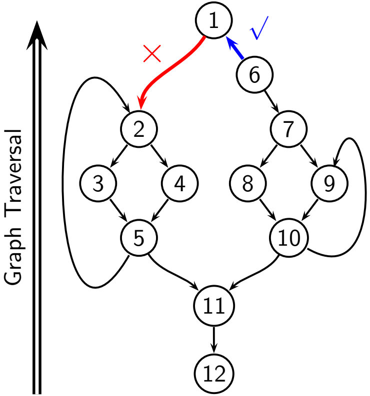
- Every "incompatible" edge traversal
  $\Rightarrow$ **One additional graph traversal**

Complexity of Bidirectional Bit Vector Frameworks



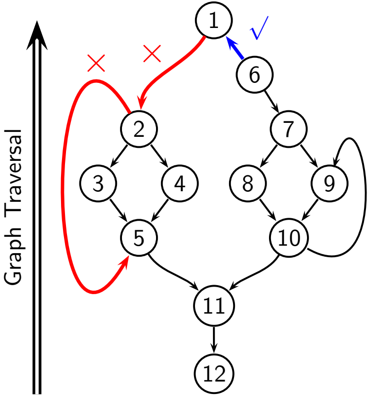
- Every "incompatible" edge traversal
  $\Rightarrow$ **One additional graph traversal**
- Max. Incompatible edge traversals
  = *Width* of the graph = **0?**
- Maximum number of traversals =
  1 + Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



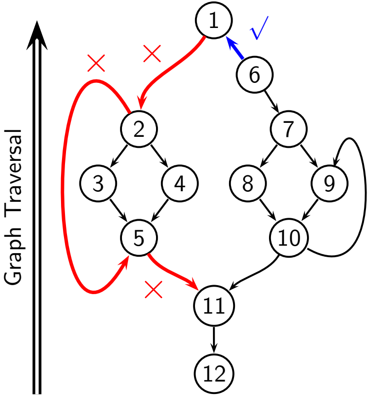
- Every "incompatible" edge traversal
  $\Rightarrow$ **One additional graph traversal**
- Max. Incompatible edge traversals
  $=$ *Width* of the graph $=$ **1?**
- Maximum number of traversals $=$
  $1 +$ Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



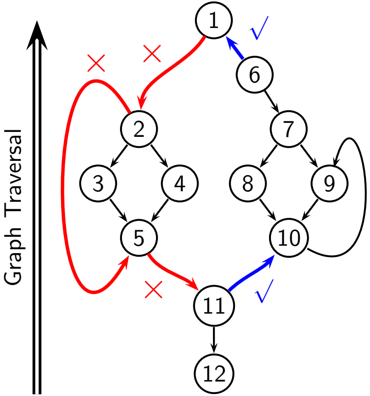
- Every "incompatible" edge traversal
  $\Rightarrow$ **One additional graph traversal**
- Max. Incompatible edge traversals
  = *Width* of the graph = **2?**
- Maximum number of traversals =
  1 + Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



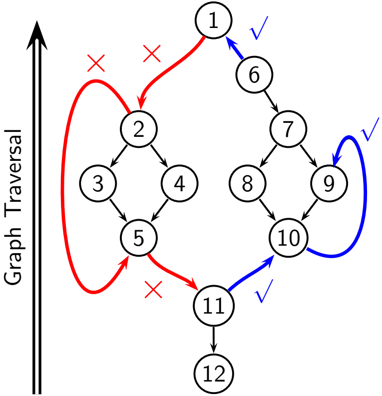
- **Every "incompatible" edge traversal $\Rightarrow$ One additional graph traversal**
- Max. Incompatible edge traversals
  = *Width* of the graph = **3?**
- Maximum number of traversals =
  1 + Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



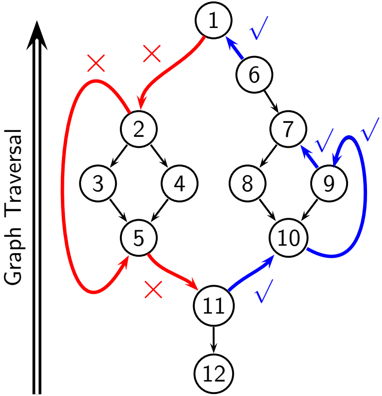
- **Every "incompatible" edge traversal**
  ⇒ One additional graph traversal
- Max. Incompatible edge traversals
  = *Width* of the graph = **3?**
- Maximum number of traversals =
  1 + Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



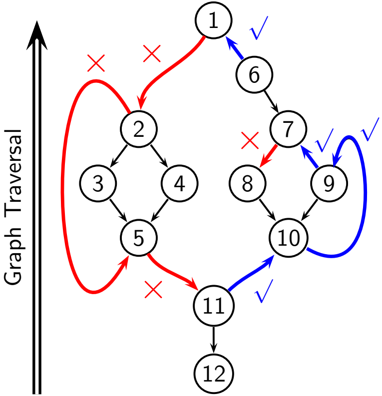
- Every "incompatible" edge traversal
  $\Rightarrow$ **One additional graph traversal**
- Max. Incompatible edge traversals
  = *Width* of the graph = **3?**
- Maximum number of traversals =
  1 + Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



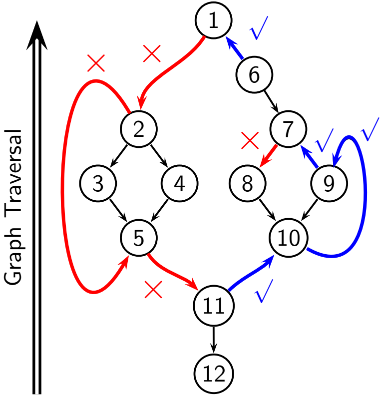
- Every "incompatible" edge traversal
  $\Rightarrow$ **One additional graph traversal**
- Max. Incompatible edge traversals
  = *Width* of the graph = **3?**
- Maximum number of traversals =
  1 + Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



- Every "incompatible" edge traversal
  ⇒ **One additional graph traversal**
- Max. Incompatible edge traversals
  = *Width* of the graph = **4**
- Maximum number of traversals =
  1 + Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



- Every "incompatible" edge traversal
  ⇒ **One additional graph traversal**
- Max. Incompatible edge traversals
  = *Width* of the graph = **4**
- Maximum number of traversals =
  1 + 4 = 5

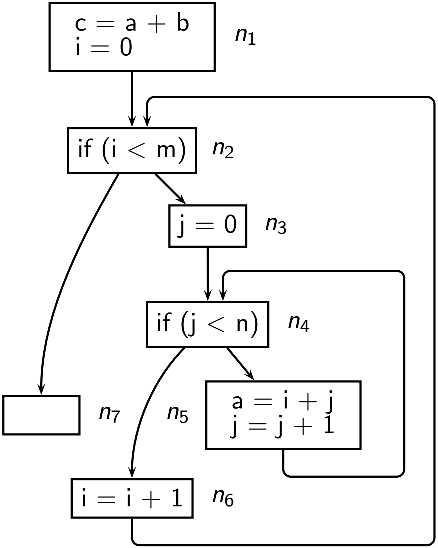
Width Subsumes Depth

- Depth is applicable only to unidirectional data flow frameworks
- Width is applicable to both unidirectional and bidirectional frameworks
- For a given graph for a unidirectional bit vector framework,
  Width $\leq$ Depth
  Width provides a tighter bound

Comparison Between Width and Depth

- Depth is purely a graph theoretic property whereas width depends on control flow graph as well as the data framework
- Comparison between width and depth is meaningful only
  - For unidirectional frameworks
  - When the direction of traversal for computing width is the natural direction of traversal
- Since width excludes bypassed path segments, width can be smaller than depth

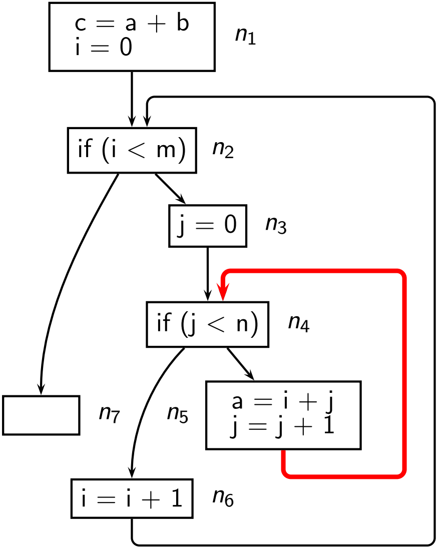
Width and Depth



Assuming reverse postorder traversal for available expressions analysis

- Depth = 2

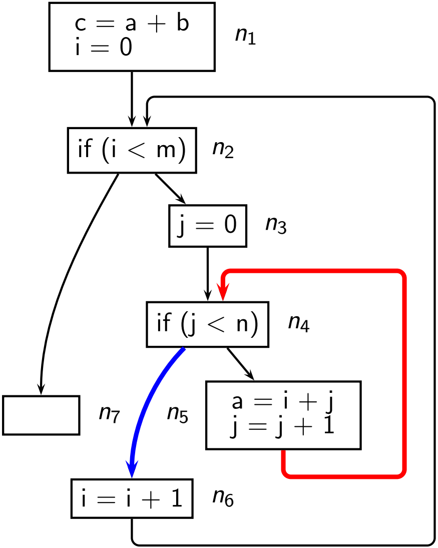
Width and Depth



Assuming reverse postorder traversal for available expressions analysis

- Depth = 2
- Information generation point n_5 kills expression "a + b"

Width and Depth



Assuming reverse postorder traversal for available expressions analysis

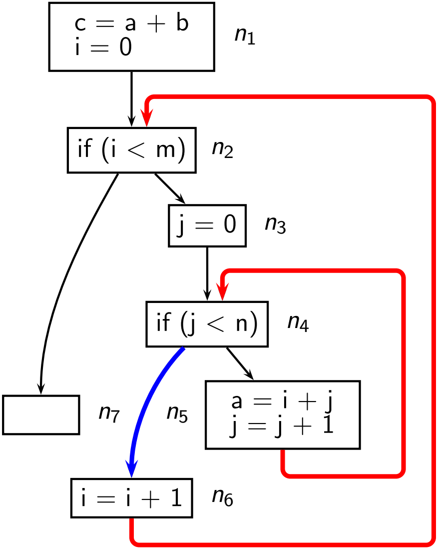
- Depth = 2
- Information generation point

  n_5 kills expression "a + b"
- Information propagation path

  $n_5 \rightarrow n_4 \rightarrow n_6 \rightarrow n_2$

  No Gen or Kill for "a + b" along this path

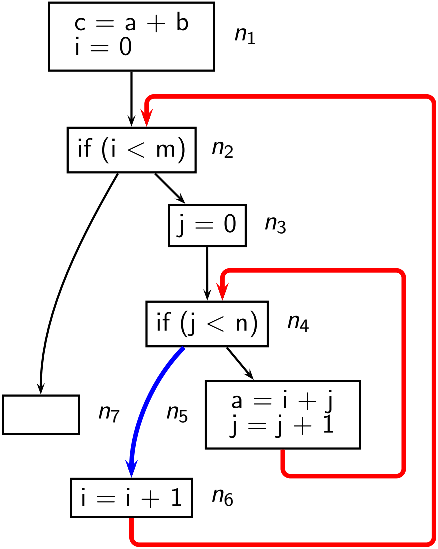
Width and Depth



Assuming reverse postorder traversal for available expressions analysis

- Depth = 2
- Information generation point
  n_5 kills expression "a + b"
- Information propagation path
  $n_5 \rightarrow n_4 \rightarrow n_6 \rightarrow n_2$
  No Gen or Kill for "a + b" along this path
- Width = 2

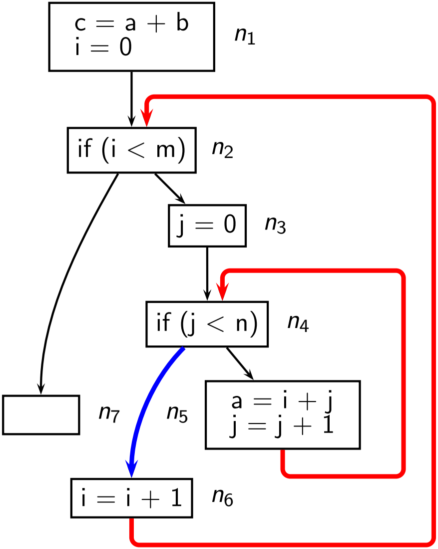
Width and Depth



Assuming reverse postorder traversal for available expressions analysis

- Depth = 2
- Information generation point
  n_5 kills expression "a + b"
- Information propagation path
  $n_5 \rightarrow n_4 \rightarrow n_6 \rightarrow n_2$
  No Gen or Kill for "a + b" along this path
- Width = 2
- What about "j + 1"?

Width and Depth



Assuming reverse postorder traversal for available expressions analysis

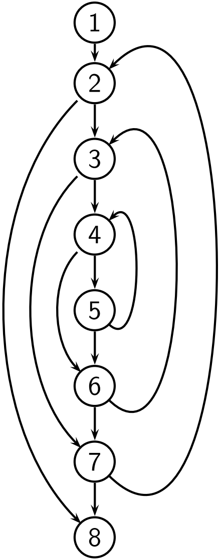
- Depth = 2
- Information generation point

  n_5 kills expression "a + b"
- Information propagation path

  $n_5 \rightarrow n_4 \rightarrow n_6 \rightarrow n_2$

  No Gen or Kill for "a + b" along this path
- Width = 2
- What about "j + 1"?
- Not available on entry to the loop

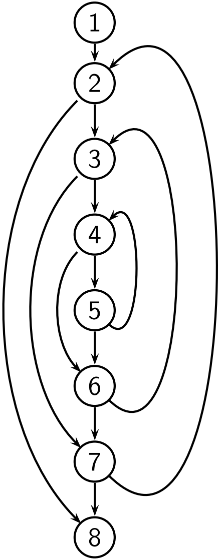
Width and Depth



Structures resulting from repeat-until loops with premature exits

- Depth = 3

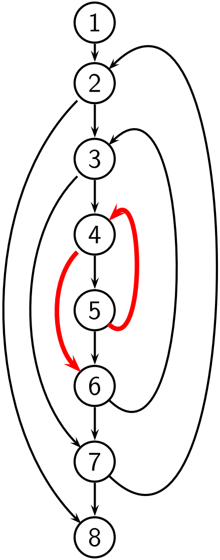
Width and Depth



Structures resulting from repeat-until loops with premature exits

- Depth = 3
- However, any unidirectional bit vector analysis is guaranteed to converge in 2 + 1 iterations

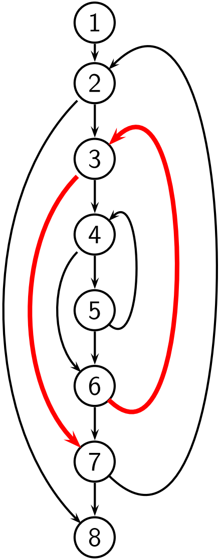
Width and Depth



Structures resulting from repeat-until loops with premature exits

- Depth = 3
- However, any unidirectional bit vector analysis is guaranteed to converge in 2 + 1 iterations
- ifp $5 \rightarrow 4 \rightarrow 6$ is bypassed by the edge $5 \rightarrow 6$

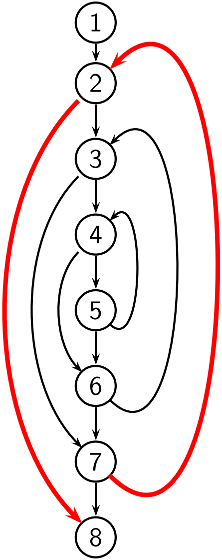
Width and Depth



Structures resulting from repeat-until loops with premature exits

- Depth = 3
- However, any unidirectional bit vector analysis is guaranteed to converge in 2 + 1 iterations
- ifp $5 \rightarrow 4 \rightarrow 6$ is bypassed by the edge $5 \rightarrow 6$
- ifp $6 \rightarrow 3 \rightarrow 7$ is bypassed by the edge $6 \rightarrow 7$

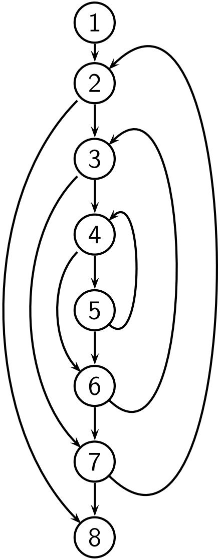
Width and Depth



Structures resulting from repeat-until loops with premature exits

- Depth = 3
- However, any unidirectional bit vector analysis is guaranteed to converge in 2 + 1 iterations
- ifp $5 \rightarrow 4 \rightarrow 6$ is bypassed by the edge $5 \rightarrow 6$
- ifp $6 \rightarrow 3 \rightarrow 7$ is bypassed by the edge $6 \rightarrow 7$
- ifp $7 \rightarrow 2 \rightarrow 8$ is bypassed by the edge $7 \rightarrow 8$

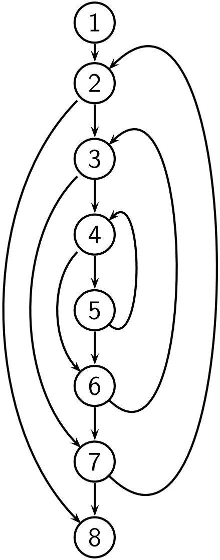
Width and Depth



Structures resulting from repeat-until loops with premature exits

- Depth = 3
- However, any unidirectional bit vector analysis is guaranteed to converge in 2 + 1 iterations
- ifp 5 → 4 → 6 is bypassed by the edge 5 → 6
- ifp 6 → 3 → 7 is bypassed by the edge 6 → 7
- ifp 7 → 2 → 8 is bypassed by the edge 7 → 8
- For forward unidirectional frameworks, width is 1

Width and Depth



Structures resulting from repeat-until loops with premature exits

- Depth = 3
- However, any unidirectional bit vector analysis is guaranteed to converge in 2 + 1 iterations
- ifp $5 \rightarrow 4 \rightarrow 6$ is bypassed by the edge $5 \rightarrow 6$
- ifp $6 \rightarrow 3 \rightarrow 7$ is bypassed by the edge $6 \rightarrow 7$
- ifp $7 \rightarrow 2 \rightarrow 8$ is bypassed by the edge $7 \rightarrow 8$
- For forward unidirectional frameworks, width is 1
- Splitting the bypassing edges and inserting nodes along those edges increases the width

Work List Based Iterative Algorithm

Directly traverses information flow paths

$In_0 = BI$
for all $j \neq 0$ **do**
$\{$ $In_j = \top$
  Add j to LIST
$\}$
while LIST is not empty **do**
$\{$ Let j be the first node in LIST. Remove it from LIST
  $temp = \displaystyle\prod_{p \in pred(j)} f_p(In_p)$
  if $temp \neq In_j$ **then**
  $\{$ $In_j = temp$
    Add all successors of j to LIST
  $\}$
$\}$

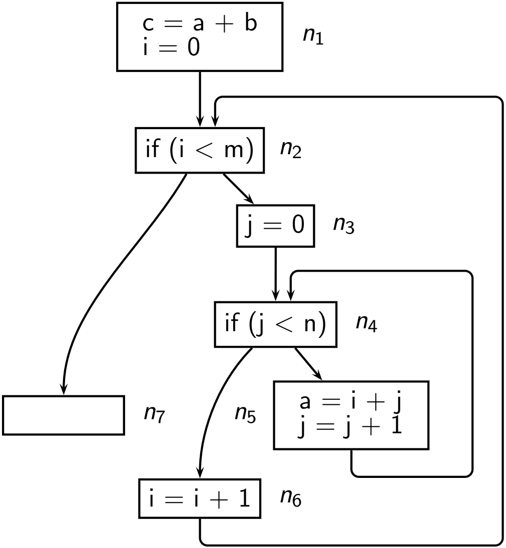
Tutorial Problem

Perform work list based iterative analysis for earlier examples. Assume that the work list follows FIFO (First in First Out) policy

Show the trace of the analysis in the following format:

Step	Node	Remaining work list	*Out* DFV	Change?	Node Added	Resulting work list

Tutorial Problem for Work List Based Analysis



For available expressions analysis

- Round robin method needs 3+1 iterations

  Total number of nodes processed $= 7 \times 4 = 28$
- We illustrate work list method for expression $a + b$

  (other expressions are unavailable in the first iteration because of BI)

Tutorial Problem for Work List Based Analysis

Step	Node	Remaining work list	*Out* DFV	Change?	Node Added	Resulting work list
1	n_1	$n_2, n_3, n_4, n_5, n_6, n_7$	1	No		$n_2, n_3, n_4, n_5, n_6, n_7$
2	n_2	n_3, n_4, n_5, n_6, n_7	1	No		n_3, n_4, n_5, n_6, n_7
3	n_3	n_4, n_5, n_6, n_7	1	No		n_4, n_5, n_6, n_7
4	n_4	n_5, n_6, n_7	1	No		n_5, n_6, n_7
5	n_5	n_6, n_7	0	Yes	n_4	n_6, n_7, n_4
6	n_6	n_7, n_4	1	No		n_7, n_4
7	n_7	n_4	1	No		n_4
8	n_4		0	Yes	n_5, n_6	n_5, n_6
9	n_5	n_6	0	No		n_6
10	n_6		0	Yes	n_2	n_2
11	n_2		0	Yes	n_3, n_7	n_3, n_7
12	n_3	n_7	0	Yes	n_4	n_7, n_4
13	n_7	n_4	0	Yes		n_4
14	n_4		0	No		Empty $\Rightarrow$ End

Comparing the Algorithms for Performing Data Flow Analysis

Round Robin Algorithm

$In_0 = BI$
for all $j \neq 0$ **do**
    $In_j = \top$
$change = true$
while $change$ **do**
{ $change = false$
    for $j = 1$ to $N - 1$ **do**
    { $temp = \bigsqcap_{p \in pred(j)} f_p(In_p)$
        if $temp \neq In_j$ **then**
        { $In_j = temp$
            $change = true$
        }
    }
}

Work List Algorithm

$In_0 = BI$
for all $j \neq 0$ **do**
{ $In_j = \top$
    Add j to LIST
}
while LIST is not empty **do**
{ Let j be the first node in LIST
    Remove node j from LIST
    $temp = \bigsqcap_{p \in pred(j)} f_p(In_p)$
    if $temp \neq In_j$ **then**
    { $In_j = temp$
        Add all successors of j to LIST
    }
}

An Efficient Work List Algorithm (1)

- Combines the traversal order of round robin algorithm with a need-based processing of work list algorithm
- The work list is initialized for nodes j such that $\text{OUT}_j = f_j(\top) \neq \top$
- Function *Process_Node*(*rpo*)
  - Computes the *In* and *Out* values of the node with RPO number *rpo*
  - If there is a change for the node
    - It adds the successors of the node to the work list, and
    - returns *true* if the RPO number of a successor is smaller than *rpo*

  In the latter case, the work list must be examined from the beginning
- Notation
  - The work list is an array *WL* whose indices are RPO numbers
    $WL[\,i\,] = true \Rightarrow$ the node with RPO number i needs to be processed
  - $RPO[i]$ gives the RPO number of node i
  - $NODE[i]$ gives the node whose RPO number is i

An Efficient Work List Algorithm (2)

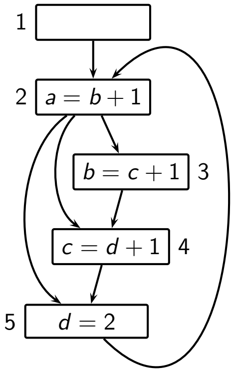
```
Efficient_Work_List_DFA()
{   Initialise()
    Begin: /* Statement label */
    for rpo = 0 to N − 1 do
        if Process_Node(rpo) then
            goto Begin:
}
Initialise()
{   j = NODE[0]
    In_j = BI
    for rpo = 0 to N − 1 do
    {   j = NODE[rpo]
        if rpo = 0 then
            Out_j = f_j(BI)
        else Out_j = f_j(⊤)
        if Out_j ≠ ⊤ then
            WL[rpo] = true
        else WL[rpo] = false
    }
}

Process_Node(rpo)
{   restart = false
    if WL[rpo] = true
    {   WL[rpo] = false
        j = NODE[rpo]
        In_j = ⊓_{p ∈ pred(j)} (Out_p)
        temp = f_j(In_j)
        if temp ≠ Out_j then
        {   Out_j = temp
            for all s ∈ succ(j) do
            {   rpos = RPO[s]
                WL[rpos] = true
                if rpos ≤ rpo then
                    restart = true
            }
        }
    }
    return restart
}
```

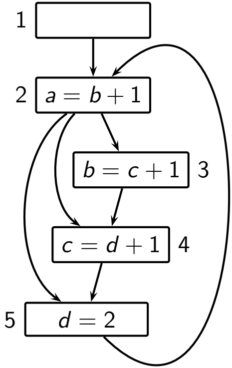
Part 10

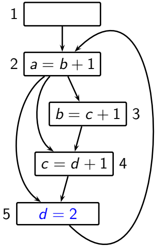
Precise Modelling of General Flows

Complexity of Constant Propagation?



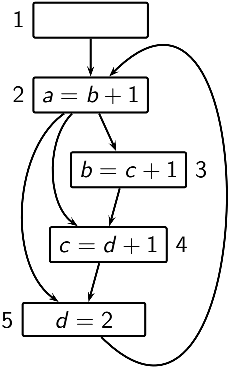
Complexity of Constant Propagation?

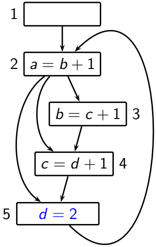




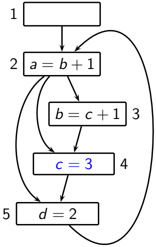
Iteration #1

Complexity of Constant Propagation?



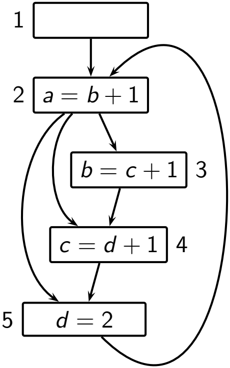


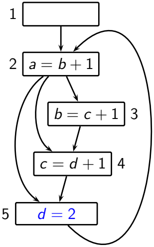
Iteration #1



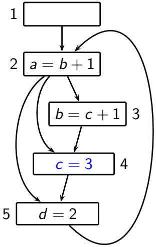
Iteration #2

Complexity of Constant Propagation?

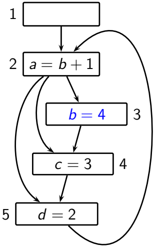




Iteration #1

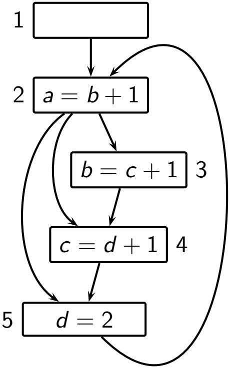


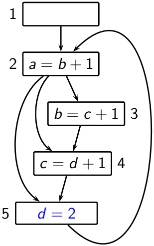
Iteration #2



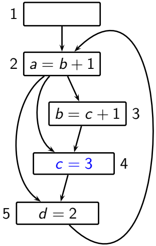
Iteration #3

Complexity of Constant Propagation?

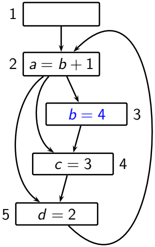




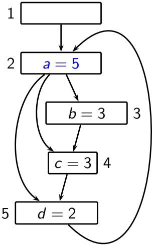
Iteration #1



Iteration #2



Iteration #3



Iteration #4

Part 11

Another View of Soundness and Precision

Two Views of Static Analysis

- Static analyis computes some approximtions of MoP
- An alternative view of static analysis

  Discovering paths satisfying a property
- In this part, we relate the two views
- We look at soundness and precision in these two views

Soundness and Precision in MoP View

What we have seen so far

- Soundness. Any value weaker than MoP is sound
- Precision. A sound value is more precise than another if it is "closer" to MoP

Examples of Conservative and Definite Information

- Liveness is uncertain (also called conservative)

  If a variable is declared live at a program point, it may or may not be used beyond that program point at run time

  (Why is it harmless if the variable is not actually used?)

- Deadness (i.e. absence of liveness) is certain (also called definite)

  If a variable is declared to be dead at a program point, it is guaranteed to be not used beyond that program point at run time

  (Why is it harmful if the variable is not actually dead?)

An Example Program

```
int main()
{ int a, b, c, n;

  a = 4;
  b = 2;
  c = 3;
  n = c*2;
  while (a <= n)
  {
    a = a+1;
  }
  if (a < 12)
    a = a+b+c;
   return a;
}
```

An Example Program

```
int main()
{ int a, b, c, n;

  a = 4;
  b = 2;
  c = 3;
  n = c*2;
  while (a <= n)
  {
    a = a+1;
  }
  if (a < 12)
    a = a+b+c;
   return a;
}
```

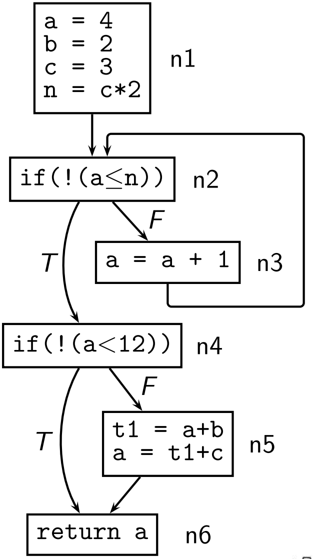
```
 1. a = 4
 2. b = 2
 3. c = 3
 4. n = c*2
 5. if (!(a≤n))
          goto 8
 6. a = a + 1
 7. goto 5
 8. if (!(a<12))
          goto 11
 9. t1 = a+b
10. a = t1+c
11. return a
```

An Example Program

```
int main()
{ int a, b, c, n;

  a = 4;
  b = 2;
  c = 3;
  n = c*2;
  while (a <= n)
  {
    a = a+1;
  }
  if (a < 12)
    a = a+b+c;
   return a;
}
```

```
 1. a = 4
 2. b = 2
 3. c = 3
 4. n = c*2
 5. if (!(a≤n))
          goto 8
 6. a = a + 1
 7. goto 5
 8. if (!(a<12))
          goto 11
 9. t1 = a+b
10. a = t1+c
11. return a
```



Motivating Example for Introducing Soundness and Precision

Example Program

```
int a;
int f(int b)
{  int c;
   c = a%2;
   b = - abs(b);
   while (b < c)
      b = b+1;
   if (b > 0)
      b = 0;
   return b;
}
```

Control Flow Graph

Motivating Example for Introducing Soundness and Precision

Example Program

Absolute

```
int a;
int f(int b)
{   int c;
    c = a%2;
    b = - abs(b);
    while (b < c)
        b = b+1;
    if (b > 0)
        b = 0;
    return b;
}
```

Control Flow Graph

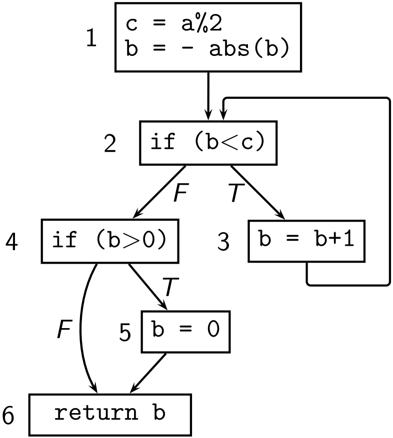
Motivating Example for Introducing Soundness and Precision

Example Program

Absolute

```
int a;
int f(int b)
{   int c;
    c = a%2;
    b = - abs(b);
    while (b < c)
        b = b+1;
    if (b > 0)
        b = 0;
    return b;
}
```

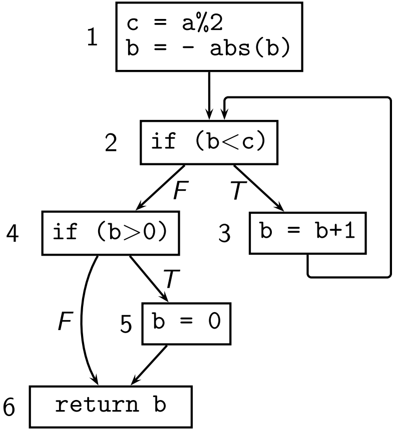
Control Flow Graph



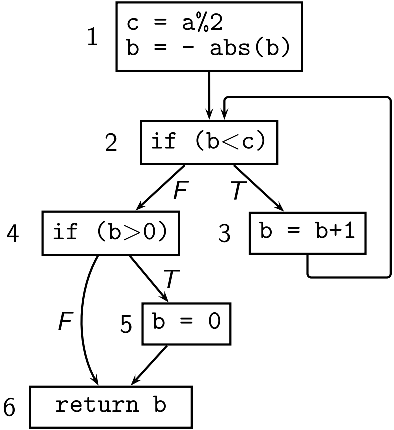
Execution Traces for Concrete Semantics (1)

- States
  - A *data* state: Variables $\rightarrow$ Values
  - A *program* state: (Program Point, A data state)
- Execution traces (or traces, for short)
  - Valid sequences of program states starting with a given initial state

Execution Traces for Concrete Semantics (2)



Execution Traces for Concrete Semantics (2)

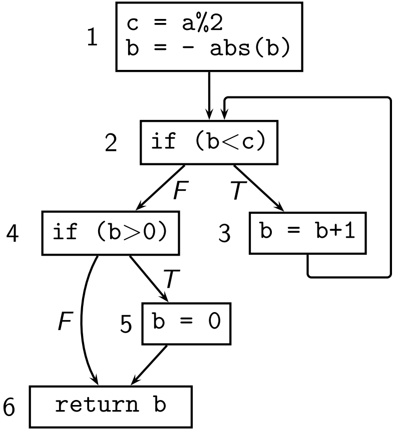


Trace 1

$a \quad b \quad c$

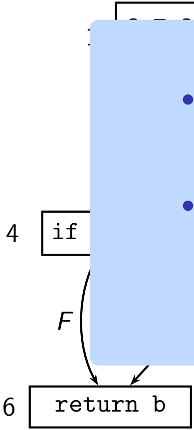
$Entry_1, (5, 2, 7)$
$Entry_2, (5, -2, 1)$
$Entry_3, (5, -2, 1)$
$Entry_2, (5, -1, 1)$
$Entry_3, (5, -1, 1)$
$Entry_2, (5, 0, 1)$
$Entry_3, (5, 0, 1)$
$Entry_2, (5, 1, 1)$
$Entry_4, (5, 1, 1)$
$Entry_5, (5, 1, 1)$
$Entry_6, (5, 0, 1)$

Execution Traces for Concrete Semantics (2)



Trace 1	a b c	*Trace 2*	a b c
$Entry_1$,	$(5, 2, 7)$	$Entry_1$,	$(-5, -2, 8)$
$Entry_2$,	$(5, -2, 1)$	$Entry_2$,	$(-5, -2, -1)$
$Entry_3$,	$(5, -2, 1)$	$Entry_3$,	$(-5, -2, -1)$
$Entry_2$,	$(5, -1, 1)$	$Entry_2$,	$(-5, -1, -1)$
$Entry_3$,	$(5, -1, 1)$	$Entry_4$,	$(-5, -1, -1)$
$Entry_2$,	$(5, 0, 1)$	$Entry_6$,	$(-5, -1, -1)$
$Entry_3$,	$(5, 0, 1)$		
$Entry_2$,	$(5, 1, 1)$		
$Entry_4$,	$(5, 1, 1)$		
$Entry_5$,	$(5, 1, 1)$		
$Entry_6$,	$(5, 0, 1)$		

Execution Traces for Concrete Semantics (2)



- A separate trace for each combination of inputs
  - The number of traces is potentially infinite
- Program points may repeat in the traces
  - Traces may be very long
  - Non-terminating traces: Infinitely long

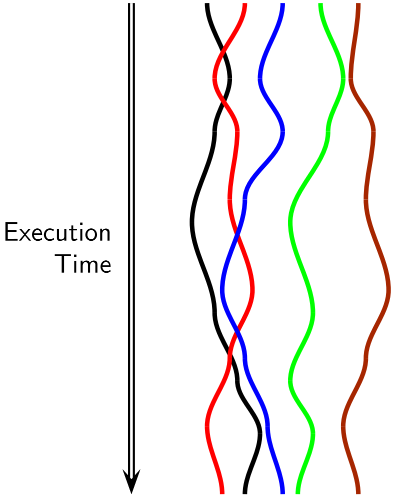
$Entry_5, (5, 1, 1)$
$Entry_6, (5, 0, 1)$

Abstract States

A static analysis computes abstract states

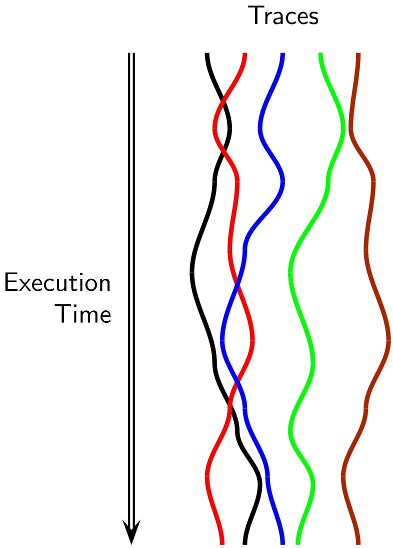
- The values are abstract values and are decided by the analysis
- An analysis may record values for other program entities such as expressions, statements, procedures etc.

A Pictorial Representation of Execution Traces

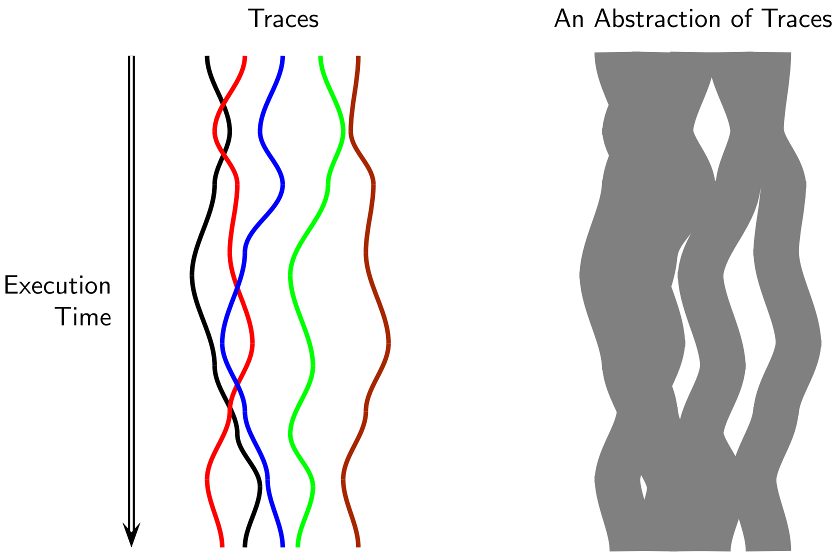


- The X-axis shows the states
- The Y-axis shows the execution time

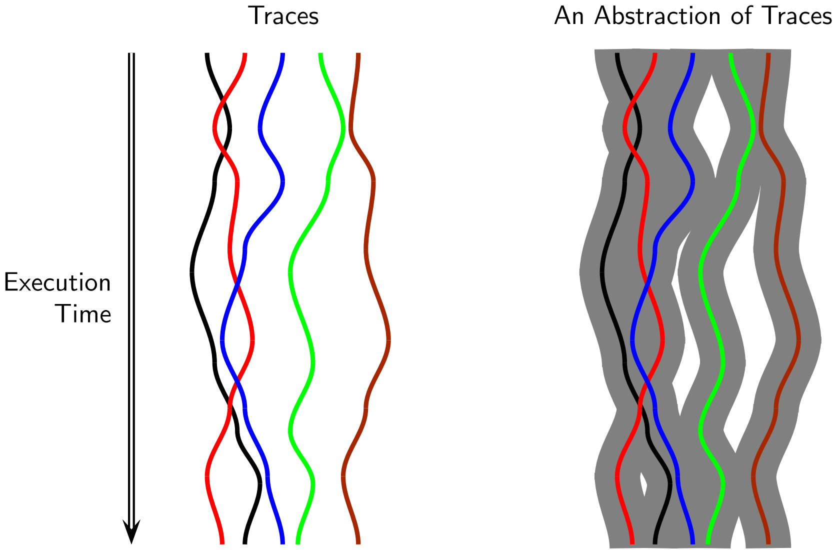
Static Analysis Computes Abstractions of Traces (1)



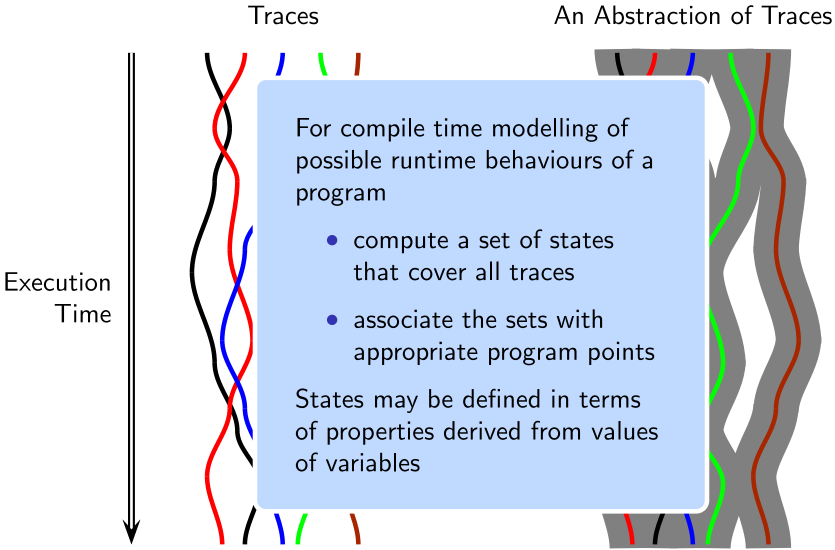
Static Analysis Computes Abstractions of Traces (1)



Static Analysis Computes Abstractions of Traces (1)



Static Analysis Computes Abstractions of Traces (1)



For compile time modelling of possible runtime behaviours of a program

- compute a set of states that cover all traces
- associate the sets with appropriate program points

States may be defined in terms of properties derived from values of variables

Static Analysis Computes Abstractions of Traces (2)

A possible static abstraction using sets

Trace 1

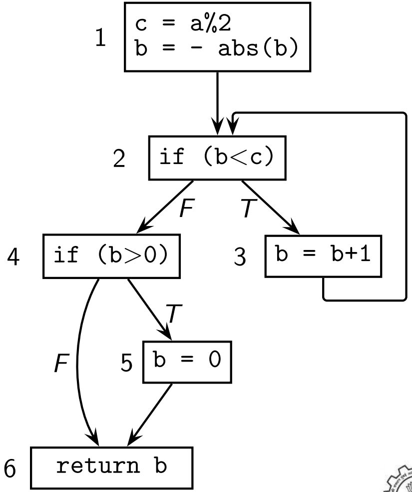
$a \; b \; c$

$Entry_1, (5, 2, 7)$
$Entry_2, (5, -2, 1)$
$Entry_3, (5, -2, 1)$
$Entry_2, (5, -1, 1)$
$Entry_3, (5, -1, 1)$
$Entry_2, (5, 0, 1)$
$Entry_3, (5, 0, 1)$
$Entry_2, (5, 1, 1)$
$Entry_4, (5, 1, 1)$
$Entry_5, (5, 1, 1)$
$Entry_6, (5, 0, 1)$

Trace 2

$a \; b \; c$

$Entry_1, (-5, -2, 8)$
$Entry_2, (-5, -2, -1)$
$Entry_3, (-5, -2, -1)$
$Entry_2, (-5, -1, -1)$
$Entry_4, (-5, -1, -1)$
$Entry_6, (-5, -1, -1)$



Static Analysis Computes Abstractions of Traces (2)

A possible static abstraction using sets

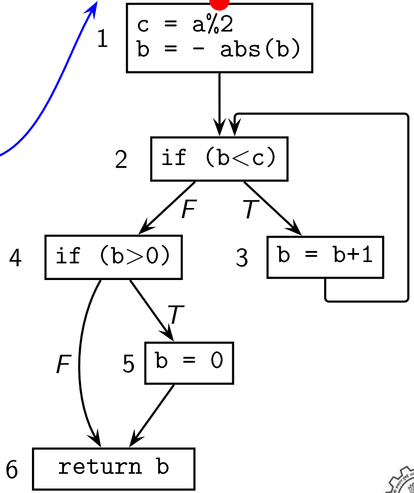
$\{(5,2,7),(-5,-2,8)\}$

Trace 1

	a b c
$Entry_1$,	$(5,2,7)$
$Entry_2$,	$(5,-2,1)$
$Entry_3$,	$(5,-2,1)$
$Entry_2$,	$(5,-1,1)$
$Entry_3$,	$(5,-1,1)$
$Entry_2$,	$(5,0,1)$
$Entry_3$,	$(5,0,1)$
$Entry_2$,	$(5,1,1)$
$Entry_4$,	$(5,1,1)$
$Entry_5$,	$(5,1,1)$
$Entry_6$,	$(5,0,1)$

Trace 2

	a b c
$Entry_1$,	$(-5,-2,8)$
$Entry_2$,	$(-5,-2,-1)$
$Entry_3$,	$(-5,-2,-1)$
$Entry_2$,	$(-5,-1,-1)$
$Entry_4$,	$(-5,-1,-1)$
$Entry_6$,	$(-5,-1,-1)$



Static Analysis Computes Abstractions of Traces (2)

A possible static abstraction using sets

$a = \{-5, 5\}, b = \{-2, 2\}, c = \{7, 8\}$

Trace 1

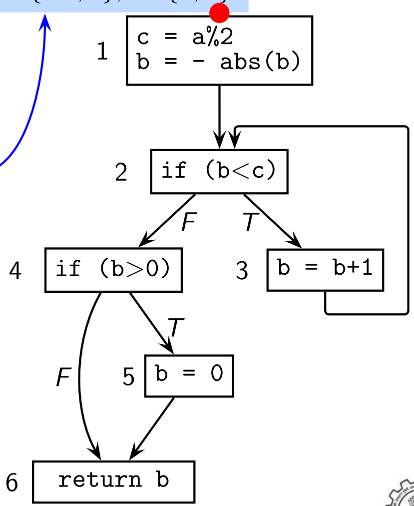
$a \quad b \quad c$

$Entry_1, (5, 2, 7)$
$Entry_2, (5, -2, 1)$
$Entry_3, (5, -2, 1)$
$Entry_2, (5, -1, 1)$
$Entry_3, (5, -1, 1)$
$Entry_2, (5, 0, 1)$
$Entry_3, (5, 0, 1)$
$Entry_2, (5, 1, 1)$
$Entry_4, (5, 1, 1)$
$Entry_5, (5, 1, 1)$
$Entry_6, (5, 0, 1)$

Trace 2

$a \quad b \quad c$

$Entry_1, (-5, -2, 8)$
$Entry_2, (-5, -2, -1)$
$Entry_3, (-5, -2, -1)$
$Entry_2, (-5, -1, -1)$
$Entry_4, (-5, -1, -1)$
$Entry_6, (-5, -1, -1)$



Static Analysis Computes Abstractions of Traces (2)

A possible static abstraction using sets

$a = \{-5, 5\}, b = \{-2, 2\}, c = \{7, 8\}$

We only show the values of b

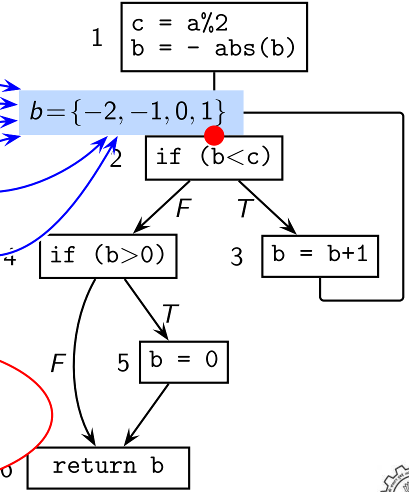
Trace 1

	a	b	c
$Entry_1$	5	2	7
$Entry_2$	5	−2	1
$Entry_3$	5	−2	1
$Entry_2$	5	−1	1
$Entry_3$	5	−1	1
$Entry_2$	5	0	1
$Entry_3$	5	0	1
$Entry_2$	5	1	1
$Entry_4$	5	1	1
$Entry_5$	5	1	1
$Entry_6$	5	0	1

Trace 2

	a	b	c
$Entry_1$	−5	−2	8
$Entry_2$	−5	−2	−1
$Entry_3$	−5	−2	−1
$Entry_2$	−5	−1	−1
$Entry_4$	−5	−1	−1
$Entry_6$	−5	−1	−1

$b = \{-2, -1, 0, 1\}$



Combine the values across all occurrences of a program point

Static Analysis Computes Abstractions of Traces (2)

A possible static abstraction using sets

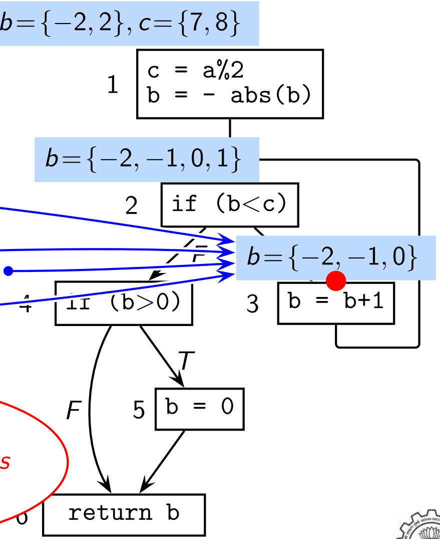
We only show the values of b

Trace 1

	a b c
$Entry_1$,	$(5, 2, 7)$
$Entry_2$,	$(5, -2, 1)$
$Entry_3$,	$(5, -2, 1)$
$Entry_2$,	$(5, -1, 1)$
$Entry_3$,	$(5, -1, 1)$
$Entry_2$,	$(5, 0, 1)$
$Entry_3$,	$(5, 0, 1)$
$Entry_2$,	$(5, 1, 1)$
$Entry_4$,	$(5, 1, 1)$
$Entry_5$,	$(5, 1, 1)$
$Entry_6$,	$(5, 0, 1)$

Trace 2

	a b c
$Entry_1$,	$(-5, -2, 8)$
$Entry_2$,	$(-5, -2, -1)$
$Entry_3$,	$(-5, -2, -1)$
$Entry_2$,	$(-5, -1, -1)$
$Entry_4$,	$(-5, -1, -1)$
$Entry_6$,	$(-5, -1, -1)$



Combine the values across all occurrences of a program point

Static Analysis Computes Abstractions of Traces (2)

A possible static abstraction using sets

Trace 1

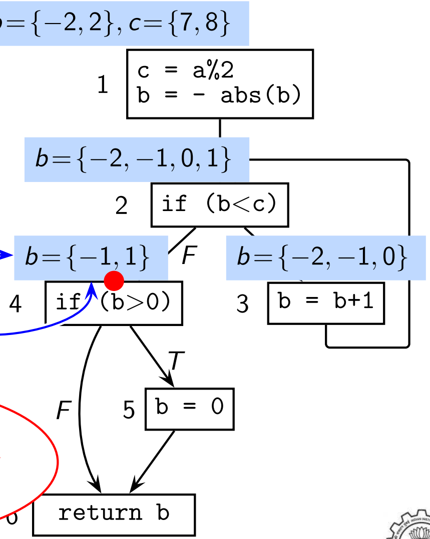
	a	b	c
$Entry_1$	5	2	7
$Entry_2$	5	−2	1
$Entry_3$	5	−2	1
$Entry_2$	5	−1	1
$Entry_3$	5	−1	1
$Entry_2$	5	0	1
$Entry_3$	5	0	1
$Entry_2$	5	1	1
$Entry_4$	5	1	1
$Entry_5$	5	1	1
$Entry_6$	5	0	1

We only show the values of b

Trace 2

	a	b	c
$Entry_1$	−5	−2	8
$Entry_2$	−5	−2	−1
$Entry_3$	−5	−2	−1
$Entry_2$	−5	−1	−1
$Entry_4$	−5	−1	−1
$Entry_6$	−5	−1	−1

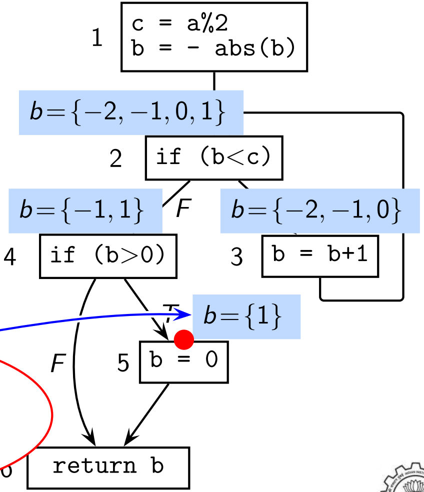
Combine the values across all occurrences of a program point



Static Analysis Computes Abstractions of Traces (2)

A possible static abstraction using sets

$a = \{-5, 5\}, b = \{-2, 2\}, c = \{7, 8\}$



Trace 1

	a	b	c
$Entry_1$,	(5,	2,	7)
$Entry_2$,	(5,	−2,	1)
$Entry_3$,	(5,	−2,	1)
$Entry_2$,	(5,	−1,	1)
$Entry_3$,	(5,	−1,	1)
$Entry_2$,	(5,	0,	1)
$Entry_3$,	(5,	0,	1)
$Entry_2$,	(5,	1,	1)
$Entry_4$,	(5,	1,	1)
$Entry_5$,	(5,	1,	1)
$Entry_6$,	(5,	0,	1)

We only show the values of b

Trace 2

	a	b	c
$Entry_1$,	(−5,	−2,	8)
$Entry_2$,	(−5,	−2,	−1)
$Entry_3$,	(−5,	−2,	−1)
$Entry_2$,	(−5,	−1,	−1)
$Entry_4$,	(−5,	−1,	−1)
$Entry_6$,	(−5,	−1,	−1)

Combine the values across all occurrences of a program point

Static Analysis Computes Abstractions of Traces (2)

A possible static abstraction using sets

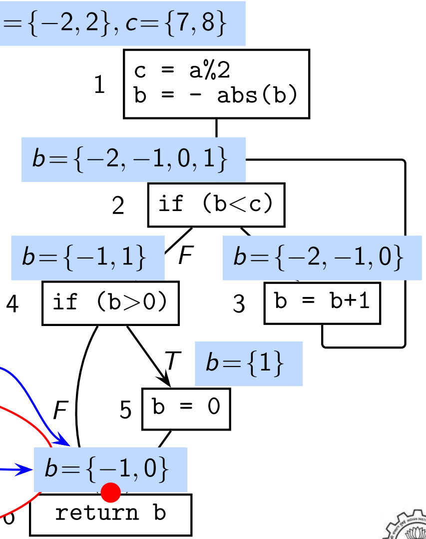
We only show the values of b

Trace 1

	a	b	c
$Entry_1$	5	2	7
$Entry_2$	5	−2	1
$Entry_3$	5	−2	1
$Entry_2$	5	−1	1
$Entry_3$	5	−1	1
$Entry_2$	5	0	1
$Entry_3$	5	0	1
$Entry_2$	5	1	1
$Entry_4$	5	1	1
$Entry_5$	5	1	1
$Entry_6$	5	0	1

Trace 2

	a	b	c
$Entry_1$	−5	−2	8
$Entry_2$	−5	−2	−1
$Entry_3$	−5	−2	−1
$Entry_2$	−5	−1	−1
$Entry_4$	−5	−1	−1
$Entry_6$	−5	−1	−1



Combine the values across all occurrences of a program point

Computing Static Abstraction for Liveness of Variables

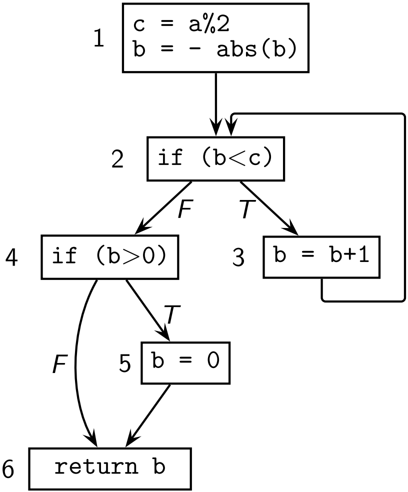
At a program point p
$a \mapsto 1 \Rightarrow a$ is live at p
$a \mapsto 0 \Rightarrow a$ is not live at p

Trace 1

	a b c
$Entry_1$,	$(1,1,0)$
$Entry_2$,	$(0,1,1)$
$Entry_3$,	$(0,1,1)$
$Entry_2$,	$(0,1,1)$
$Entry_3$,	$(0,1,1)$
$Entry_2$,	$(0,1,1)$
$Entry_3$,	$(0,1,1)$
$Entry_2$,	$(0,1,1)$
$Entry_4$,	$(0,1,0)$
$Entry_5$,	$(0,0,0)$
$Entry_6$,	$(0,1,0)$

Trace 2

	a b c
$Entry_1$,	$(1,1,0)$
$Entry_2$,	$(0,1,1)$
$Entry_3$,	$(0,0,1)$
$Entry_2$,	$(0,1,1)$
$Entry_4$,	$(0,1,0)$
$Entry_6$,	$(0,1,0)$



Computing Static Abstraction for Liveness of Variables

At a program point p

$a \mapsto 1 \Rightarrow a$ is live at p

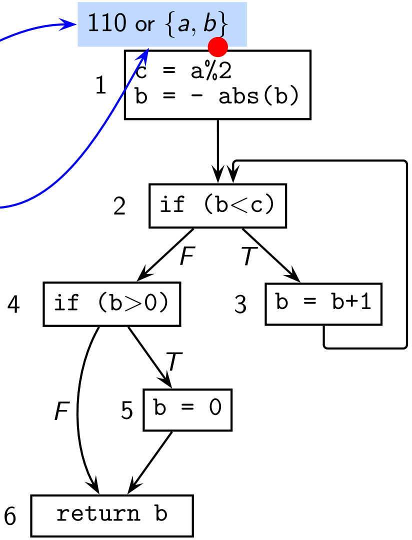
$a \mapsto 0 \Rightarrow a$ is not live at p

Trace 1

	a	b	c
$Entry_1$	1	1	0
$Entry_2$	0	1	1
$Entry_3$	0	1	1
$Entry_2$	0	1	1
$Entry_3$	0	1	1
$Entry_2$	0	1	1
$Entry_3$	0	1	1
$Entry_2$	0	1	1
$Entry_4$	0	1	0
$Entry_5$	0	0	0
$Entry_6$	0	1	0

Trace 2

	a	b	c
$Entry_1$	1	1	0
$Entry_2$	0	1	1
$Entry_3$	0	0	1
$Entry_2$	0	1	1
$Entry_4$	0	1	0
$Entry_6$	0	1	0



Computing Static Abstraction for Liveness of Variables

At a program point p

$a \mapsto 1 \Rightarrow a$ is live at p

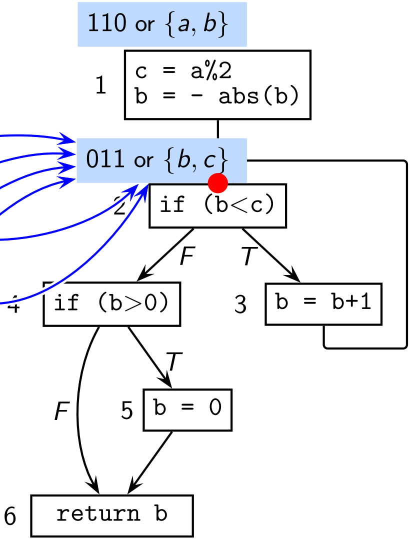
$a \mapsto 0 \Rightarrow a$ is not live at p

Trace 1

	a	b	c
$Entry_1$	1	1	0
$Entry_2$	0	1	1
$Entry_3$	0	1	1
$Entry_2$	0	1	1
$Entry_3$	0	1	1
$Entry_2$	0	1	1
$Entry_3$	0	1	1
$Entry_2$	0	1	1
$Entry_4$	0	1	0
$Entry_5$	0	0	0
$Entry_6$	0	1	0

Trace 2

	a	b	c
$Entry_1$	1	1	0
$Entry_2$	0	1	1
$Entry_3$	0	0	1
$Entry_2$	0	1	1
$Entry_4$	0	1	0
$Entry_6$	0	1	0



Computing Static Abstraction for Liveness of Variables

At a program point p

$a \mapsto 1 \Rightarrow a$ is live at p

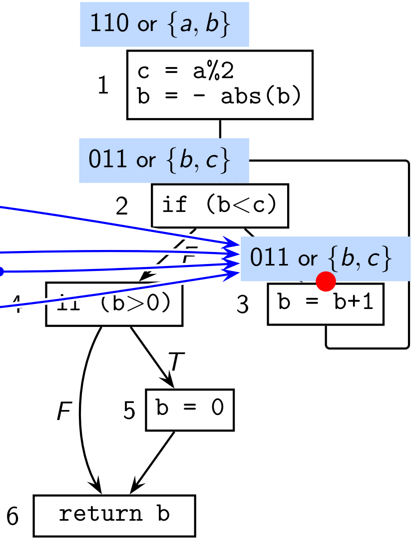
$a \mapsto 0 \Rightarrow a$ is not live at p

Trace 1

	a b c
$Entry_1$,	$(1, 1, 0)$
$Entry_2$,	$(0, 1, 1)$
$Entry_3$,	$(0, 1, 1)$
$Entry_2$,	$(0, 1, 1)$
$Entry_3$,	$(0, 1, 1)$
$Entry_2$,	$(0, 1, 1)$
$Entry_3$,	$(0, 1, 1)$
$Entry_2$,	$(0, 1, 1)$
$Entry_4$,	$(0, 1, 0)$
$Entry_5$,	$(0, 0, 0)$
$Entry_6$,	$(0, 1, 0)$

Trace 2

	a b c
$Entry_1$,	$(1, 1, 0)$
$Entry_2$,	$(0, 1, 1)$
$Entry_3$,	$(0, 0, 1)$
$Entry_2$,	$(0, 1, 1)$
$Entry_4$,	$(0, 1, 0)$
$Entry_6$,	$(0, 1, 0)$



Computing Static Abstraction for Liveness of Variables

At a program point p

$a \mapsto 1 \Rightarrow a$ is live at p

$a \mapsto 0 \Rightarrow a$ is not live at p

Trace 1

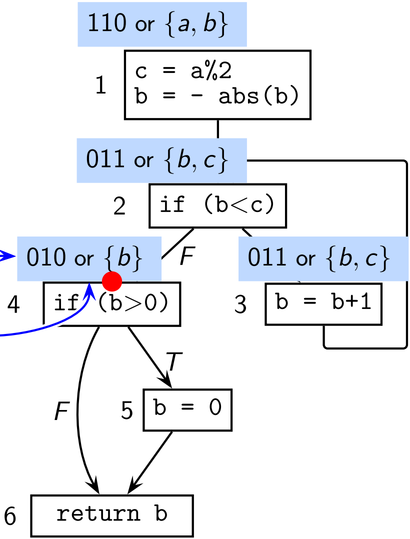
$a \; b \; c$

$Entry_1, (1,1,0)$
$Entry_2, (0,1,1)$
$Entry_3, (0,1,1)$
$Entry_2, (0,1,1)$
$Entry_3, (0,1,1)$
$Entry_2, (0,1,1)$
$Entry_3, (0,1,1)$
$Entry_2, (0,1,1)$
$Entry_4, (0,1,0)$
$Entry_5, (0,0,0)$
$Entry_6, (0,1,0)$

Trace 2

$a \; b \; c$

$Entry_1, (1,1,0)$
$Entry_2, (0,1,1)$
$Entry_3, (0,0,1)$
$Entry_2, (0,1,1)$
$Entry_4, (0,1,0)$
$Entry_6, (0,1,0)$



Computing Static Abstraction for Liveness of Variables

At a program point p

$a \mapsto 1 \Rightarrow a$ is live at p

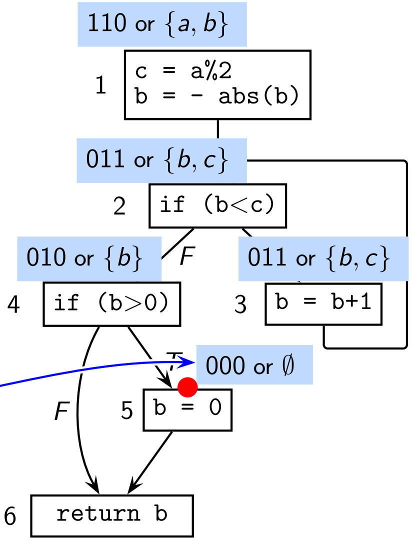
$a \mapsto 0 \Rightarrow a$ is not live at p

Trace 1

	a	b	c
$Entry_1$	1	1	0
$Entry_2$	0	1	1
$Entry_3$	0	1	1
$Entry_2$	0	1	1
$Entry_3$	0	1	1
$Entry_2$	0	1	1
$Entry_3$	0	1	1
$Entry_2$	0	1	1
$Entry_4$	0	1	0
$Entry_5$	0	0	0
$Entry_6$	0	1	0

Trace 2

	a	b	c
$Entry_1$	1	1	0
$Entry_2$	0	1	1
$Entry_3$	0	0	1
$Entry_2$	0	1	1
$Entry_4$	0	1	0
$Entry_6$	0	1	0



Computing Static Abstraction for Liveness of Variables

At a program point p

$a \mapsto 1 \Rightarrow a$ is live at p

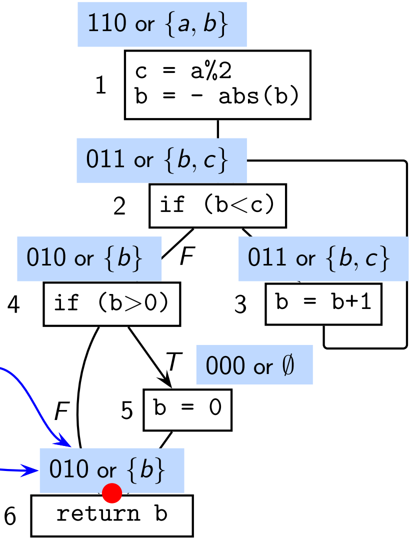
$a \mapsto 0 \Rightarrow a$ is not live at p

Trace 1

	a b c
$Entry_1$,	$(1, 1, 0)$
$Entry_2$,	$(0, 1, 1)$
$Entry_3$,	$(0, 1, 1)$
$Entry_2$,	$(0, 1, 1)$
$Entry_3$,	$(0, 1, 1)$
$Entry_2$,	$(0, 1, 1)$
$Entry_3$,	$(0, 1, 1)$
$Entry_2$,	$(0, 1, 1)$
$Entry_4$,	$(0, 1, 0)$
$Entry_5$,	$(0, 0, 0)$
$Entry_6$,	$(0, 1, 0)$

Trace 2

	a b c
$Entry_1$,	$(1, 1, 0)$
$Entry_2$,	$(0, 1, 1)$
$Entry_3$,	$(0, 0, 1)$
$Entry_2$,	$(0, 1, 1)$
$Entry_4$,	$(0, 1, 0)$
$Entry_6$,	$(0, 1, 0)$



Computing Static Abstraction for Liveness of Variables

At a program point p
$a \mapsto 1 \Rightarrow a$ is live at p
$a \mapsto 0 \Rightarrow a$ is not live at p

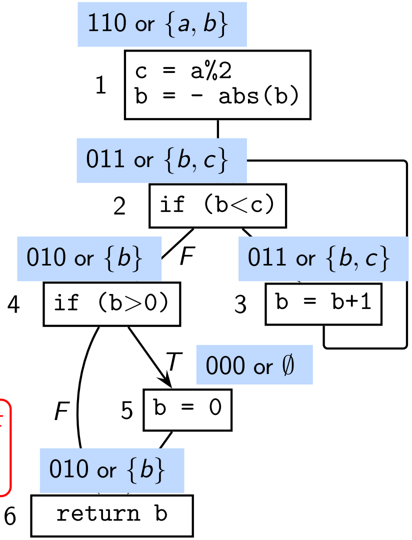
Trace 1

	a	b	c
$Entry_1$	1	1	0
$Entry_2$	0	1	1
$Entry_3$	0	1	1
$Entry_2$	0	1	1
$Entry_3$	0	1	1
$Entry_2$	0	1	1
$Entry_3$	0	1	1
$Entry_2$	0	1	1
$Entry_4$	0	1	0
$Entry_5$	0	0	0
$Entry_6$	0	1	0

Trace 2

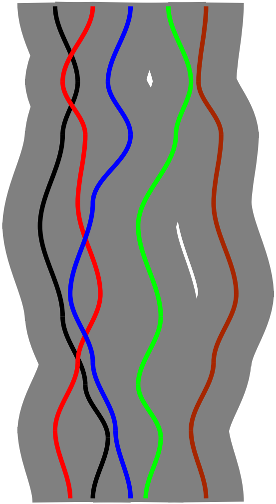
	a	b	c
$Entry_1$	1	1	0
$Entry_2$	0	1	1
$Entry_3$	0	0	1
$Entry_2$	0	1	1
$Entry_4$	0	1	0
$Entry_6$	0	1	0

Trace 2 does not add anything to the abstraction



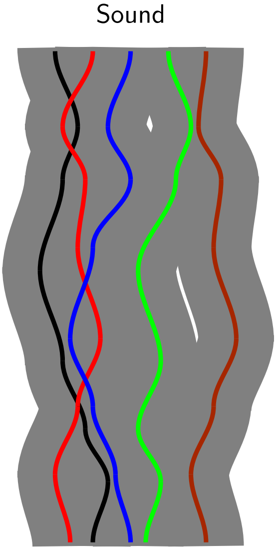
Soundness of Abstractions (1)

Sound

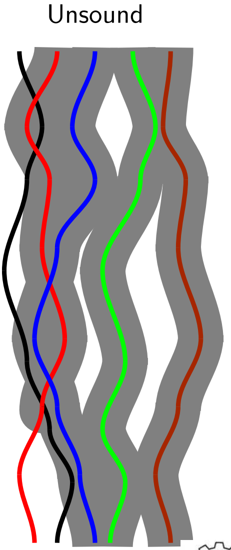


- An over-approximation of traces is sound

Soundness of Abstractions (1)

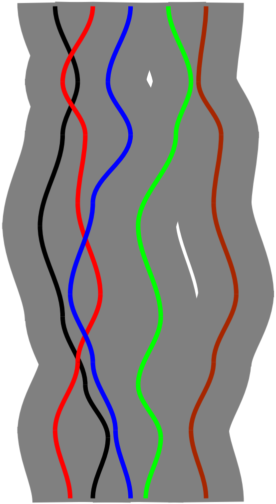


- An over-approximation of traces is sound
- Missing any state in any trace causes unsoundness



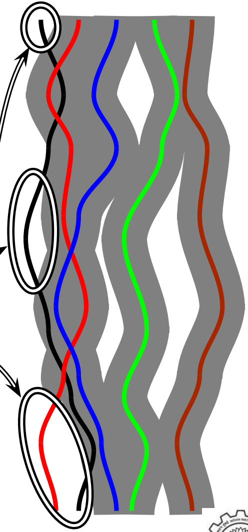
Soundness of Abstractions (1)

Sound



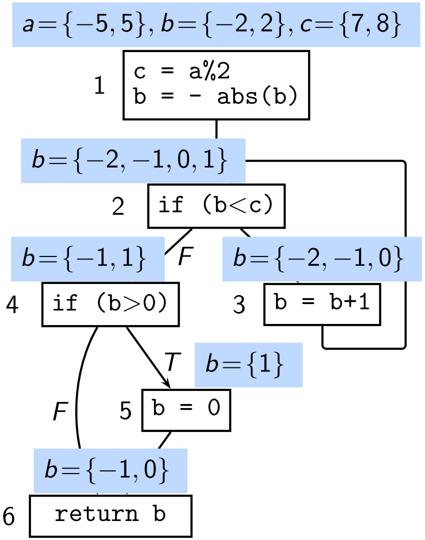
- An over-approximation of traces is sound
- Missing any state in any trace causes unsoundness

Unsound



Soundness of Abstractions (2)

An unsound abstraction



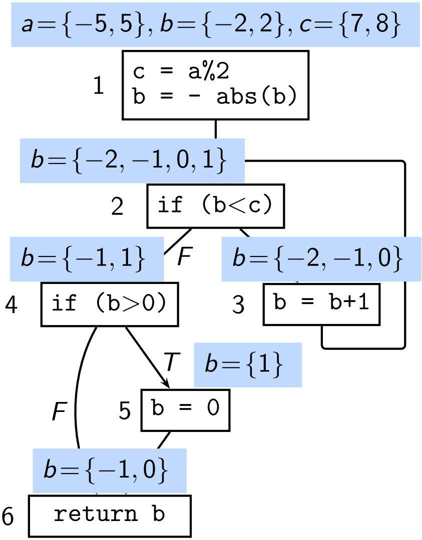
All variables can have arbitrary values at the start.

b can have many more values at the entry of

- blocks 2 and 3 (e.g. -3, -8, . . .)
- block 4 (e.g. 0)

Soundness of Abstractions (2)

An unsound abstraction



A sound abstraction using intervals

- Over-approximated range of values denoted by

$$\Big[\mathit{low\_limit}, \mathit{high\_limit}\Big]$$

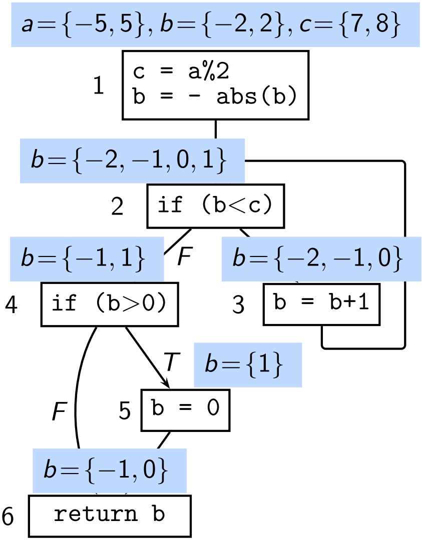
- Inclusive limits with

$$\mathit{low\_limit} \leq \mathit{high\_limit}$$

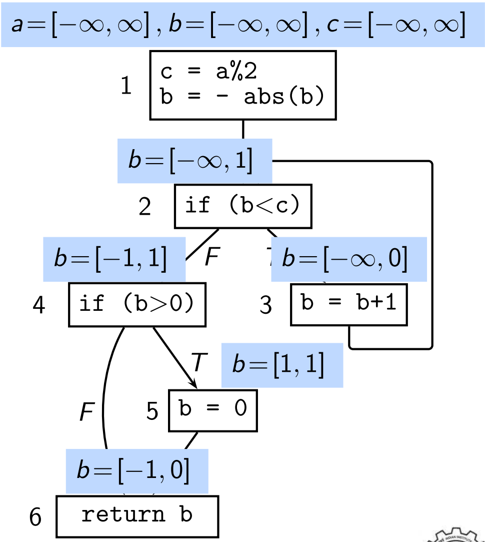
- One contiguous range per variable with no "holes"

Soundness of Abstractions (2)

An unsound abstraction

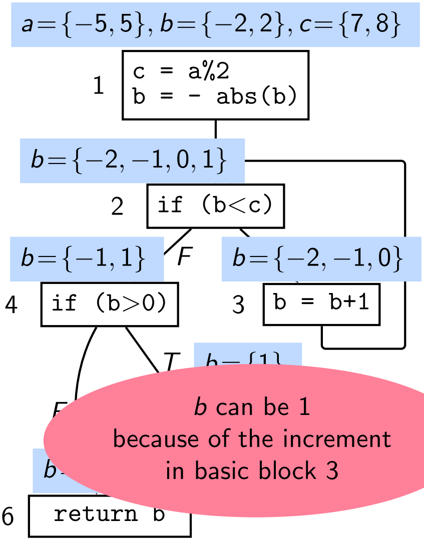


A sound abstraction using intervals



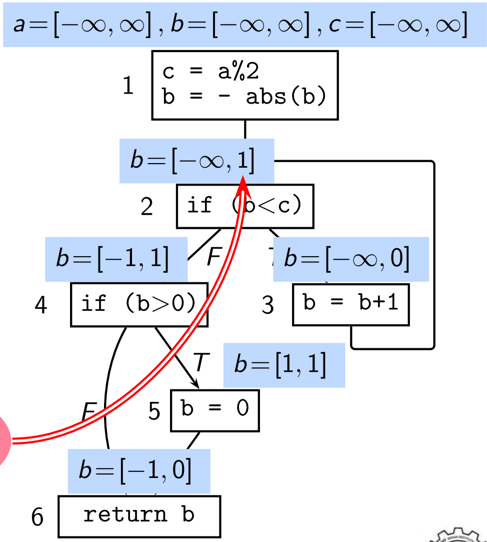
Soundness of Abstractions (2)

An unsound abstraction



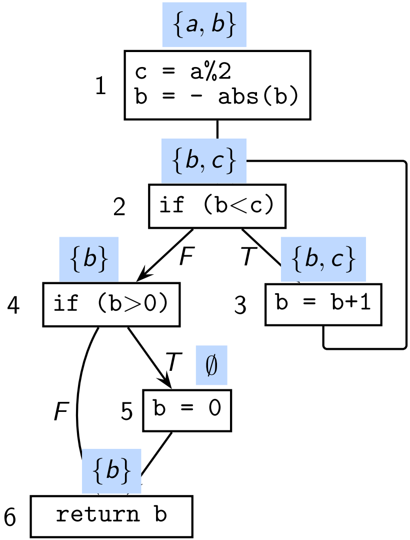
b can be 1
because of the increment
in basic block 3

A sound abstraction using intervals

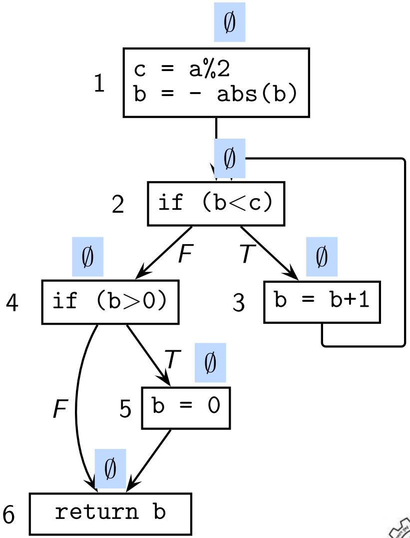


Soundness of Abstractions for Liveness Analysis

A sound abstraction

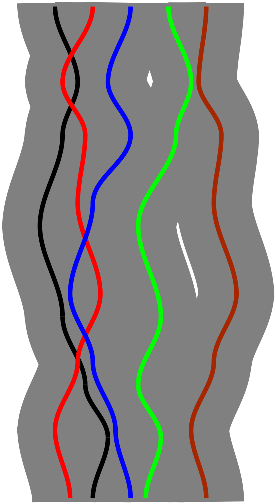


An unsound abstraction



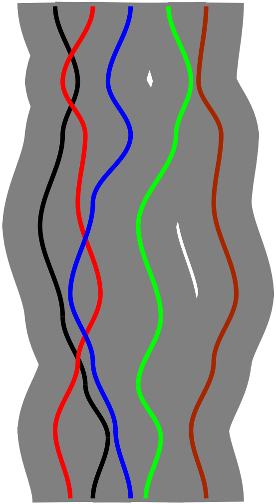
Precision of Sound Abstractions(1)

Sound but imprecise

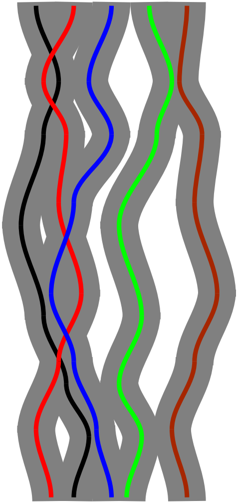


Precision of Sound Abstractions(1)

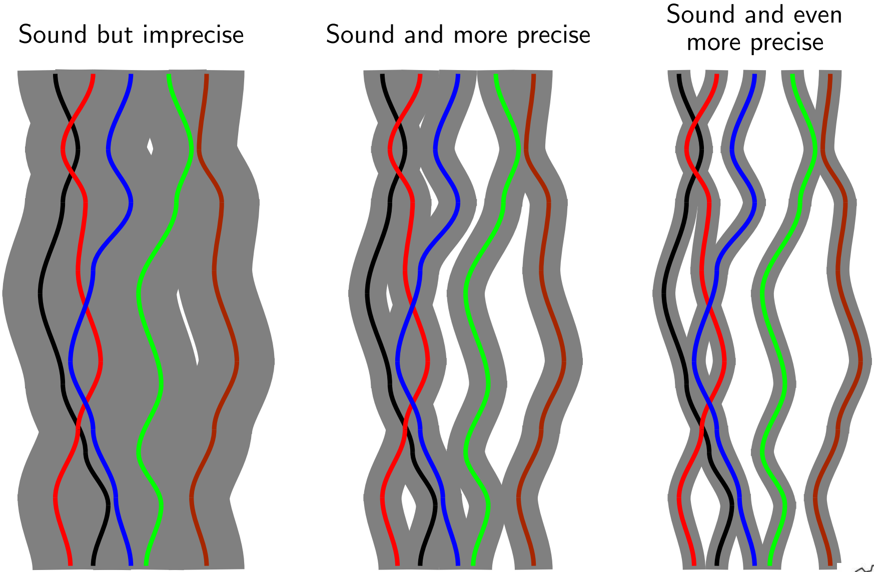
Sound but imprecise



Sound and more precise

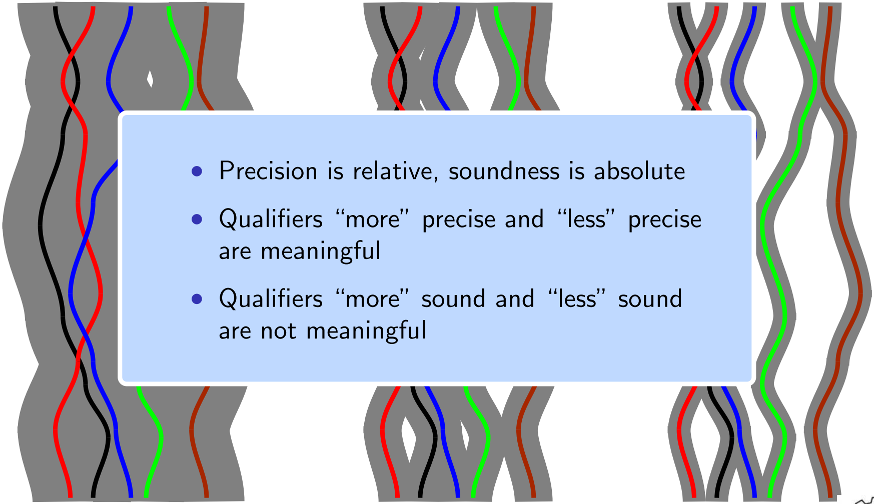


Precision of Sound Abstractions(1)



Precision of Sound Abstractions(1)

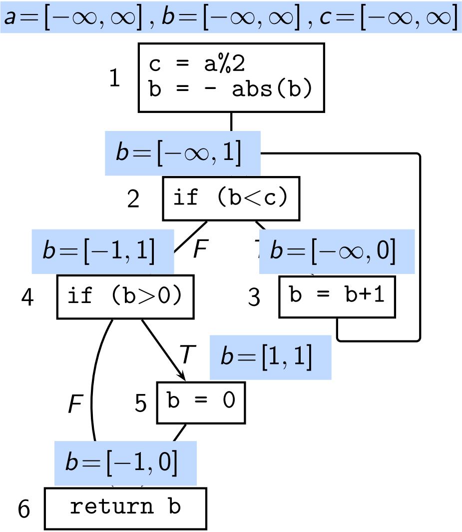
Sound but imprecise | Sound and more precise | Sound and even more precise



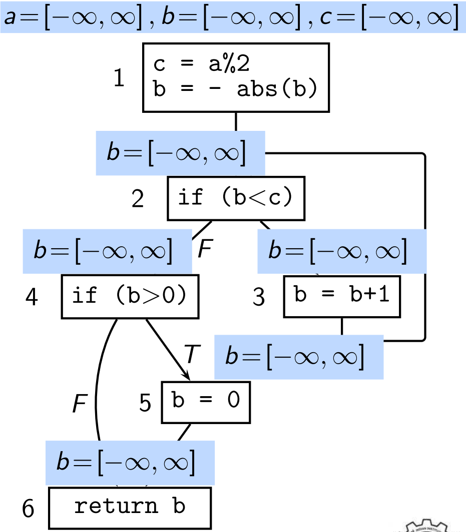
- Precision is relative, soundness is absolute
- Qualifiers "more" precise and "less" precise are meaningful
- Qualifiers "more" sound and "less" sound are not meaningful

Precision of Sound Abstractions(2)

A precise abstraction using intervals

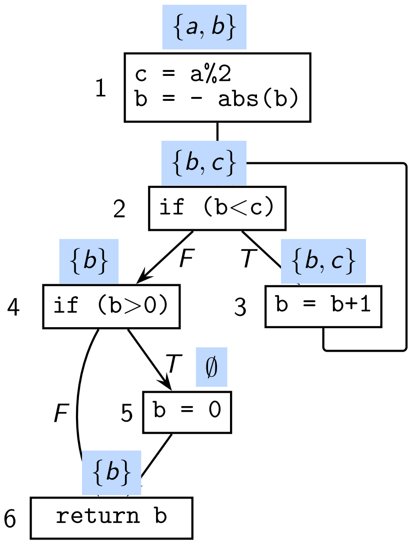


An imprecise abstraction using intervals

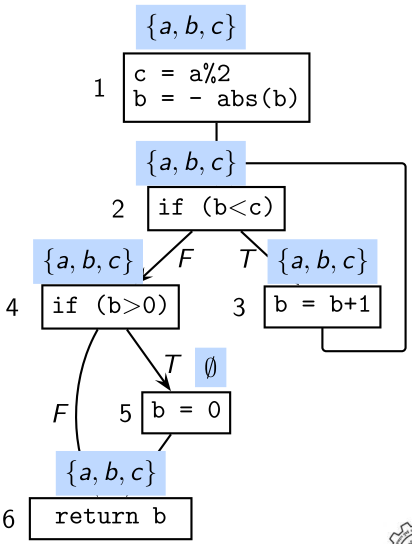


Precision of Abstractions for Liveness Analysis

A precise abstraction



An imprecise abstraction



Other Terminologies

- We have talked about Soundness and Precision in the context of conventional program analysis
- Some other terminologies from program analysis in other contexts
  - From the world or verification and validation/testing
    True Positive, False Positive, True Negative, and False Negative
    Precision and Recall
  - From the world of logic
    Soundness and Completeness

All of them are related to our notions of Soundness and Precision

MoP View and Property-Based View of Program Analysis

- Let the set of control flow paths involving a program point n be defined as

$$Paths(n) = \left\{ (\overrightarrow{\rho}, n, \overleftarrow{\rho}) \mid \begin{array}{l} \overrightarrow{\rho} \text{ is a path from START to } n, \\ \overleftarrow{\rho} \text{ is a path from } n \text{ to END} \end{array} \right\}$$

$\rho \in Paths(n)$ generically denotes $\overrightarrow{\rho}$ or $\overleftarrow{\rho}$ as may be appropriate

- The goal of an analysis A is to compute

$$\forall n,\ A_n = \prod_{\rho_n \in Paths(n)} f_{\rho_n}(BI) \qquad \text{(Conventional MoP View)}$$

$$\forall n,\ A_n \Leftrightarrow \exists \rho \in Paths(n) \text{ s.t. } p(\rho) \qquad \text{(Propery-based view)}$$

where predicate p defines a path property

- Some times we need a property q to hold along every path

In such a situation, we choose $p(\rho) = \neg q(\rho)$

Examples of Path Properties

- Properties that hold at program point n along SOME path
  - Live Variables Analysis. Variable v is live
  - Range Analysis. Variable v has a value between the range $[low, high]$
  - MAY Points-to Analysis. Pointer w holds the address of location l

Choose p as the same property

Examples of Path Properties

- Properties that hold at program point n along SOME path
  - Live Variables Analysis. Variable v is live
  - Range Analysis. Variable v has a value between the range $[low, high]$
  - MAY Points-to Analysis. Pointer w holds the address of location l

  Choose p as the same property

- Properties that hold at program point n along EVERY path
  - Available Expressions Analysis. Expression $a * b$ is available
  - MUST Points-to Analysis. Pointer w holds the address of location l
  - Constant Propagation. Variable v has a constant value c

Examples of Path Properties

- Properties that hold at program point n along SOME path
  - Live Variables Analysis. Variable v is live
  - Range Analysis. Variable v has a value between the range $[low, high]$
  - MAY Points-to Analysis. Pointer w holds the address of location l

  Choose p as the same property

- Properties that hold at program point n along EVERY path
  - Available Expressions Analysis. Expression $a * b$ is available
    Choose p as: Expression $a * b$ is NOT available
  - MUST Points-to Analysis. Pointer w holds the address of location l
    Choose p as: Pointer w DOES NOT hold the address of location l
  - Constant Propagation. Variable v has a constant value c
    Choose p as: Variable v DOES NOT have a constant value c

Definiteness and Conservativeness

- The goal of an analysis A is to compute

$$\forall n, A_n \Leftrightarrow \; \exists \rho \in \mathit{Paths}(n) \text{ s.t. } \; p(\rho)$$

  where predicate p defines a path property

- Due to the SOME path property

Definiteness and Conservativeness

- The goal of an analysis A is to compute

$$\forall n, A_n \Leftrightarrow \boxed{\exists \rho \in Paths(n)} \text{ s.t. } p(\rho)$$

where predicate p defines a path property

- Due to the SOME path property

$$\begin{aligned} A_n &= \text{true is } \textcolor{blue}{\text{conservative}} && \equiv \text{may not hold along every path} \\ A_n &= \text{false is } \textcolor{blue}{\text{definite}} && \equiv \text{guaranteed to hold along every path} \end{aligned}$$

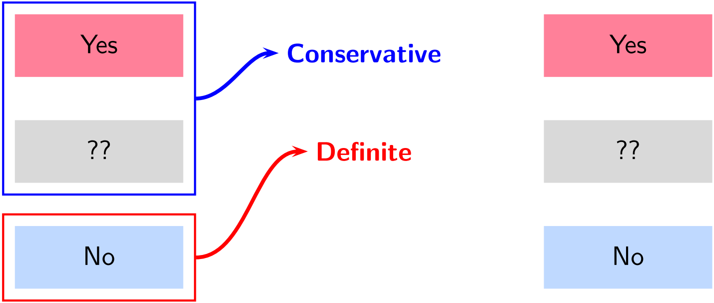
Relating Definiteness and Conservativeness to Some Path Property

Live Variables Analysis Existential property (Satisfied by some path)		Available Expressions Analysis Universal property (Satisfied by every path)
Yes	**Conservative**	Yes
??	**Definite**	??
No		No

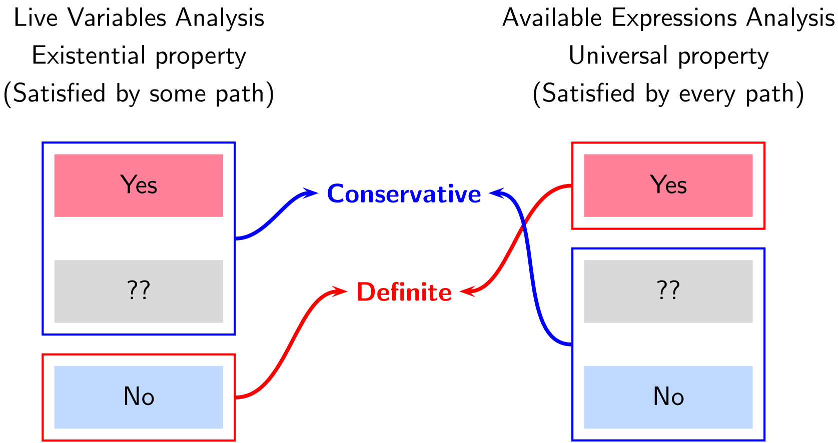
Relating Definiteness and Conservativeness to Some Path Property

Live Variables Analysis
Existential property
(Satisfied by some path)

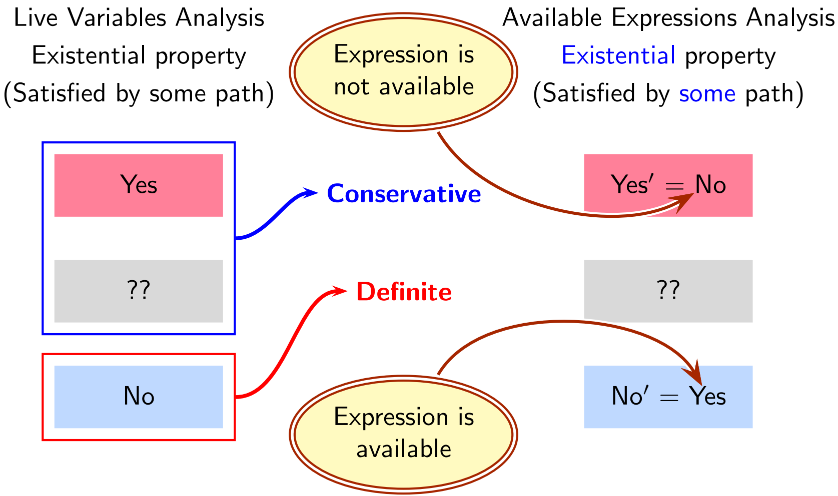
Available Expressions Analysis
Universal property
(Satisfied by every path)



Relating Definiteness and Conservativeness to Some Path Property



Relating Definiteness and Conservativeness to Some Path Property



Relating Definiteness and Conservativeness to Some Path Property

Live Variables Analysis

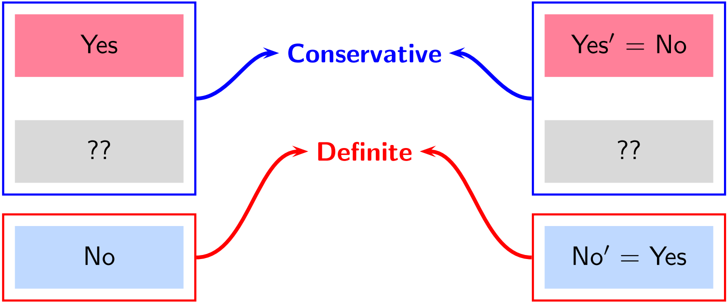
Existential property

(Satisfied by some path)

Available Expressions Analysis

Existential property

(Satisfied by some path)



Examples of Definiteness and Conservativeness

Conservative Properties	Definite Properties
Variable x is live	Variable x is NOT live
Pointer w points to x (under MAY Points-to Analysis)	Pointer w does NOT point to x (under MAY Points-to Analysis)
Variable x has a value in the range $[2, 30]$	Variable x does NOT have a value in the range $[2, 30]$
Expression $a * b$ is NOT available	Expression $a * b$ is available
Pointer w does NOT point to x (under MUST Points-to Analysis)	Pointer w points to x (under MUST Points-to Analysis)
Variable x does NOT have value 4	Variable x has value 4

$\{$True, False$\} \times \{$Positive, Negative$\}$

	Analysis finds that A_n holds	Analysis finds that A_n does not hold
A_n holds	True Positive	False Negative
A_n does not hold	False Positive	True Negative

$\{$**True**, **False**$\} \times \{$**Positive**, **Negative**$\}$

	Analysis finds that A_n holds	Analysis finds that A_n does not hold
A_n holds	True Positive	False Negative
A_n does not hold	False Positive	True Negative

$\{$True, False$\} \times \{$Positive, Negative$\}$

	Analysis finds that A_n holds	Analysis finds that A_n does not hold
A_n holds	True Positive	False Negative
A_n does not hold	False Positive	True Negative

{True, False} × {Positive, Negative}

	Analysis finds that A_n holds	Analysis finds that A_n does not hold
A_n holds	True Positive	False Negative
A_n does not hold	False Positive	True Negative

No mismatch between prediction and reality

{True, False} × {Positive, Negative}

	Analysis finds that A_n holds	Analysis finds that A_n does not hold
A_n holds	True Positive	False Negative
A_n does not hold	False Positive	True Negative

Mismatch between prediction and reality

$\{$True, False$\} \times \{$Positive, Negative$\}$

Harmless Error

	Analysis finds that A_n holds	Analysis finds that A_n does not hold
A_n holds	True Positive	False Negative
A_n does not hold	False Positive	True Negative

Mismatch between prediction and reality

{True, False} × {Positive, Negative}

Harmless Error

Harmful Error

	Analysis finds that A_n holds	Analysis finds that A_n does not hold
A_n holds	True Positive	False Negative
A_n does not hold	False Positive	True Negative

Mismatch between prediction and reality

$\{$True, False$\} \times \{$Positive, Negative$\}$

Harmless Error

Harmful Error

	Analysis finds that A_n holds	Analysis finds that A_n does not hold
A_n holds	True Positive	$\boxed{\text{False Negative}}$
A_n does not hold	$\boxed{\text{False Positive}}$	True Negative

Mismatch between prediction and reality

Examples of Imprecision and Unsoundness

Imprecision	Unsoundness
Variable x should not be live but is marked live	Variable x should live but is not marked live
Pointer w should not point to x but does point to x (under MAY Points-to Analysis)	Pointer w should point to x but does not point to x (under MAY Points-to Analysis)
Variable x should have a range $[5, 30]$ but has the range $[2, 40]$	Variable x should have a range $[5, 30]$ but has the range $[10, 20]$
Expression $a * b$ should be available but is not available	Expression $a * b$ should not be available but is available
Pointer w should to x but does not point to x (under MUST Points-to Analysis)	Pointer w should not point to x but does point to x (under MUST Points-to Analysis)
Variable x should have value 4 but has values 4 and 5	Variable x should have values 4 and 5 but has value 4

Relating to the Terminologies Used in Logic

- Terms used in Logic
  - A logic is sound if every theorem is a tautology
  - A logic is complete if every tautology is a theorem

- Terms used in Program Anlaysis

  Define *MUST information* as the run time information that holds in every execution path

  - An analysis is sound if definite information is MUST
  - An analysis is complete if MUST information is definite

- Completeness $\equiv$ Precision

  When MUST info is guaranteed to be definite, the analysis does not miss any definite information $\Rightarrow$ No False Positives

Part 12

Extra Topics

Tarski's Fixed Point Theorem

Given monotonic $f : L \to L$ where L is a complete lattice

Define

$$\begin{array}{lrcl} p \text{ is a fixed point of } f: & Fix(f) & = & \{p \mid f(p) = p\} \\ f \text{ is reductive at } p: & Red(f) & = & \{p \mid f(p) \sqsubseteq p\} \\ f \text{ is extensive at } p: & Ext(f) & = & \{p \mid f(p) \sqsupseteq p\} \end{array}$$

Then

$$\begin{array}{rcl} LFP(f) & = & \bigsqcap Red(f) \in Fix(f) \\ MFP(f) & = & \bigsqcup Ext(f) \in Fix(f) \end{array}$$

Tarski's Fixed Point Theorem

Given monotonic $f : L \to L$ where L is a complete lattice

Define

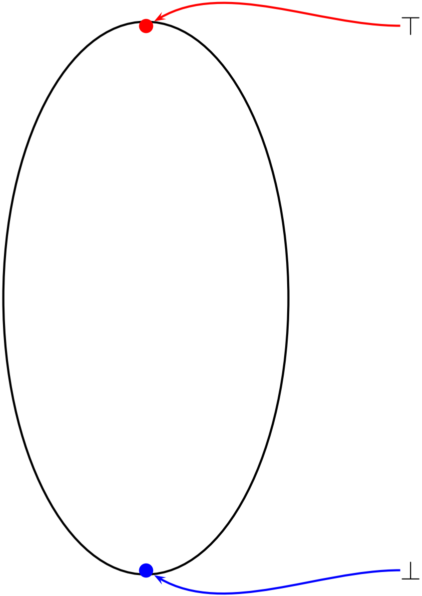
$$
\begin{array}{llcl}
p \text{ is a fixed point of } f: & Fix(f) & = & \{p \mid f(p) = p\} \\
f \text{ is reductive at } p: & Red(f) & = & \{p \mid f(p) \sqsubseteq p\} \\
f \text{ is extensive at } p: & Ext(f) & = & \{p \mid f(p) \sqsupseteq p\}
\end{array}
$$

Then

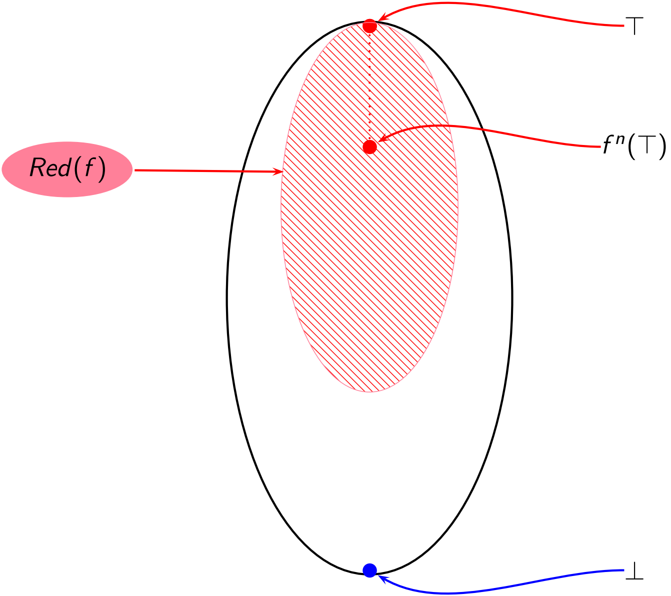
$$
\begin{array}{rcl}
LFP(f) & = & \bigsqcap Red(f) \in Fix(f) \\
MFP(f) & = & \bigsqcup Ext(f) \in Fix(f)
\end{array}
$$

Guarantees only existence, not computability of fixed points

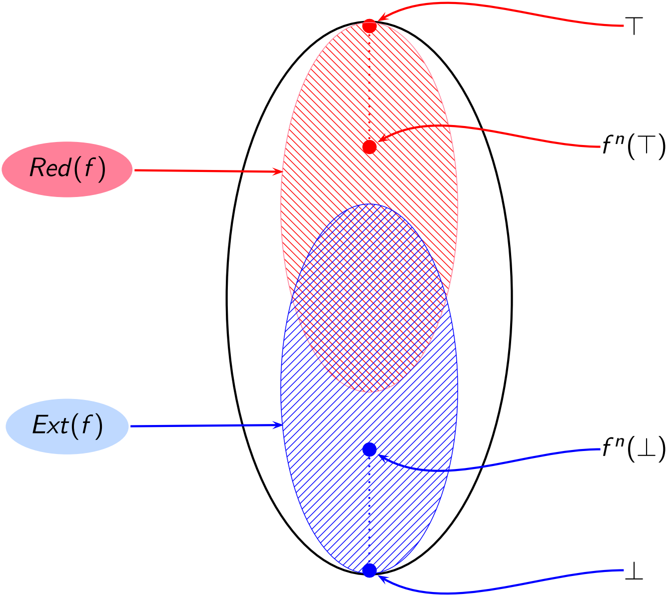
Fixed Points of a Function



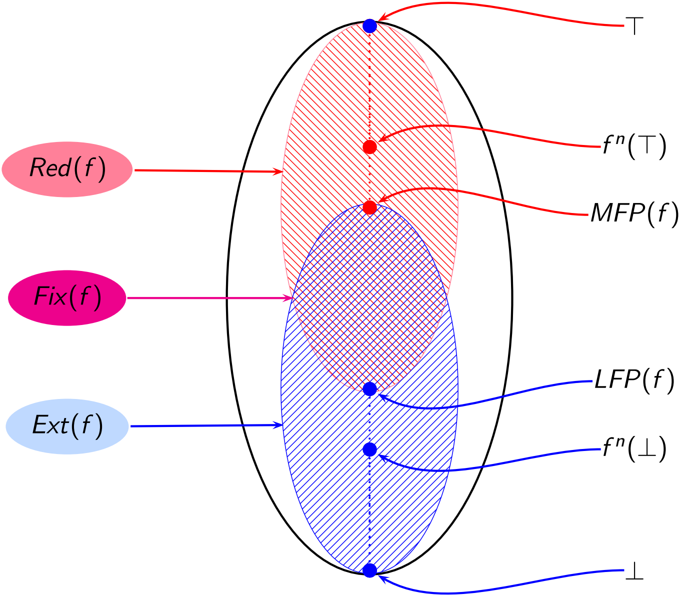
Fixed Points of a Function



Fixed Points of a Function

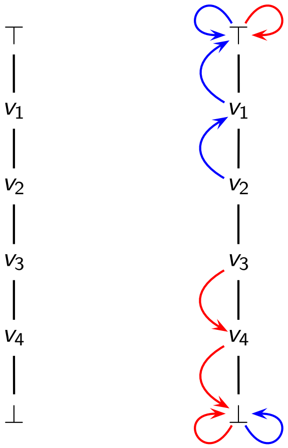


Fixed Points of a Function



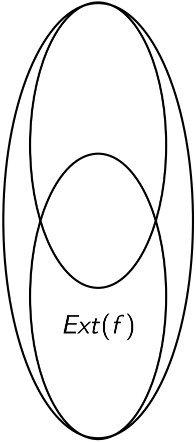
Examples of Reductive and Extensive Sets

Finite L Monotonic $f : L \to L$

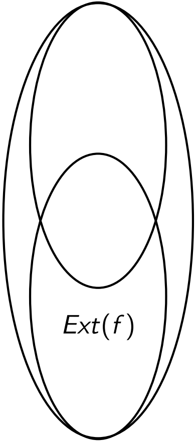


$$
\begin{aligned}
Red(f) &= \{\top, v_3, v_4, \bot\} \\
Ext(f) &= \{\top, v_1, v_2, \bot\} \\
Fix(f) &= Red(f) \cap Ext(f) \\
&= \{\top, \bot\} \\
MFP(f) &= lub\,(Ext(f)) \\
&= lub\,(Fix(f)) \\
&= \top \\
LFP(f) &= glb\,(Red(f)) \\
&= glb\,(Fix(f)) \\
&= \bot
\end{aligned}
$$

Existence of MFP: Proof of Tarski's Fixed Point Theorem

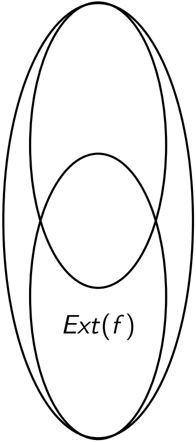


Existence of MFP: Proof of Tarski's Fixed Point Theorem



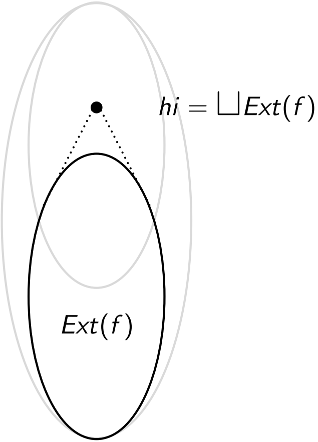
1. Claim 1: Let $X \subseteq L$.
   $\forall x \in X,\ p \sqsupseteq x \Rightarrow p \sqsupseteq \bigsqcup(X)$.

Existence of MFP: Proof of Tarski's Fixed Point Theorem



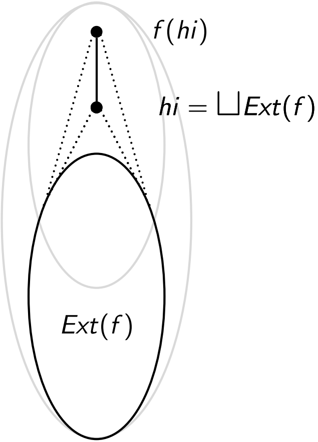
1. Claim 1: Let $X \subseteq L$.
   $\forall x \in X,\ p \sqsupseteq x \Rightarrow p \sqsupseteq \bigsqcup(X)$.
2. In the following we use $Ext(f)$ as X

Existence of MFP: Proof of Tarski's Fixed Point Theorem



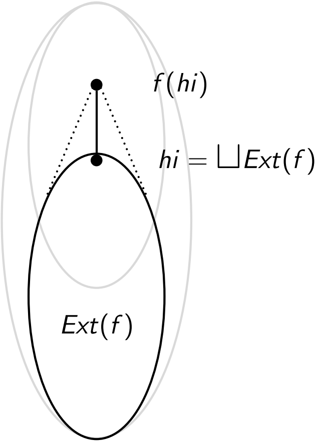
1. Claim 1: Let $X \subseteq L$.
   $\forall x \in X,\ p \sqsupseteq x \Rightarrow p \sqsupseteq \bigsqcup(X)$.
2. In the following we use $Ext(f)$ as X
3. $\forall p \in Ext(f),\ hi \sqsupseteq p$

Existence of MFP: Proof of Tarski's Fixed Point Theorem



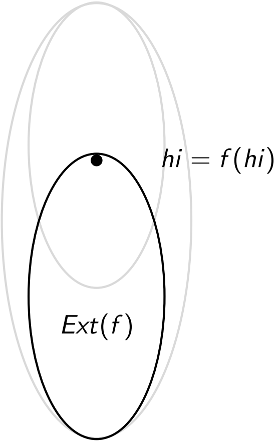
1. Claim 1: Let $X \subseteq L$.
   $\forall x \in X,\ p \sqsupseteq x \Rightarrow p \sqsupseteq \bigsqcup(X)$.
2. In the following we use $Ext(f)$ as X
3. $\forall p \in Ext(f),\ hi \sqsupseteq p$
4. $hi \sqsupseteq p \ \Rightarrow f(hi) \sqsupseteq f(p) \sqsupseteq p$ (monotonicity)
   $\Rightarrow f(hi) \sqsupseteq hi$ (claim 1)

Existence of MFP: Proof of Tarski's Fixed Point Theorem



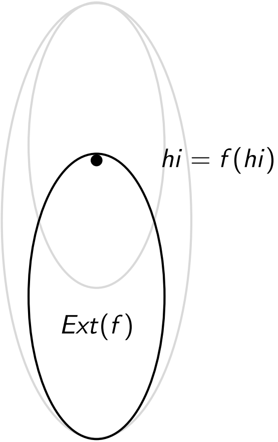
1. Claim 1: Let $X \subseteq L$.
   $\forall x \in X,\ p \sqsupseteq x \Rightarrow p \sqsupseteq \bigsqcup(X)$.
2. In the following we use $Ext(f)$ as X
3. $\forall p \in Ext(f),\ hi \sqsupseteq p$
4. $hi \sqsupseteq p \;\Rightarrow f(hi) \sqsupseteq f(p) \sqsupseteq p$ (monotonicity)
   $\Rightarrow f(hi) \sqsupseteq hi$ (claim 1)
5. f is extensive at hi also: $hi \in Ext(f)$

Existence of MFP: Proof of Tarski's Fixed Point Theorem



1. Claim 1: Let $X \subseteq L$.
   $\forall x \in X,\ p \sqsupseteq x \Rightarrow p \sqsupseteq \bigsqcup(X)$.
2. In the following we use $Ext(f)$ as X
3. $\forall p \in Ext(f),\ hi \sqsupseteq p$
4. $hi \sqsupseteq p \ \Rightarrow f(hi) \sqsupseteq f(p) \sqsupseteq p$ (monotonicity)
   $\Rightarrow f(hi) \sqsupseteq hi$ (claim 1)
5. f is extensive at hi also: $hi \in Ext(f)$
6. $f(hi) \sqsupseteq hi \Rightarrow f^2(hi) \sqsupseteq f(hi)$
   $\Rightarrow f(hi) \in Ext(f)$
   $\Rightarrow hi \sqsupseteq f(hi)$ (from 3)
   $\Rightarrow hi = f(hi) \Rightarrow hi \in Fix(f)$

Existence of MFP: Proof of Tarski's Fixed Point Theorem



1. Claim 1: Let $X \subseteq L$.
   $\forall x \in X,\ p \sqsupseteq x \Rightarrow p \sqsupseteq \bigsqcup(X)$.
2. In the following we use $Ext(f)$ as X
3. $\forall p \in Ext(f),\ hi \sqsupseteq p$
4. $hi \sqsupseteq p \ \Rightarrow f(hi) \sqsupseteq f(p) \sqsupseteq p$ (monotonicity)
   $\Rightarrow f(hi) \sqsupseteq hi$ (claim 1)
5. f is extensive at hi also: $hi \in Ext(f)$
6. $f(hi) \sqsupseteq hi \Rightarrow f^2(hi) \sqsupseteq f(hi)$
   $\Rightarrow f(hi) \in Ext(f)$
   $\Rightarrow hi \sqsupseteq f(hi)$ (from 3)
   $\Rightarrow hi = f(hi) \Rightarrow hi \in Fix(f)$
7. $Fix(f) \subseteq Ext(f)$ (by definition)
   $\Rightarrow hi \sqsupseteq p,\ \forall p \in Fix(f)$

Existence and Computation of the Maximum Fixed Point

- For monotonic $f : L \rightarrow L$

Existence and Computation of the Maximum Fixed Point

- For monotonic $f : L \rightarrow L$
  - Existence: $MFP(f) = \bigsqcup Ext(f) \in Fix(f)$
    Requires L to be complete

Existence and Computation of the Maximum Fixed Point

- For monotonic $f : L \to L$
  - Existence: $MFP(f) = \bigsqcup Ext(f) \in Fix(f)$
    Requires L to be complete
  - Computation: $MFP(f) = f^{k+1}(\top) = f^{k}(\top)$ such that $f^{j+1}(\top) \neq f^{j}(\top),\ j < k$.
    Requires all *strictly descending* chains to be finite

Existence and Computation of the Maximum Fixed Point

- For monotonic $f : L \rightarrow L$
  - Existence: $MFP(f) = \bigsqcup Ext(f) \in Fix(f)$
    Requires L to be complete
  - Computation: $MFP(f) = f^{k+1}(\top) = f^{k}(\top)$ such that $f^{j+1}(\top) \neq f^{j}(\top),\ j < k$.
    Requires all *strictly descending* chains to be finite
- Finite strictly descending and ascending chains

  $\Rightarrow$ Completeness of lattice

Existence and Computation of the Maximum Fixed Point

- For monotonic $f : L \rightarrow L$
  - Existence: $MFP(f) = \bigsqcup Ext(f) \in Fix(f)$
    Requires L to be complete
  - Computation: $MFP(f) = f^{k+1}(\top) = f^{k}(\top)$ such that $f^{j+1}(\top) \neq f^{j}(\top),\ j < k$.
    Requires all *strictly descending* chains to be finite
- Finite strictly descending and ascending chains
  $\Rightarrow$ Completeness of lattice
- Completeness of lattice $\not\Rightarrow$ Finite strictly descending chains

Existence and Computation of the Maximum Fixed Point

- For monotonic $f : L \rightarrow L$
  - Existence: $MFP(f) = \bigsqcup Ext(f) \in Fix(f)$
    Requires L to be complete
  - Computation: $MFP(f) = f^{k+1}(\top) = f^{k}(\top)$ such that $f^{j+1}(\top) \neq f^{j}(\top),\ j < k$.
    Requires all *strictly descending* chains to be finite
- Finite strictly descending and ascending chains
  $\Rightarrow$ Completeness of lattice
- Completeness of lattice $\not\Rightarrow$ Finite strictly descending chains
- $\Rightarrow$ Even if MFP exists, it may not be reachable unless all strictly descending chains are finite

Framework Properties Influencing Complexity

Depends on the loop closure properties of the framework

k-Bounded Frameworks

$$f^*(x) = x \sqcap f(x) \sqcap f^2(x) \sqcap \ldots \sqcap f^{k-1}(x)$$

Necessary and sufficient

Framework Properties Influencing Complexity

Depends on the loop closure properties of the framework

k-Bounded Frameworks

Fast Frameworks ($k = 2$)

$f^2(x) \sqsupseteq f(x) \sqcap x$

$f^*(x) = x \sqcap f(x) \sqcap f^2(x) \sqcap \ldots \sqcap f^{k-1}(x)$

Necessary and sufficient

Framework Properties Influencing Complexity

Depends on the loop closure properties of the framework

k-Bounded Frameworks

Fast Frameworks ($k = 2$)

Rapid Frameworks

$f^2(x) \sqsupseteq f(x)$

$f^2(x) \sqsupseteq f(x) \sqcap x$

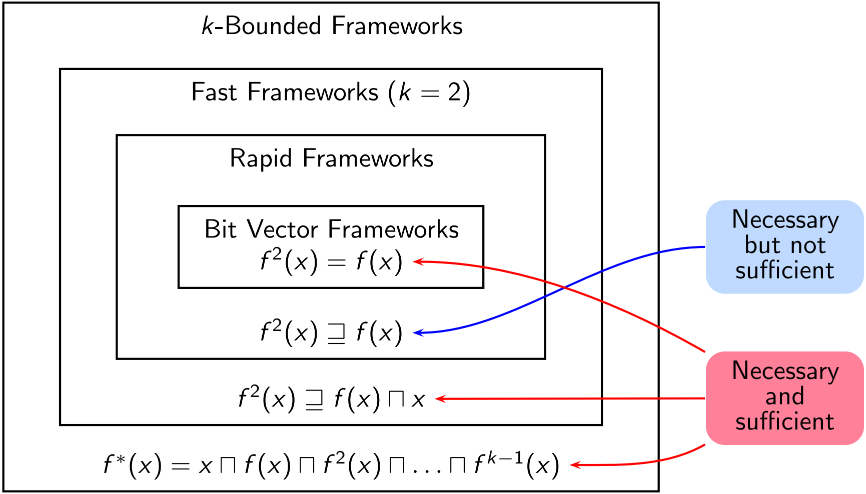
$f^*(x) = x \sqcap f(x) \sqcap f^2(x) \sqcap \ldots \sqcap f^{k-1}(x)$

Necessary but not sufficient

Necessary and sufficient

Framework Properties Influencing Complexity

Depends on the loop closure properties of the framework



Complexity of Round Robin Iterative Algorithm

- Unidirectional rapid frameworks

Task	Number of iterations	
	Irreducible G	Reducible G
Initialisation	1	1
Convergence (until *change* remains true)	$d(G,T)+1$	$d(G,T)$
Verifying convergence (*change* becomes false)	1	1