Theoretical Abstractions in Data Flow Analysis

Uday Khedker

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

August 2010

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.

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

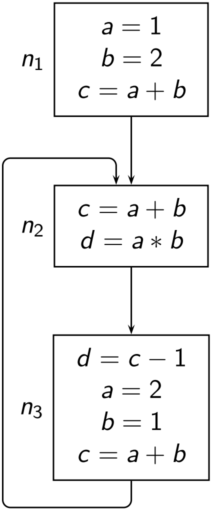
Part 2

The Need for a More General Setting

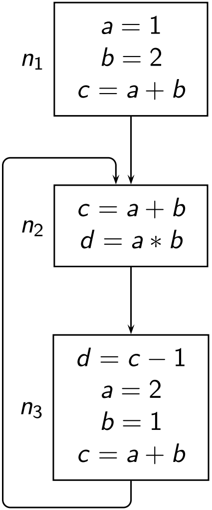
What We Have Seen So Far . . .

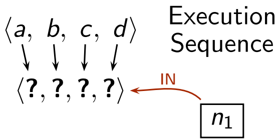
Analysis	Entity	Attribute at p	Paths	
Live variables	Variables	Use	Starting at p	Some
Available expressions	Expressions	Availability	Reaching p	All
Partially available expressions	Expressions	Availability	Reaching p	Some
Anticipable expressions	Expressions	Use	Starting at p	All
Reaching definitions	Definitions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

An Introduction to Constant Propagation

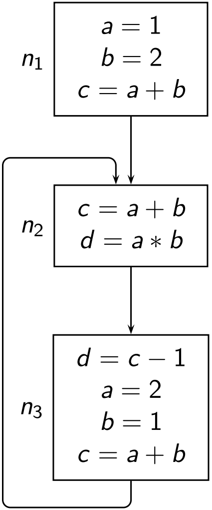


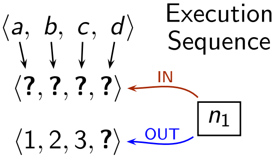
An Introduction to Constant Propagation



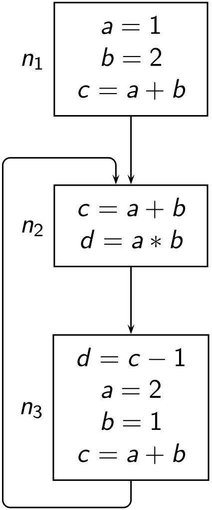


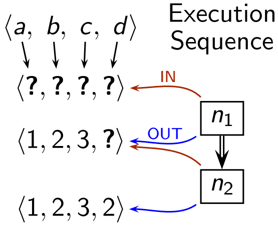
An Introduction to Constant Propagation



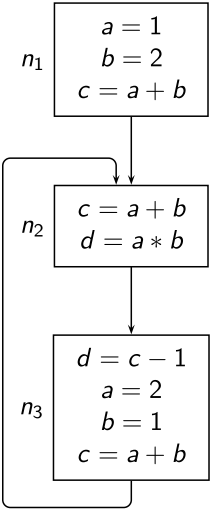


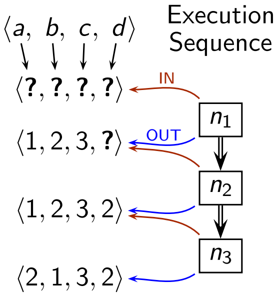
An Introduction to Constant Propagation



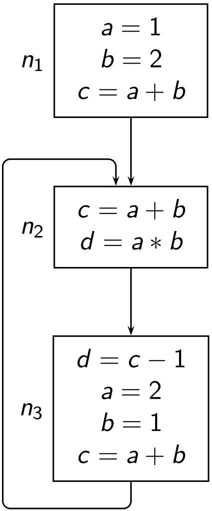


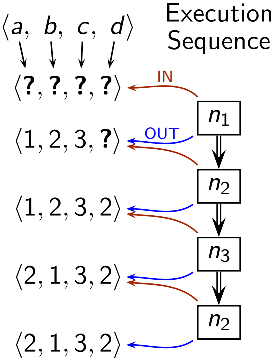
An Introduction to Constant Propagation



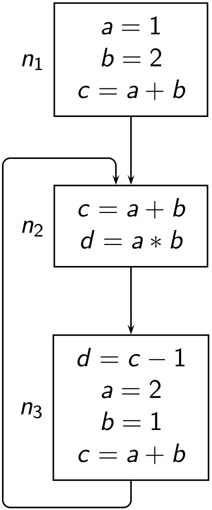


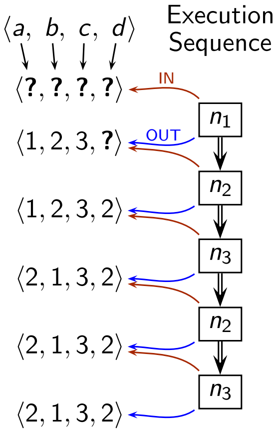
An Introduction to Constant Propagation



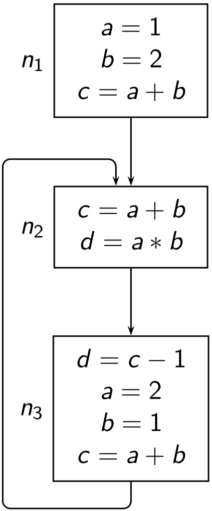


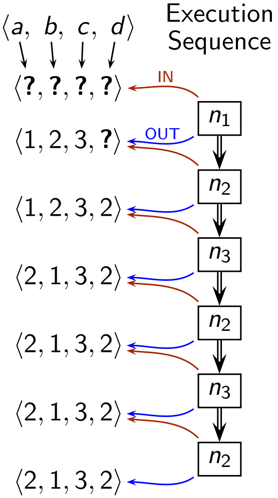
An Introduction to Constant Propagation



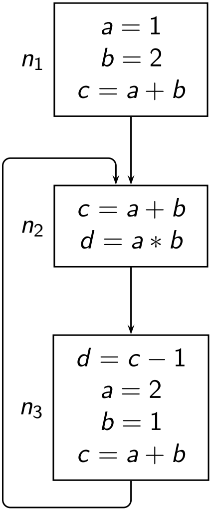


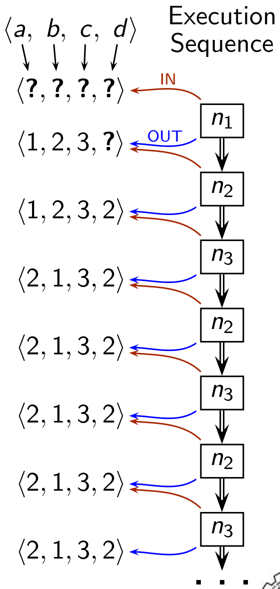
An Introduction to Constant Propagation



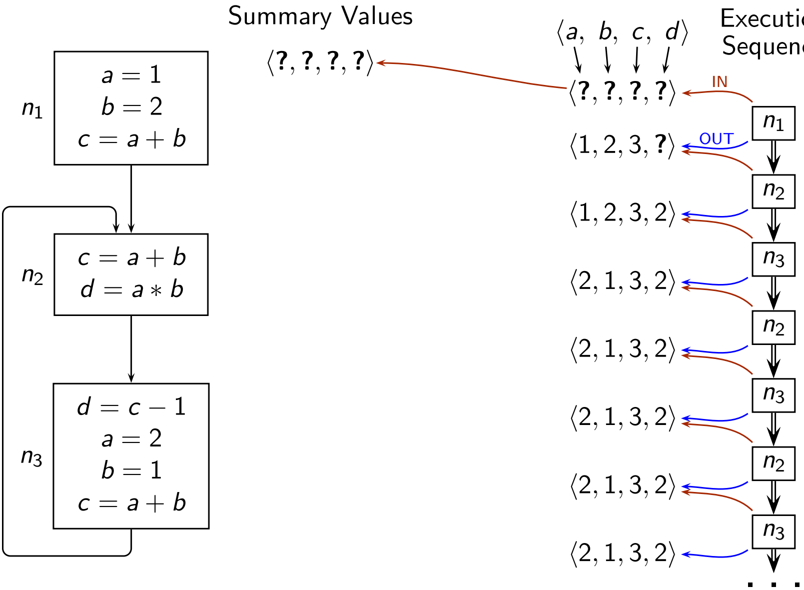


An Introduction to Constant Propagation

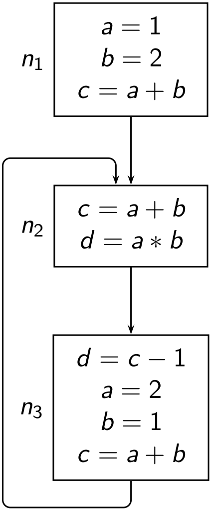




An Introduction to Constant Propagation



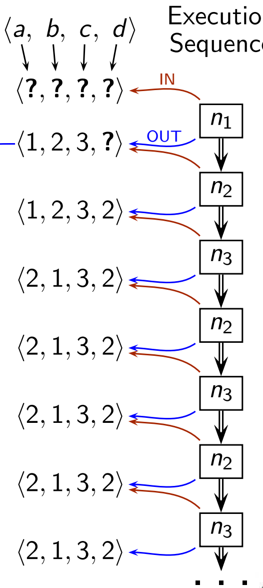
An Introduction to Constant Propagation



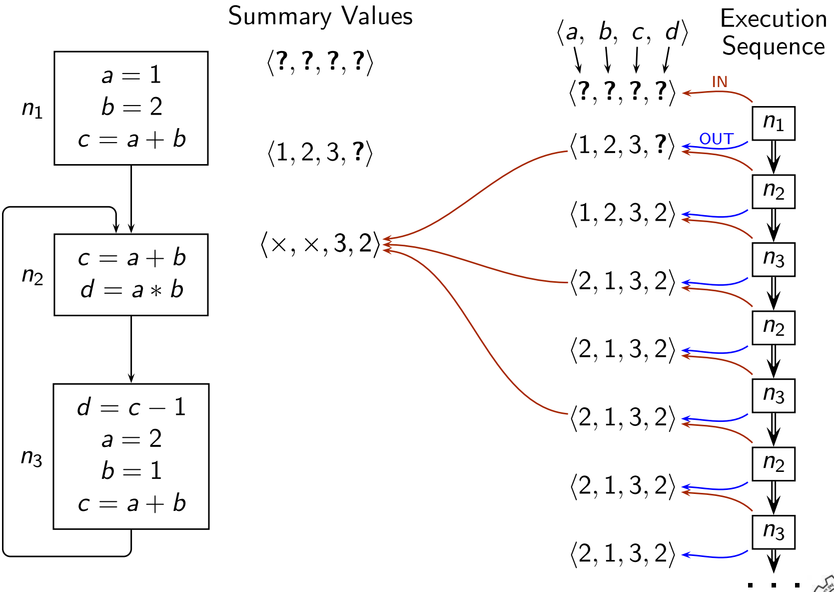
Summary Values

$\langle ?, ?, ?, ? \rangle$

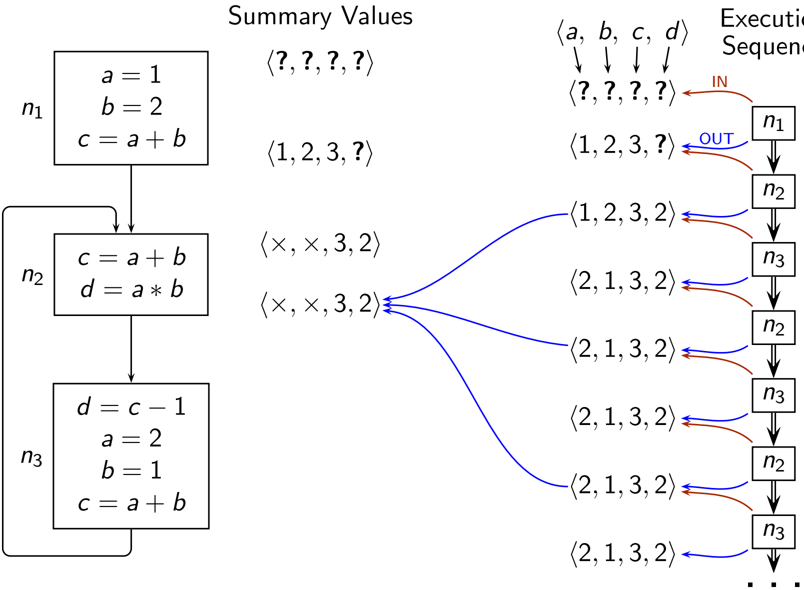
$\langle 1, 2, 3, ? \rangle$



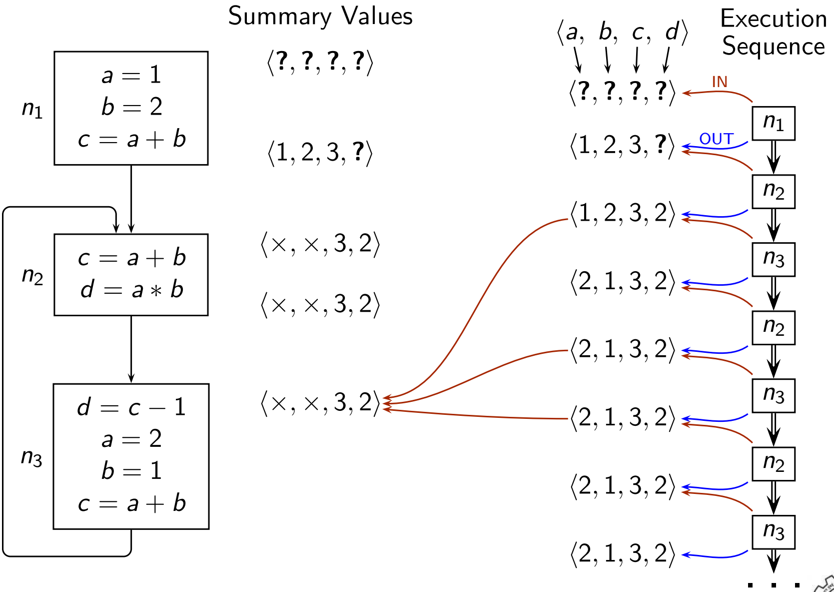
An Introduction to Constant Propagation



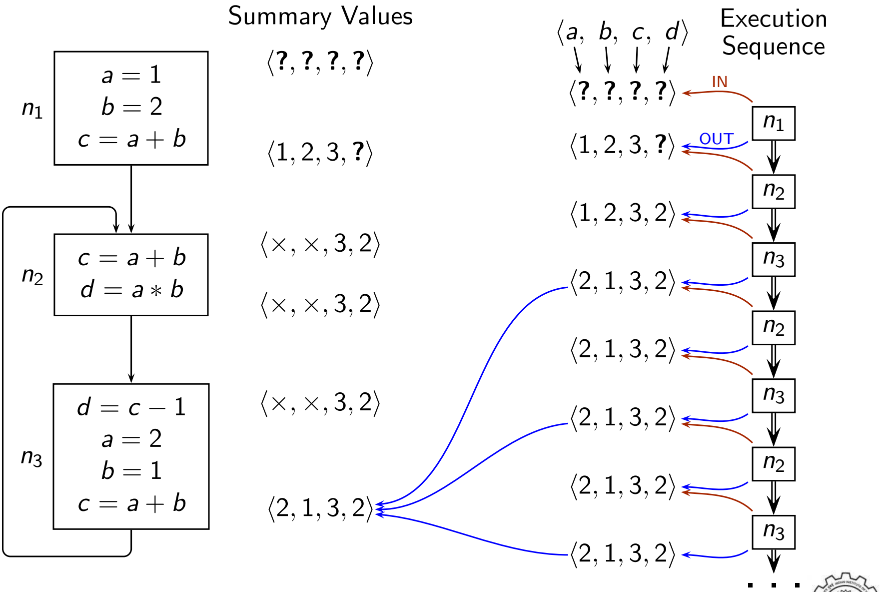
An Introduction to Constant Propagation



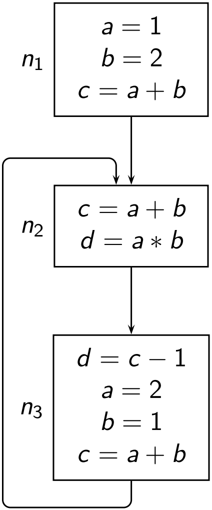
An Introduction to Constant Propagation



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

Data Flow Values for Constant Propagation

- Tuples of the form $\langle \xi_1, \xi_2, \ldots, \xi_k \rangle$ where ξ_i is the data flow value for i^{th} variable.

  Unlike bit vector frameworks, value ξ_i is not 0 or 1 (i.e. true or false). Instead, it is one of the following:
  - **?** indicating that not much is known about the constantness of variable v_i
  - $\times$ indicating that variable v_i does not have a constant value
  - An integer constant c_1 if the value of v_i is known to be c_1 at compile time

- Alternatively, sets of pairs $\langle v_i, \xi_i \rangle$ for each variable v_i.

Confluence Operation for Constant Propagation

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

$\sqcap$	$\langle a, \mathbf{?} \rangle$	$\langle a, \times \rangle$	$\langle a, c_1 \rangle$
$\langle a, \mathbf{?} \rangle$	$\langle a, \mathbf{?} \rangle$	$\langle a, \times \rangle$	$\langle a, c_1 \rangle$
$\langle a, \times \rangle$	$\langle a, \times \rangle$	$\langle a, \times \rangle$	$\langle a, \times \rangle$
$\langle a, c_2 \rangle$	$\langle a, c_2 \rangle$	$\langle a, \times \rangle$	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?

Flow Functions for Constant Propagation

- Flow function for $r = a_1 * a_2$

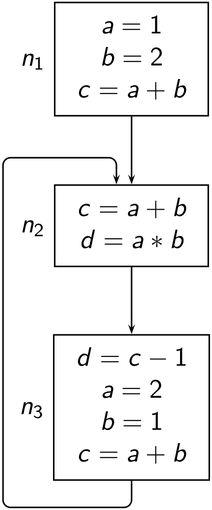
mult	$\langle a_1, \mathbf{?} \rangle$	$\langle a_1, \times \rangle$	$\langle a_1, c_1 \rangle$
$\langle a_2, \mathbf{?} \rangle$	$\langle r, \mathbf{?} \rangle$	$\langle r, \times \rangle$	$\langle r, \mathbf{?} \rangle$
$\langle a_2, \times \rangle$	$\langle r, \times \rangle$	$\langle r, \times \rangle$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

- This cannot be expressed in the form

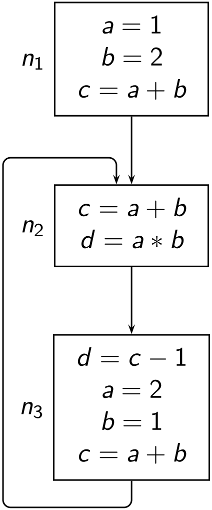
$$f_n(X) = \text{Gen}_n \cup (X - \text{Kill}_n)$$

where Gen_n and Kill_n are constant effects of block n.

Round Robin Iterative Analysis for Constant Propagation



Round Robin Iterative Analysis for Constant Propagation



Iteration #1

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

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

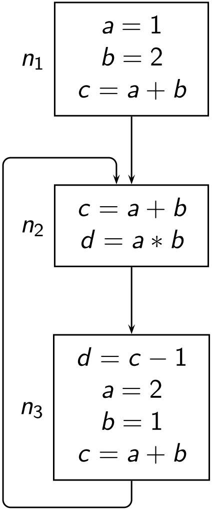
$\langle 1, 2, 3, \mathbf{?} \rangle$

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

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

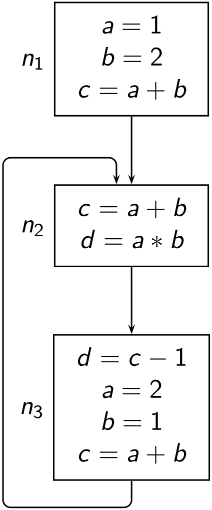
$\langle 2, 1, 3, 2 \rangle$

Round Robin Iterative Analysis for Constant Propagation



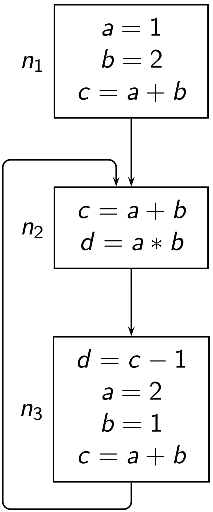
Iteration #1	Iteration #2
$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle$	$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle$
$\langle 1, 2, 3, \mathbf{?} \rangle$	$\langle 1, 2, 3, \mathbf{?} \rangle$
$\langle 1, 2, 3, \mathbf{?} \rangle$	$\langle \times, \times, 3, 2 \rangle$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

Round Robin Iterative Analysis for Constant Propagation



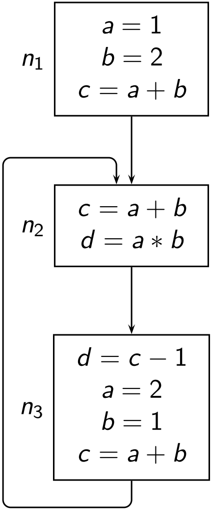
Iteration #1	Iteration #2	Iteration #3
$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle$	$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle$	$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle$
$\langle 1, 2, 3, \mathbf{?} \rangle$	$\langle 1, 2, 3, \mathbf{?}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

Round Robin Iterative Analysis for Constant Propagation



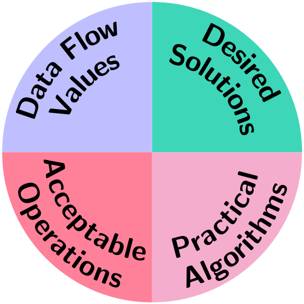
Iteration #1	Iteration #2	Iteration #3	Desired solution
$\langle ?, ?, ?, ? \rangle$	$\langle ?, ?, ?, ? \rangle$	$\langle ?, ?, ?, ? \rangle$	$\langle ?, ?, ?, ? \rangle$
$\langle 1, 2, 3, ? \rangle$	$\langle 1, 2, 3, ? \rangle$	$\langle 1, 2, 3, ? \rangle$	$\langle 1, 2, 3, ? \rangle$
$\langle 1, 2,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

Round Robin Iterative Analysis for Constant Propagation



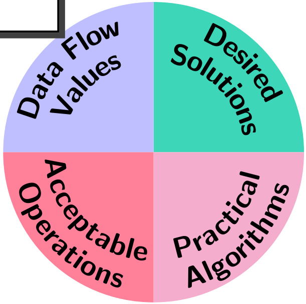
Iteration #1	Iteration #2	Iteration #3	Desired solution
$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle$	$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle$	$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle$	$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?}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

Issues in Data Flow Analysis



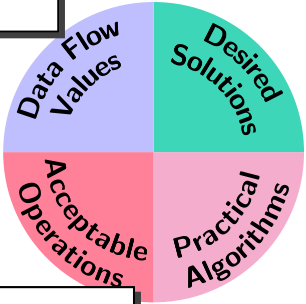
Issues in Data Flow Analysis

- Representation
- Approximation: Partial Order, Lattices



Issues in Data Flow Analysis

- Representation
- Approximation: Partial Order, Lattices

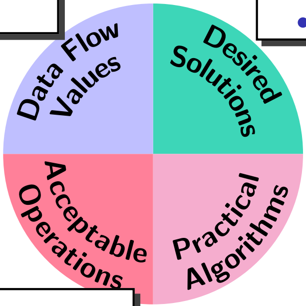


- Merge: Commutativity, Associativity, Idempotence
- Flow Functions: Monotonicity, Distributivity, Boundedness, Separability

Issues in Data Flow Analysis

- Representation
- Approximation: Partial Order, Lattices

- Existence
- Safety (soundness)
- Precision

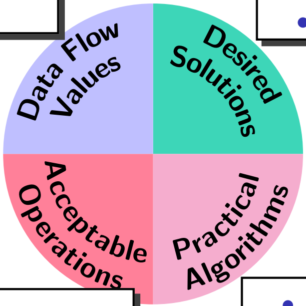


- Merge: Commutativity, Associativity, Idempotence
- Flow Functions: Monotonicity, Distributivity, Boundedness, Separability

Issues in Data Flow Analysis

- Representation
- Approximation: Partial Order, Lattices

- Existence
- Safety (soundness)
- Precision



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

- Complexity, efficiency
- Convergence
- Initialization

Part 3

Data Flow Values: An Overview

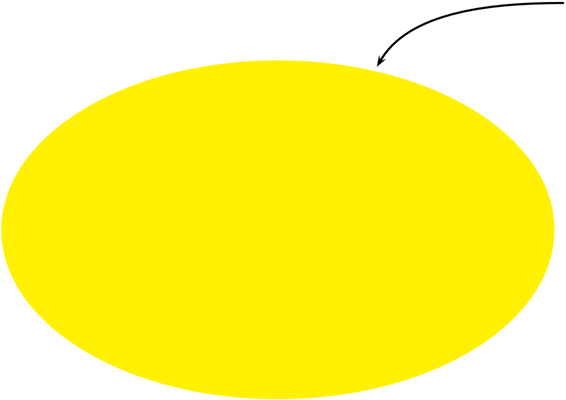
Data Flow Values: An Outline of Our Discussion

- 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
- Cartesian product of lattices
- Example of lattices

Part 4

A Digression on Lattices

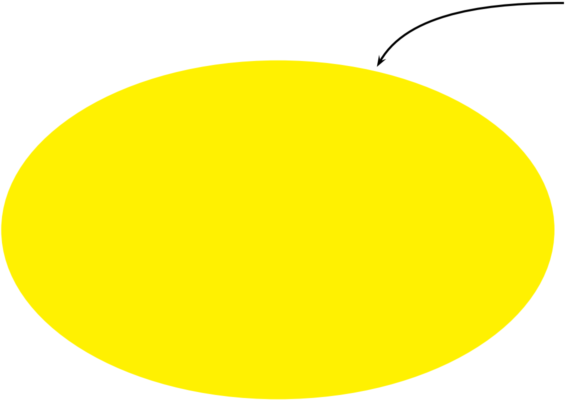
Partially Ordered Sets and Lattices



Partially ordered sets

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

Partially Ordered Sets and Lattices



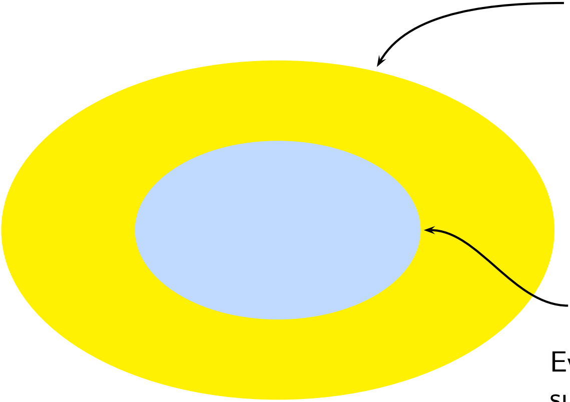
Partially ordered sets

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



Partially ordered sets

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

Lattices

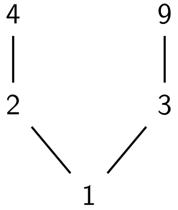
Every non-empty finite subset has a greatest lower bound (glb) and a least upper bound (lub)

Partially Ordered Sets

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

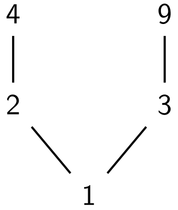
Partially Ordered Sets

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



Partially Ordered Sets

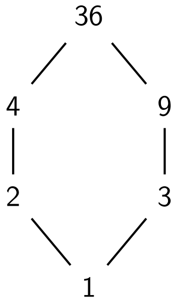
Set $\{1, 2, 3, 4, 9\}$ with $\sqsubseteq$ relation as "divides" (i.e. $a \sqsubseteq b$ iff a divides b)



Subsets $\{4, 9\}$ and $\{2, 3\}$ do not have an upper bound in the set

Lattice

Set $\{1, 2, 3, 4, 9, 36\}$ with $\sqsubseteq$ relation as "divides" (i.e. $a \sqsubseteq b$ iff a divides b)



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 $\leq$ 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 $\leq$ 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 $\leq$ 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 $\leq$ 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 $\emptyset$ (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 $\emptyset$ (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 $\emptyset$ (because there is no element in $\emptyset$).
    The greatest among these lower bounds is $\top$.
  - lub($\emptyset$) is $\bot$

Finite Lattices are Complete

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

Available Expressions Analysis

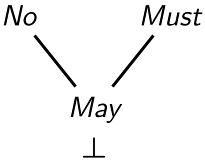
$$\begin{tikzcd}
 & (\top) & \\
 & \{e_1, e_2, e_3\} \arrow[dl] \arrow[d] \arrow[dr] & \\
\{e_1, e_2\} \arrow[d] \arrow[dr] & \{e_1, e_3\} \arrow[dl] \arrow[dr] & \{e_2, e_3\} \arrow[dl] \arrow[d] \\
\{e_1\} \arrow[dr] & \{e_2\} \arrow[d] & \{e_3\} \arrow[dl] \\
 & \emptyset & \\
 & (\bot) &
\end{tikzcd}$$

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
  However, a lub may not exist for arbitrary sets

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 $\bot$ elements.

A Bounded Lattice need not be Complete

- Let A be all finite subsets of $\mathbb{Z}$.
- The poset $(A \cup \{\mathbb{Z}\}, \subseteq)$ is a bounded lattice with $\top = \mathbb{Z}$ and $\bot = \emptyset$.
- Does the set of all sets that do not contains a given number (say 1) has an lub in $A \cup \{\mathbb{Z}\}$?

A Bounded Lattice need not be Complete

- Let A be all finite subsets of $\mathbb{Z}$.
- The poset $(A \cup \{\mathbb{Z}\}, \subseteq)$ is a bounded lattice with $\top = \mathbb{Z}$ and $\bot = \emptyset$.
- Does the set of all sets that do not contains a given number (say 1) has an lub in $A \cup \{\mathbb{Z}\}$?
- The union of all finite sets that do not contain 1 is an infinite set that does not contain 1.
  This set is not contained in $A \cup \{\mathbb{Z}\}$.

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

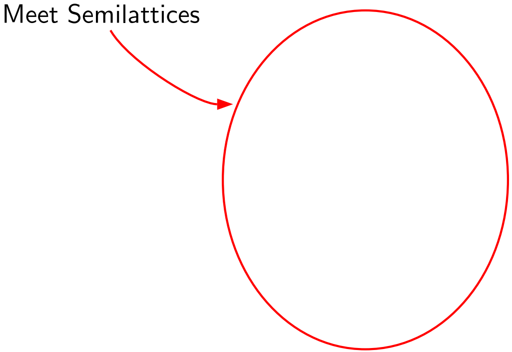
- If L satisfies acc and dcc, then
  - L has finite height, and
  - L is complete.

- A complete lattice need not have finite height (i.e. strict chains may not be finite).

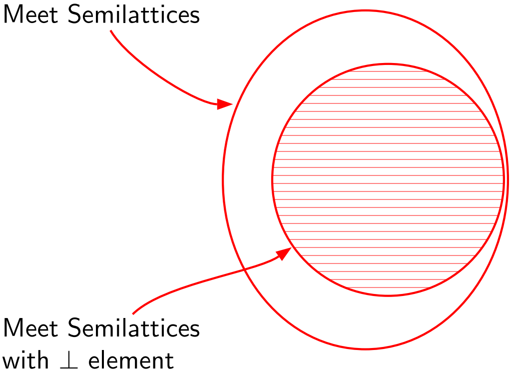
  Example:

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

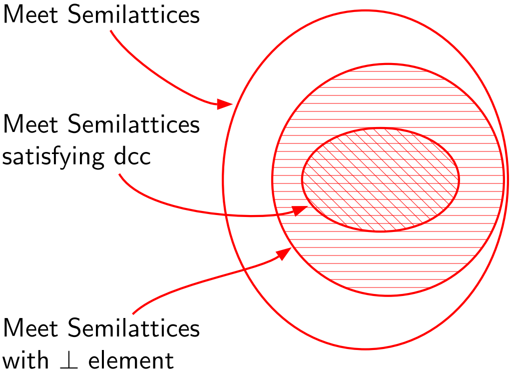
Variants of Lattices



Variants of Lattices

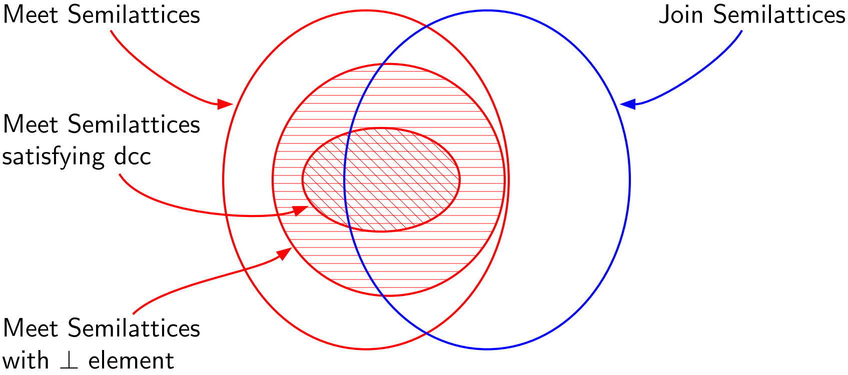


Variants of Lattices



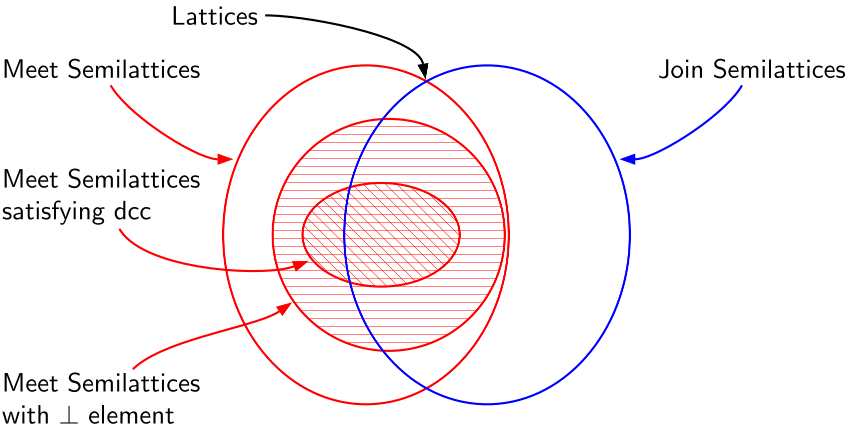
- dcc: descending chain condition

Variants of Lattices



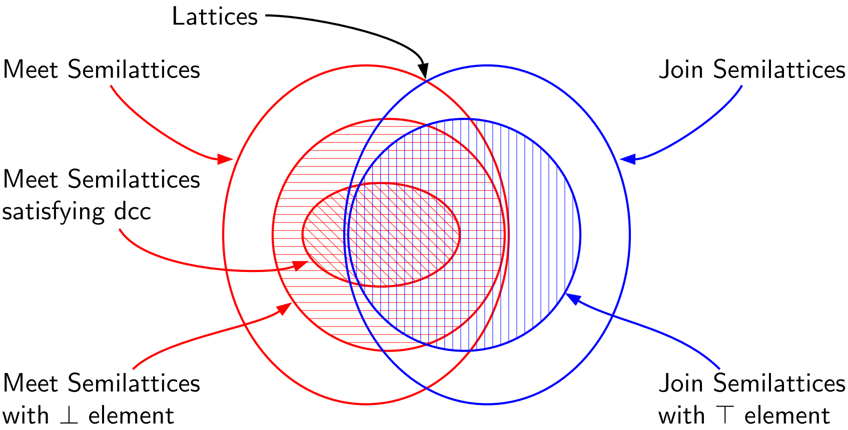
- dcc: descending chain condition

Variants of Lattices



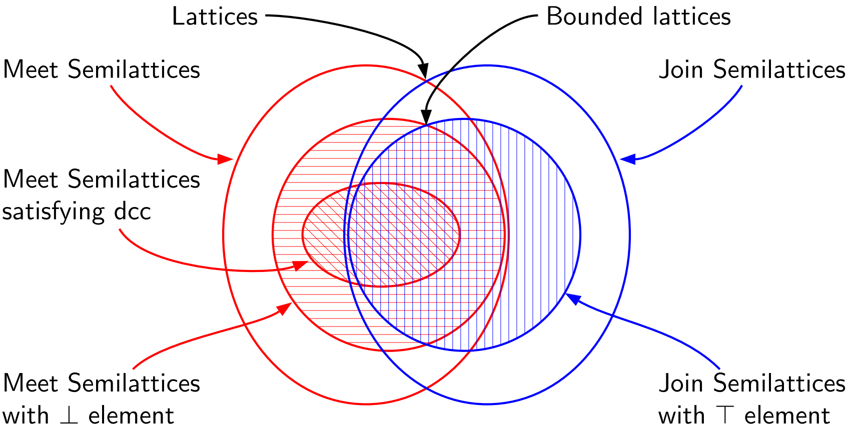
- dcc: descending chain condition

Variants of Lattices



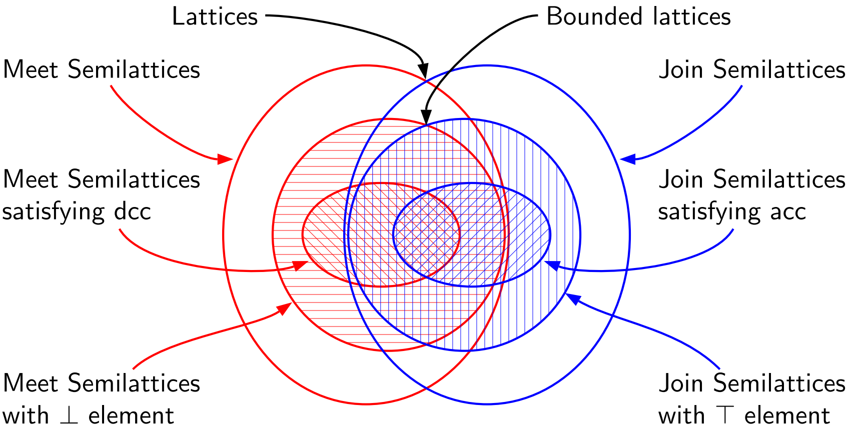
- dcc: descending chain condition

Variants of Lattices



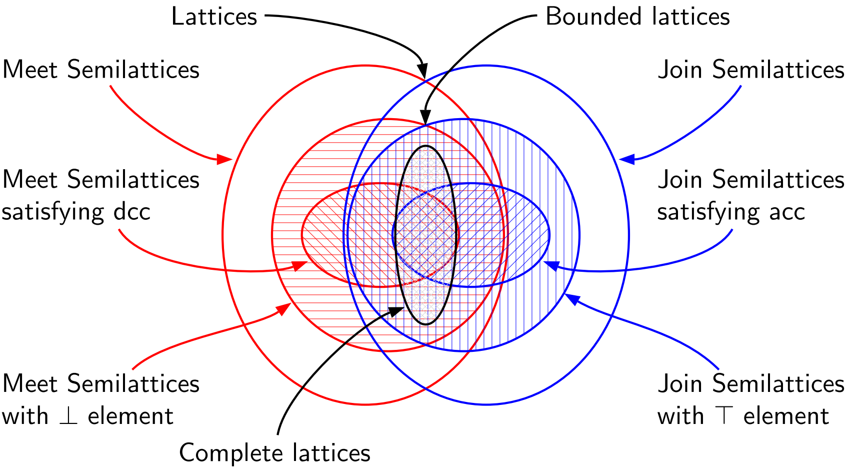
- dcc: descending chain condition

Variants of Lattices



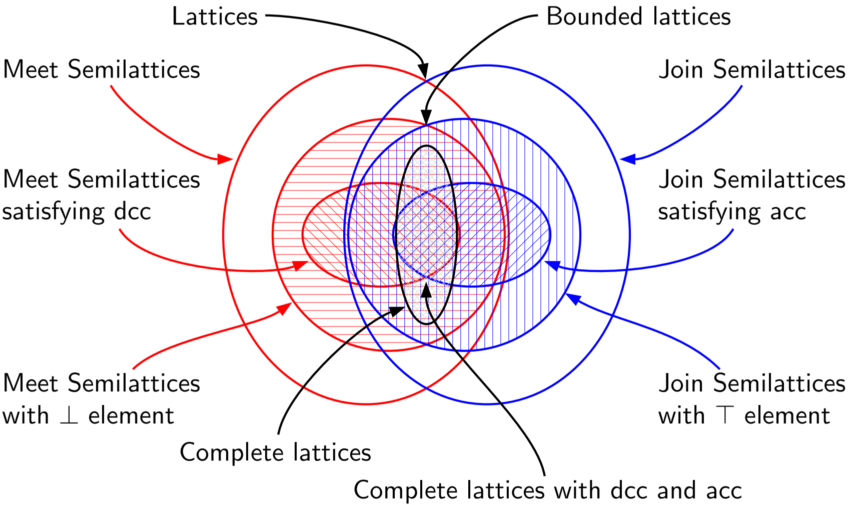
- dcc: descending chain condition
- acc: ascending chain condition

Variants of Lattices



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

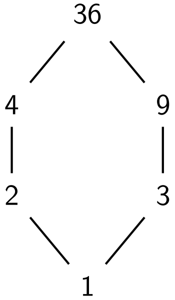
Variants of Lattices



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

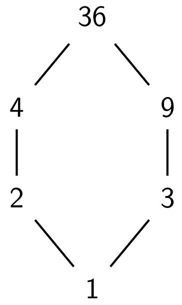
Operations on Lattices

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



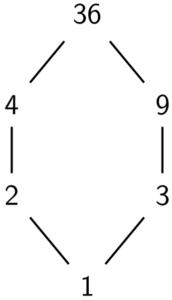
Operations on Lattices

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



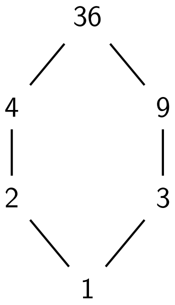
Operations on Lattices

- Meet ($\sqcap$) and Join ($\sqcap$)
  - $x \sqcap y$ 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

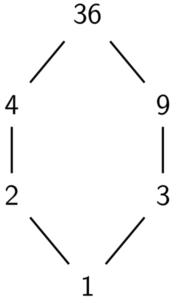
- Meet ($\sqcap$) and Join ($\sqcap$)
  - $x \sqcap y$ computes the glb of x and y.
    $z = x \sqcap y \Rightarrow z \sqsubseteq x \wedge z \sqsubseteq y$
  - $x \sqcup y$ computes the lub of x and y.
    $z = x \sqcup y \Rightarrow z \sqsupseteq x \wedge z \sqsupseteq y$
  - $\sqcap$ and $\sqcup$ are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



Operations on Lattices

- Meet ($\sqcap$) and Join ($\sqcap$)
  - $x \sqcap y$ 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

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

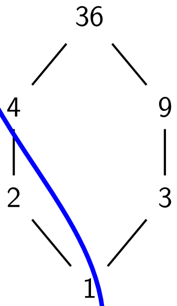


Operations on Lattices

Greatest common divisor (or highest common factor) **in the lattice**

- Meet ($\sqcap$) and Join ($\sqcap$)
  - $x \sqcap y$ 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

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



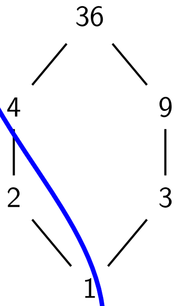
$x \sqcap y = gcd'(x, y)$

Operations on Lattices

Greatest common divisor (or highest common factor) **in the lattice**

- Meet ($\sqcap$) and Join ($\sqcap$)
  - $x \sqcap y$ 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

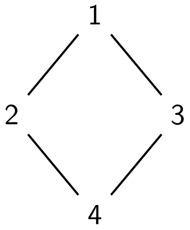
$$
\begin{aligned}
\forall x \in L,\ x \sqcap \top &= x \\
\forall x \in L,\ x \sqcup \top &= \top \\
\forall x \in L,\ x \sqcap \perp &= \perp \\
\forall x \in L,\ x \sqcup \perp &= x
\end{aligned}
$$



$$
\begin{aligned}
x \sqcap y &= gcd'(x, y) \\
x \sqcup y &= lcm'(x, y)
\end{aligned}
$$

Lowest common multiple **in the lattice**

Cartesian Product of Lattice



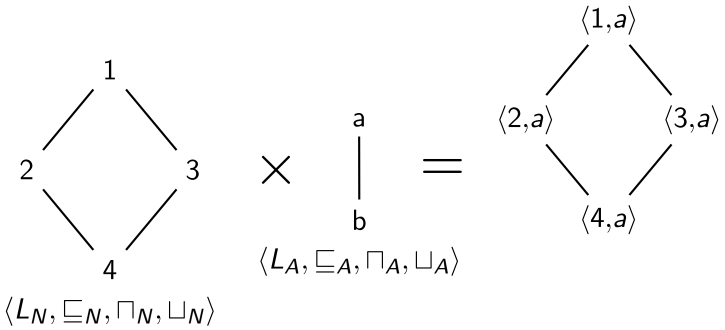
$\langle L_N, \sqsubseteq_N, \sqcap_N, \sqcup_N \rangle$

$\times$

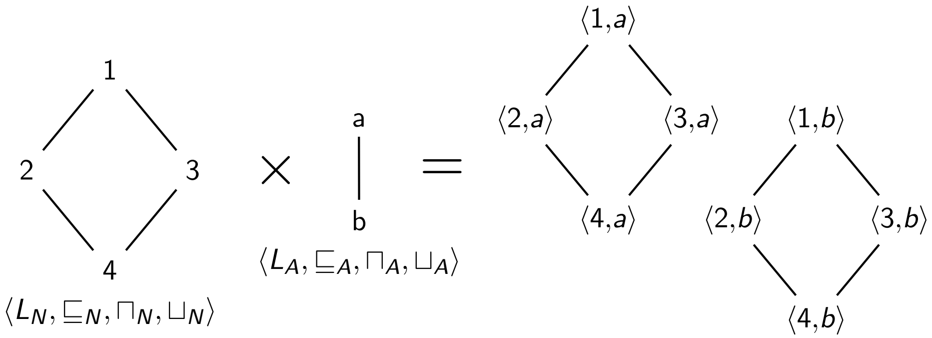
$\langle L_A, \sqsubseteq_A, \sqcap_A, \sqcup_A \rangle$

$=$

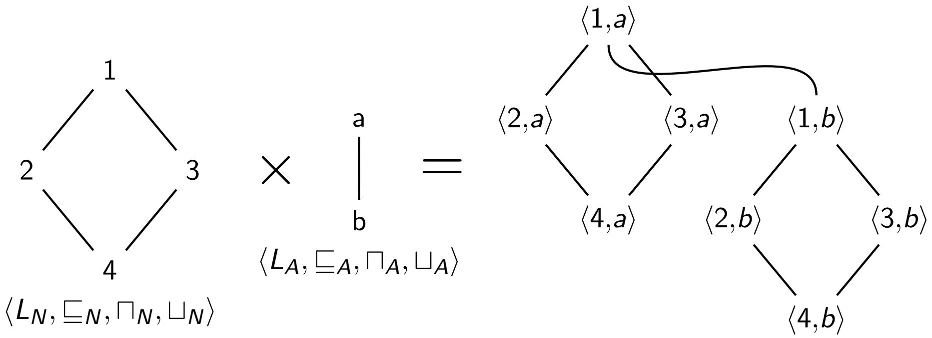
Cartesian Product of Lattice



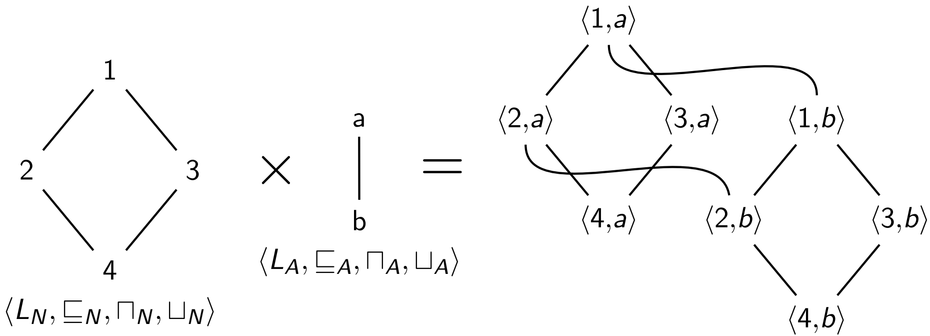
Cartesian Product of Lattice



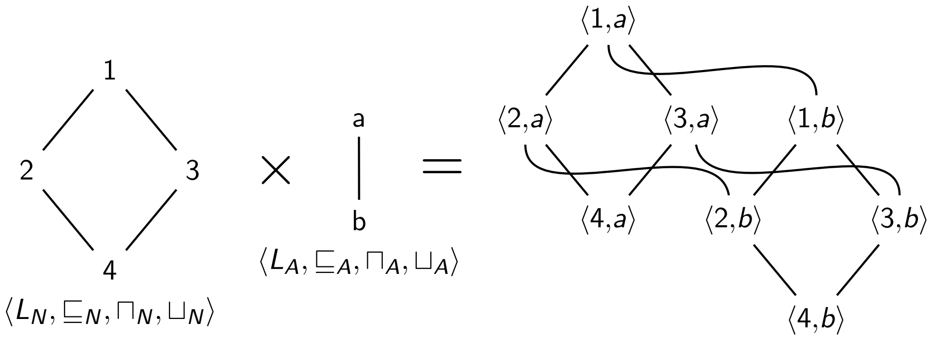
Cartesian Product of Lattice



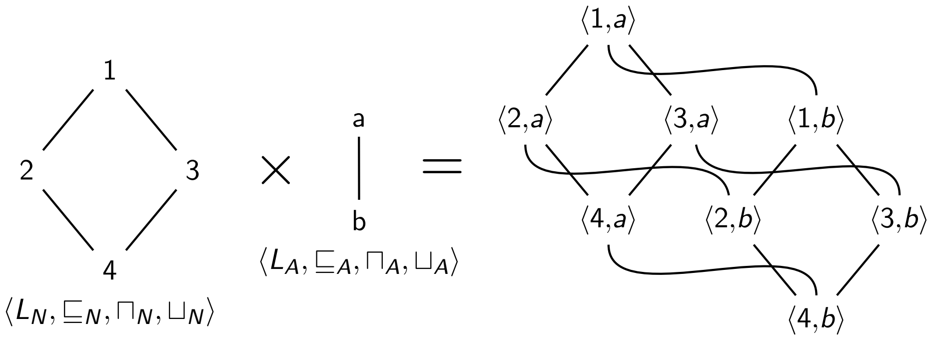
Cartesian Product of Lattice



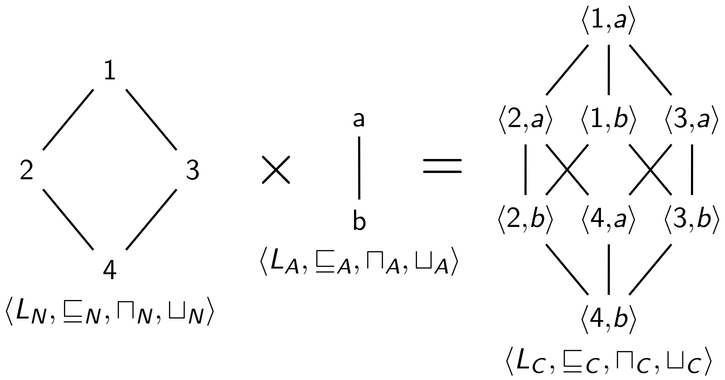
Cartesian Product of Lattice



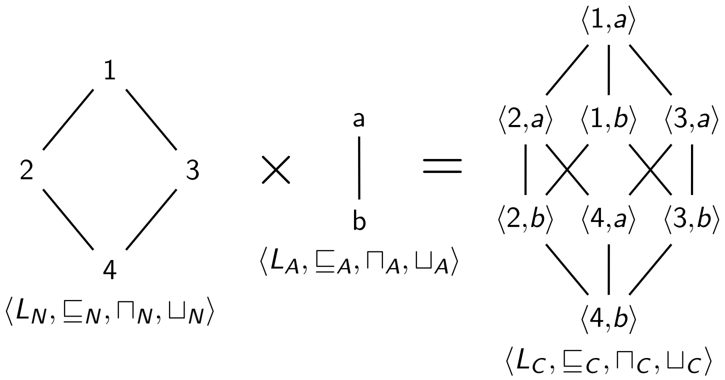
Cartesian Product of Lattice



Cartesian Product of Lattice



Cartesian Product of Lattice



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

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

- glb must exist for all non-empty finite subsets

The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- glb must exist for all non-empty finite subsets
- $\perp$ must exist
  What guarantees the presence of $\perp$?

The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

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

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

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

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

- glb must exist for all non-empty finite subsets
- $\perp$ 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.

The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- glb must exist for all non-empty finite subsets
- $\bot$ 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.
  - 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

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

Set View of the Lattice

$$
\begin{tikzcd}
Y \arrow[d, "\sqsubseteq"] \\
X
\end{tikzcd}
$$

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

$$\begin{tikzcd}
Y \arrow[d, "\sqsubseteq"] \\
X
\end{tikzcd}$$

$$\begin{tikzcd}
 & 111 \arrow[dl] \arrow[d] \arrow[dr] & \\
110 \arrow[d] \arrow[dr] & 101 \arrow[dl] \arrow[dr] & 011 \arrow[dl] \arrow[d] \\
100 \arrow[dr] & 010 \arrow[d] & 001 \arrow[dl] \\
 & 000 &
\end{tikzcd}$$

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

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

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

May prohibit correct optimization

May enable wrong optimization

Analysis	Application	Safe Approximation	Exhaustive Approximation

Context Determines the Validity of Approximations

May prohibit correct optimization

May enable wrong 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

May prohibit correct optimization

May enable wrong 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

May prohibit correct optimization

May enable wrong 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

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
  - 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
  - It may be unsafe, though.

We want most exhaustive information which is also safe.

Most Approximate Values in a Complete Lattice

- *Top*. $\forall x \in L,\ x \sqsubseteq \top$. The most exhaustive value.

- *Bottom*. $\forall x \in L,\ \bot \sqsubseteq x$. The safest value.

Most Approximate Values in a Complete Lattice

- *Top*. $\forall x \in L,\ x \sqsubseteq \top$. The most exhaustive value.
  - Using $\top$ in place of any data flow value will never miss out (or rule out) any possible value.
- *Bottom*. $\forall x \in L,\ \bot \sqsubseteq x$. The safest value.

Most Approximate Values in a Complete Lattice

- *Top*. $\forall x \in L,\ x \sqsubseteq \top$. The most exhaustive value.
  - Using $\top$ in place of any data flow value will never miss out (or rule out) any possible value.
  - The consequences may be sematically *unsafe*, or *incorrect*.
- *Bottom*. $\forall x \in L,\ \bot \sqsubseteq x$. The safest value.

Most Approximate Values in a Complete Lattice

- *Top*. $\forall x \in L,\ x \sqsubseteq \top$. The most exhaustive value.
  - Using $\top$ in place of any data flow value will never miss out (or rule out) any possible value.
  - The consequences may be sematically *unsafe*, or *incorrect*.
- *Bottom*. $\forall x \in L,\ \bot \sqsubseteq x$. The safest value.
  - Using $\bot$ in place of any data flow value will never be *unsafe*, or *incorrect*.

Most Approximate Values in a Complete Lattice

- *Top*. $\forall x \in L,\ x \sqsubseteq \top$. The most exhaustive value.
  - Using $\top$ in place of any data flow value will never miss out (or rule out) any possible value.
  - The consequences may be sematically *unsafe*, or *incorrect*.
- *Bottom*. $\forall x \in L,\ \bot \sqsubseteq x$. The safest value.
  - 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*. $\forall x \in L,\ x \sqsubseteq \top$. The most exhaustive value.
  - Using $\top$ in place of any data flow value will never miss out (or rule out) any possible value.
  - The consequences may be sematically *unsafe*, or *incorrect*.
- *Bottom*. $\forall x \in L,\ \bot \sqsubseteq x$. The safest value.
  - 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

$$\begin{tikzcd}
 & 111 \arrow[dl] \arrow[d] \arrow[dr] & \\
110 \arrow[d] \arrow[dr] & 101 \arrow[dl] \arrow[dr] & 011 \arrow[dl] \arrow[d] \\
100 \arrow[dr] & 010 \arrow[d] & 001 \arrow[dl] \\
 & 000 &
\end{tikzcd}$$

$\widehat{L}$ = Lattice for a single expression

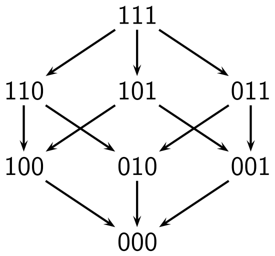
(Expression e is available)

$$\begin{tikzcd}
1 \text{ or } \{e\} \arrow[d] \\
0 \text{ or } \emptyset
\end{tikzcd}$$

(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)

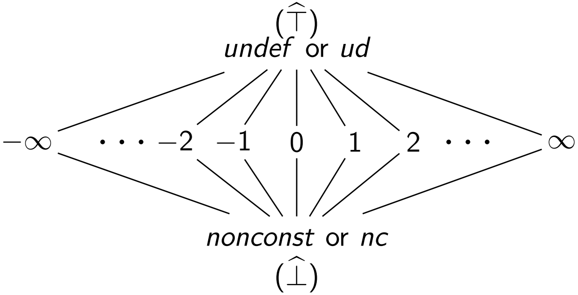
Cartesian products if sets are used, vectors (or tuples) if bit are used.

- $L = \widehat{L} \times \widehat{L} \times \widehat{L}$ 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

$(\widehat{\top})$	$(\widehat{\top})$
1	0
0	1
$(\widehat{\perp})$	$(\widehat{\perp})$
$\sqcap$ is $\cap$ or Boolean AND	$\sqcap$ is $\cup$ or Boolean OR

Component Lattice for Integer Constant Propagation



- Overall lattice L is the product of $\widehat{L}$ for all variables.
- $\sqcap$ 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.

$$(\widehat{\top})$$
$$\emptyset$$

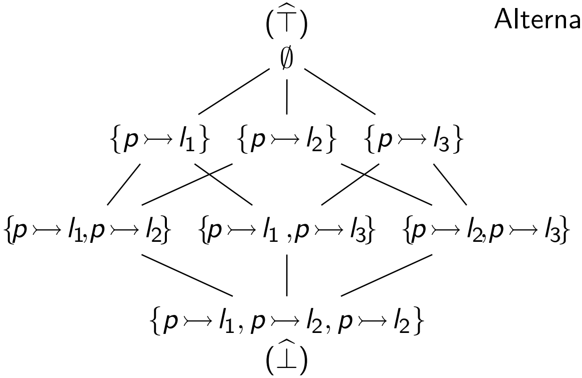
$$\{p \rightarrowtail l_1\} \quad \{p \rightarrowtail l_2\} \quad \{p \rightarrowtail l_3\}$$

$$\{p \rightarrowtail l_1, p \rightarrowtail l_2\} \quad \{p \rightarrowtail l_1, p \rightarrowtail l_3\} \quad \{p \rightarrowtail l_2, p \rightarrowtail l_3\}$$

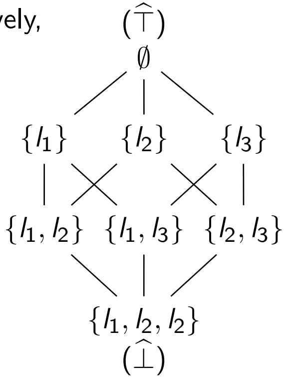
$$\{p \rightarrowtail l_1, p \rightarrowtail l_2, p \rightarrowtail l_2\}$$
$$(\widehat{\bot})$$

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.

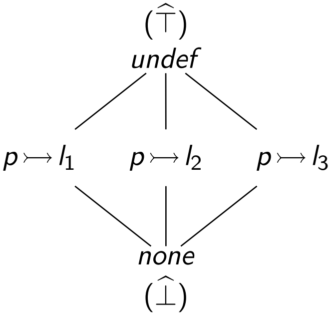


Alternatively,

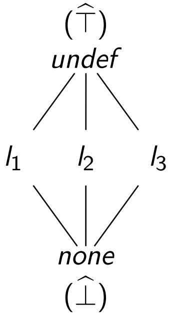


Component Lattice for Must Points-To Analysis

- 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)

unknown
(Bits 11)

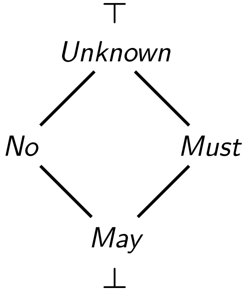
must-be-available (Bits 10)   *is-not-available* (Bits 01)

may-be-available
(Bits 00)

  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 safer than definite information.

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. $(p \rightarrowtail l, \mathit{May})$, $(p \rightarrowtail l, \mathit{Must})$, $(p \rightarrowtail l, \mathit{No})$, or $(p \rightarrowtail l, \mathit{Unknown})$.
- 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 $f : L \mapsto L$ 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

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

- 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

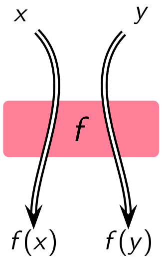
$$f(x) = \text{Gen} \cup (x - \text{Kill})$$

  - Lattices are powersets with partial orders as $\subseteq$ or $\supseteq$ relations
  - Information is merged using $\cap$ or $\cup$
- Flow functions in Faint Variables 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.

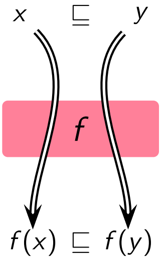
Monotonicity of Flow Functions

- 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

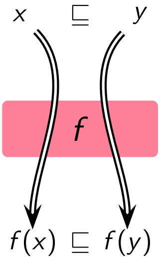
- Partial order is preserved: If x can be safely used in place of y then $f(x)$ can be safely used in place of $f(y)$



Monotonicity of Flow Functions

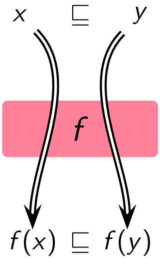
- Partial order is preserved: If x can be safely used in place of y then $f(x)$ can be safely used in place of $f(y)$

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



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 $f(y)$



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

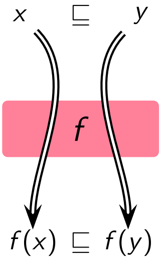
- Alternative definition

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

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 $f(y)$

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



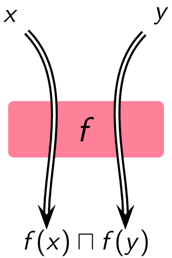
- Alternative definition

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

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

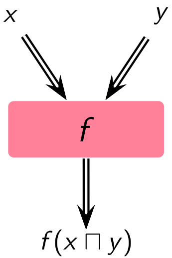
Distributivity of Flow Functions

- Merging distributes over function application



Distributivity of Flow Functions

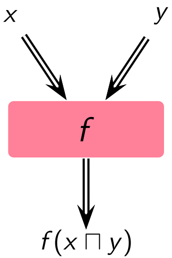
- Merging distributes over function application



Distributivity of Flow Functions

- Merging distributes over function application

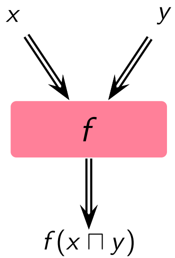
$$\forall x, y \in L, x \sqsubseteq y \Rightarrow f(x \sqcap y) = f(x) \sqcap f(y)$$



Distributivity of Flow Functions

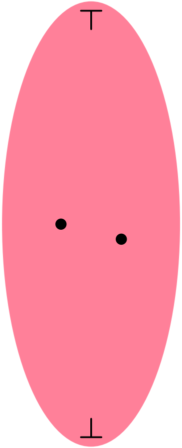
- Merging distributes over function application

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



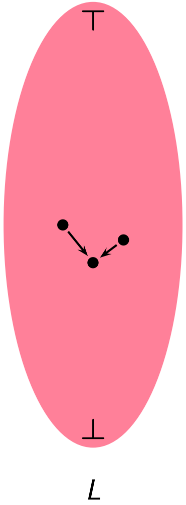
- Merging at intermediate points in shared segments of paths does not lead to imprecision.

Monotonicity and Distributivity

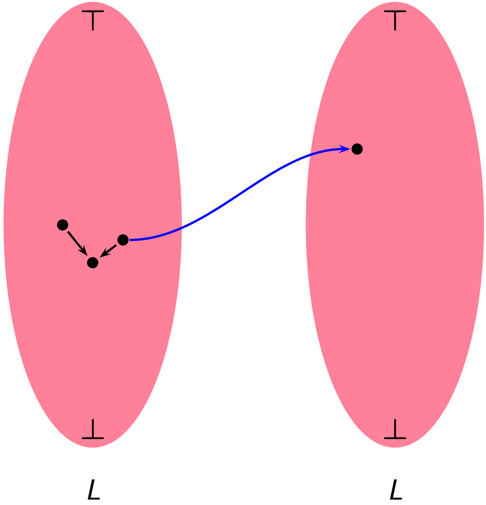


L

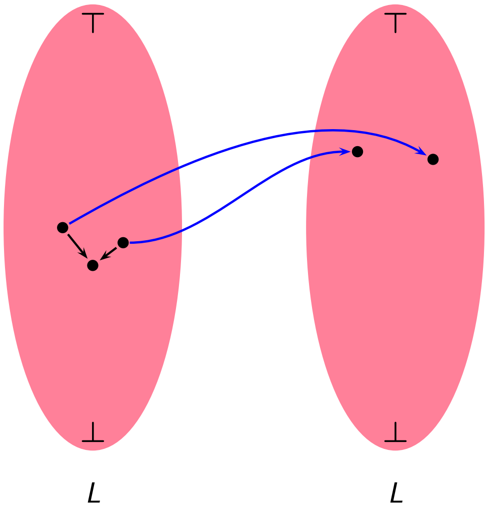
Monotonicity and Distributivity



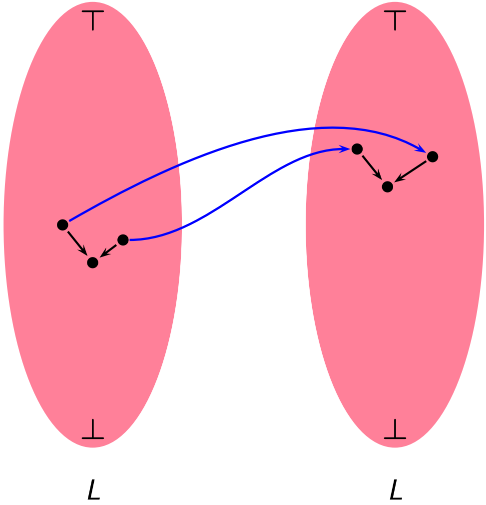
Monotonicity and Distributivity



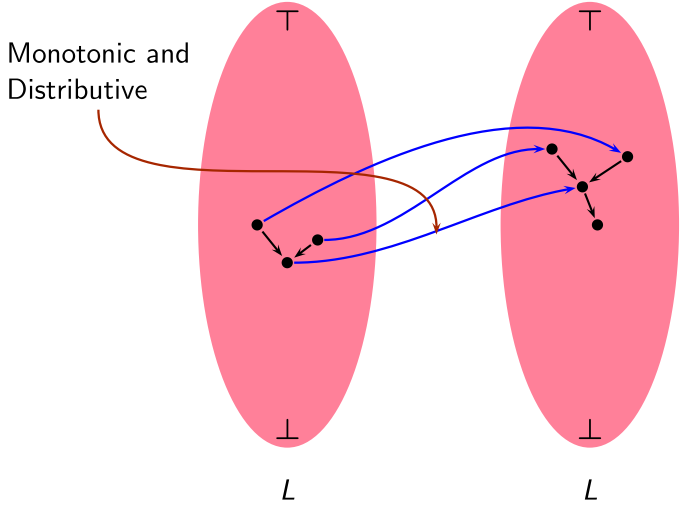
Monotonicity and Distributivity



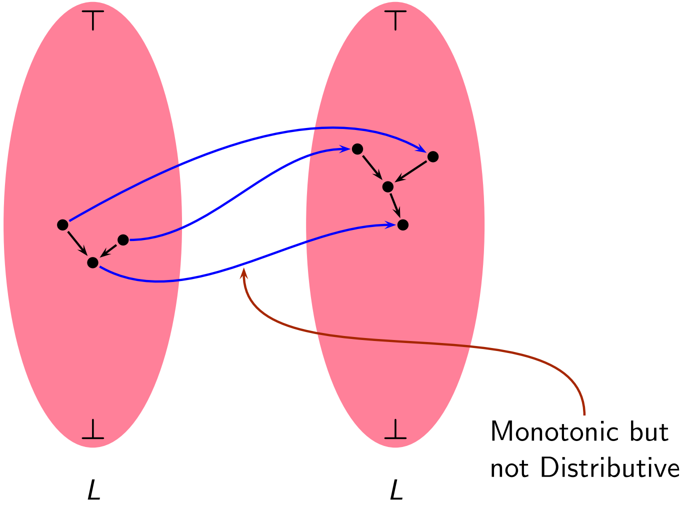
Monotonicity and Distributivity



Monotonicity and Distributivity



Monotonicity and Distributivity



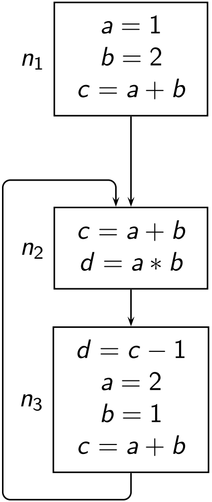
Distributivity of Bit Vector Frameworks

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

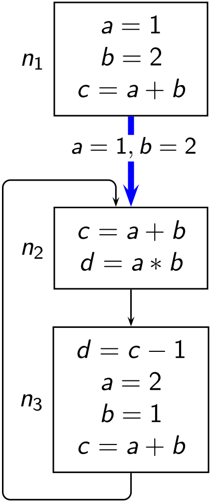
$$
\begin{aligned}
f(x \cup y) &= \text{Gen} \cup ((x \cup y) - \text{Kill}) \\
&= \text{Gen} \cup ((x - \text{Kill}) \cup (y - \text{Kill})) \\
&= (\text{Gen} \cup (x - \text{Kill}) \cup \text{Gen} \cup (y - \text{Kill})) \\
&= f(x) \cup f(y)
\end{aligned}
$$

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

Non-Distributivity of Constant Propagation

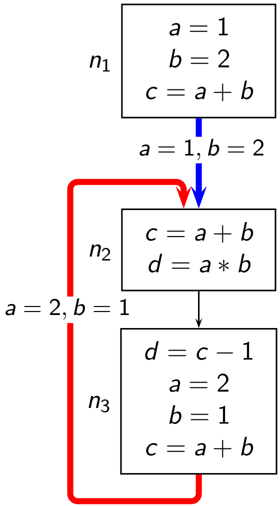


Non-Distributivity of Constant Propagation



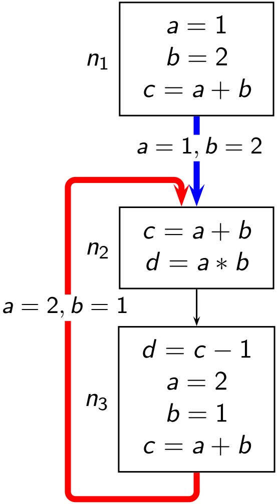
- $x = \langle 1, 2, 3, ud \rangle$ (Along $Out_{n_1} \rightarrow In_{n_2}$)

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 $Out_{n_3} \rightarrow In_{n_2}$)

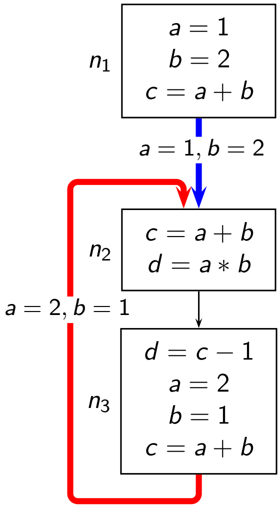
Non-Distributivity of Constant Propagation



- $x = \langle 1, 2, 3, ud \rangle$ (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 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



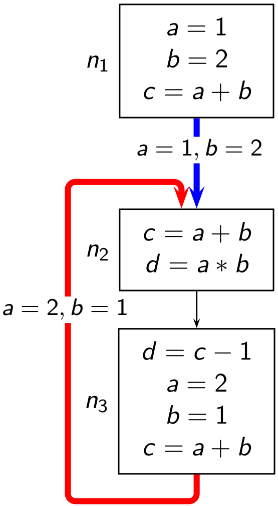
- $x = \langle 1, 2, 3, ud \rangle$ (Along $Out_{n_1} \rightarrow In_{n_2}$)
- $y = \langle 2, 1, 3, 2 \rangle$ (Along $Out_{n_3} \rightarrow In_{n_2}$)
- Function application before merging

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

- Function application after merging

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

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 before merging

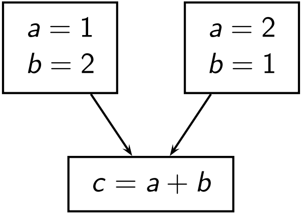
$$\begin{aligned} f(x) \sqcap f(y) &= f(\langle 1, 2, 3, ud \rangle) \sqcap f(\langle 2, 1, 3, 2 \rangle) \\ &= \langle 1, 2, 3, 2 \rangle \sqcap \langle 2, 1, 3, 2 \rangle \\ &= \langle \widehat{\bot}, \widehat{\bot}, 3, 2 \rangle \end{aligned}$$

- Function application after merging

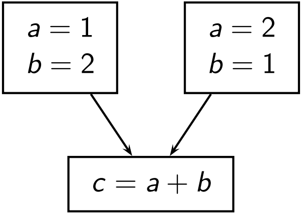
$$\begin{aligned} f(x \sqcap y) &= f(\langle 1, 2, 3, ud \rangle \sqcap \langle 2, 1, 3, 2 \rangle) \\ &= f(\langle \widehat{\bot}, \widehat{\bot}, 3, 2 \rangle) \\ &= \langle \widehat{\bot}, \widehat{\bot}, \widehat{\bot}, \widehat{\bot} \rangle \end{aligned}$$

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

Why is Constant Propagation Non-Distributive?



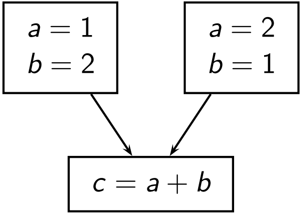
Why is Constant Propagation Non-Distribitive?



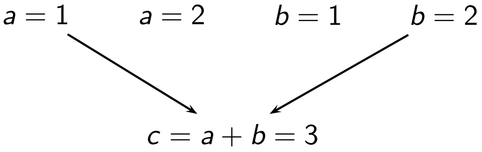
Possible combinations due to merging

$a = 1$ $a = 2$ $b = 1$ $b = 2$

Why is Constant Propagation Non-Distribitive?

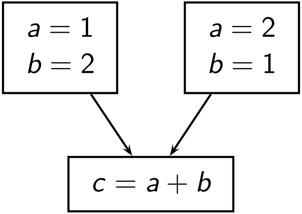


Possible combinations due to merging



- Correct combination.

Why is Constant Propagation Non-Distribitive?



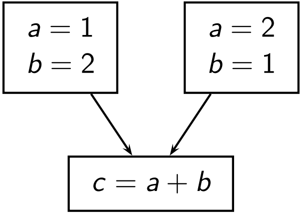
Possible combinations due to merging

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

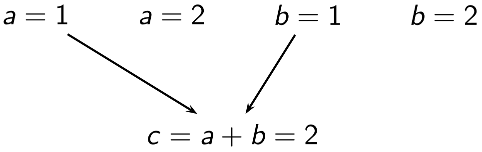
$c = a + b = 3$

- Correct combination.

Why is Constant Propagation Non-Distribitive?

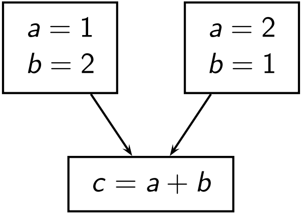


Possible combinations due to merging

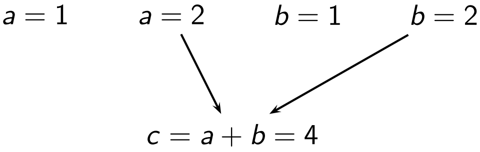


- Wrong combination.
- Mutually exclusive information.
- No execution path along which this information holds.

Why is Constant Propagation Non-Distribitive?



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
- Safety 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

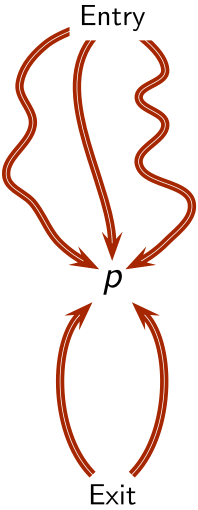
Given

- 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

Meet Over Paths (MoP) Assignment

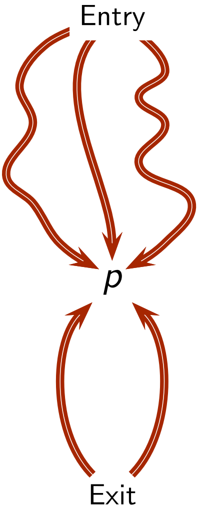


- The largest safe approximation of the information reaching a program point along all information flow paths.

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

  - 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.

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.

- 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

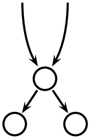
    If all paths need to be traversed $\Rightarrow$ Undecidability

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
  - 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

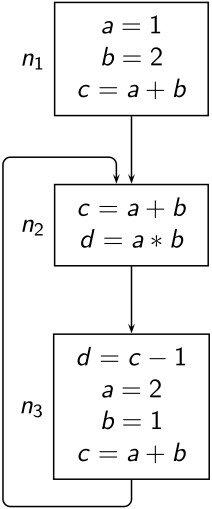
    If all paths need to be traversed ⇒ Intractability

Path based specification

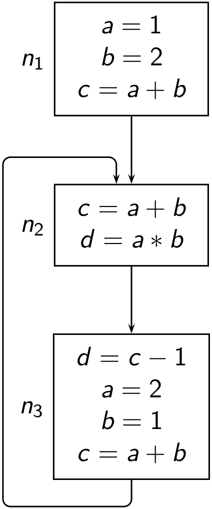
- Why not merge information at intermediate points?
  - Merging is safe but may lead to imprecision.
  - Computes fixed point solutions of data flow equations.

Edge based specifications

Assignments for Constant Propagation Example



Assignments for Constant Propagation Example



MoP

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

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

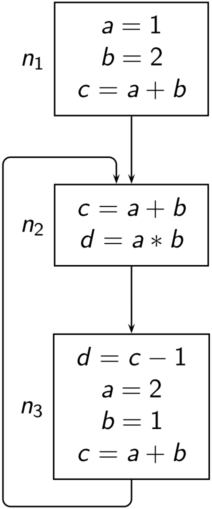
$\langle \widehat{\bot}, \widehat{\bot}, 3, 2 \rangle$

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

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

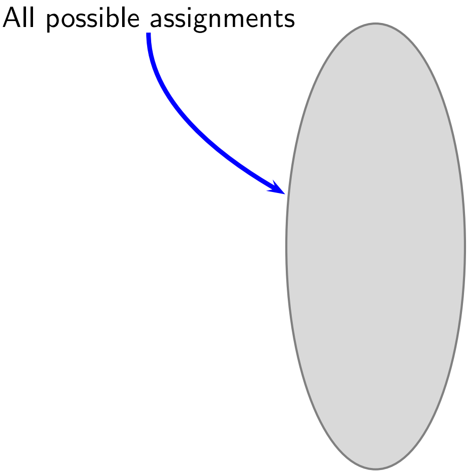
$\langle 2, 1, 3, 2 \rangle$

Assignments for Constant Propagation Example

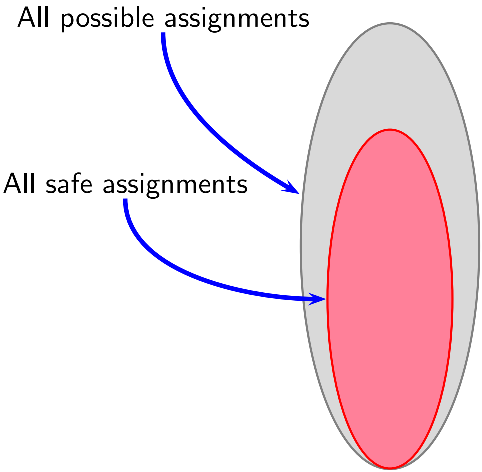


MoP	MFP
$\langle \widehat{\top}, \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle$	$\langle \widehat{\top}, \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle$
$\langle 1, 2, 3, \widehat{\top} \rangle$	$\langle 1, 2, 3, \widehat{\top} \rangle$
$\langle \widehat{\bot}, \widehat{\bot}, 3, 2 \rangle$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

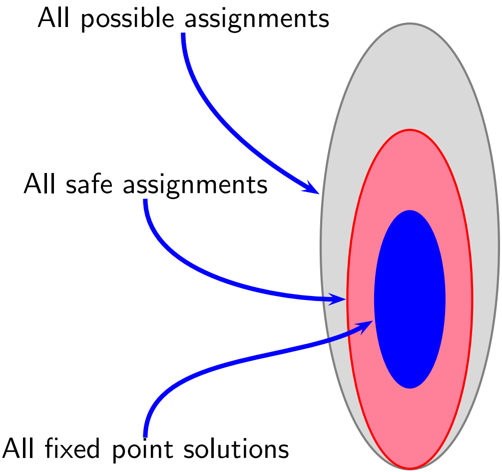
Possible Assignments as Solutions of Data Flow Analyses



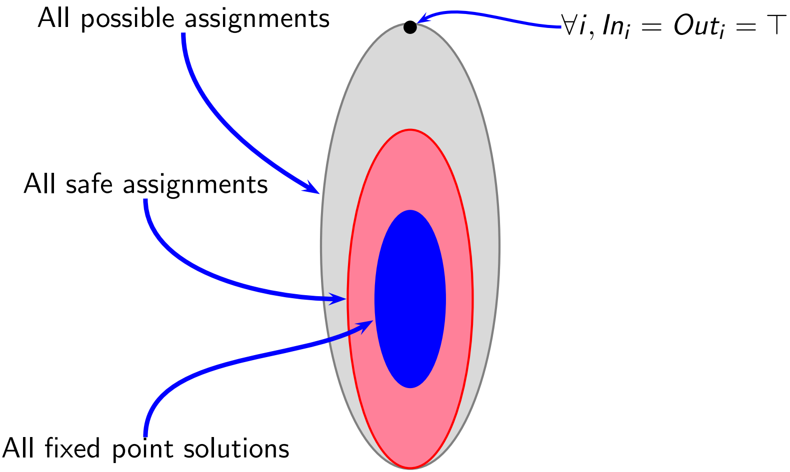
Possible Assignments as Solutions of Data Flow Analyses



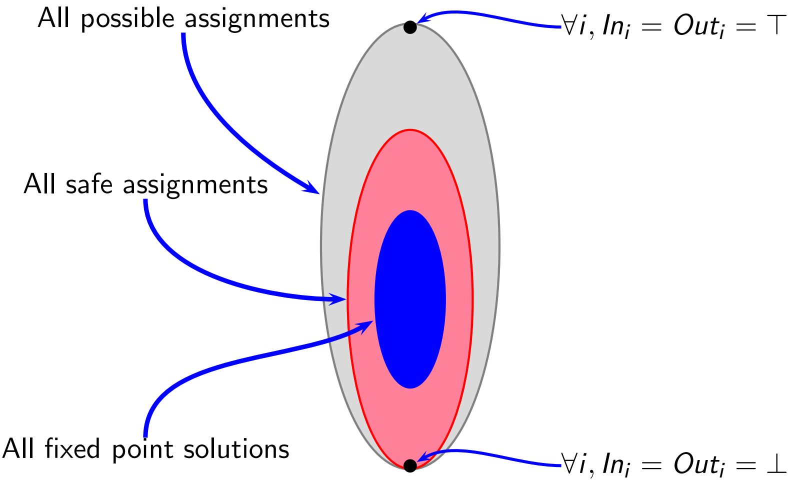
Possible Assignments as Solutions of Data Flow Analyses



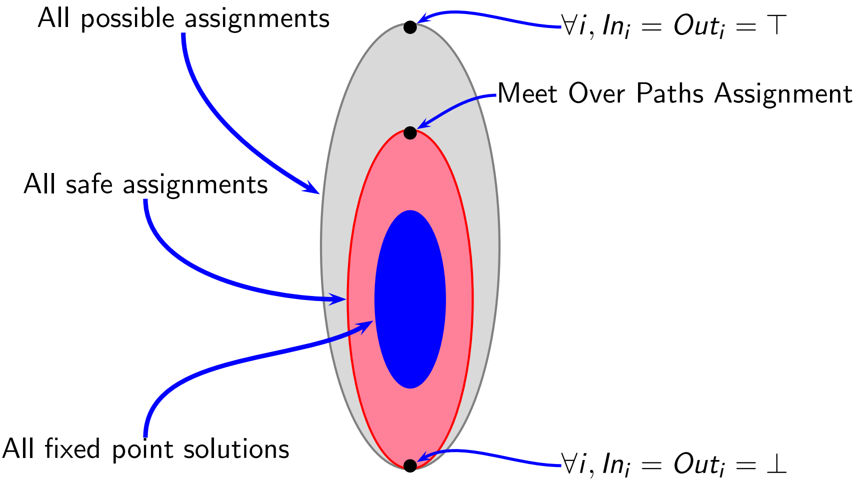
Possible Assignments as Solutions of Data Flow Analyses



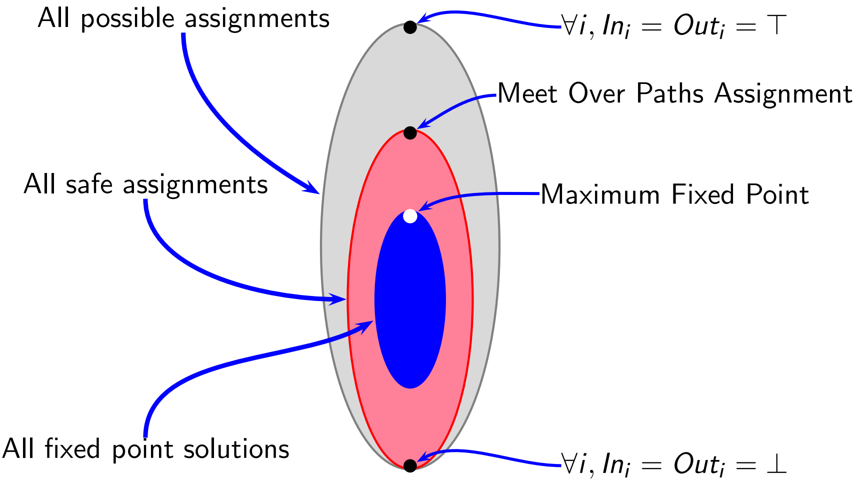
Possible Assignments as Solutions of Data Flow Analyses



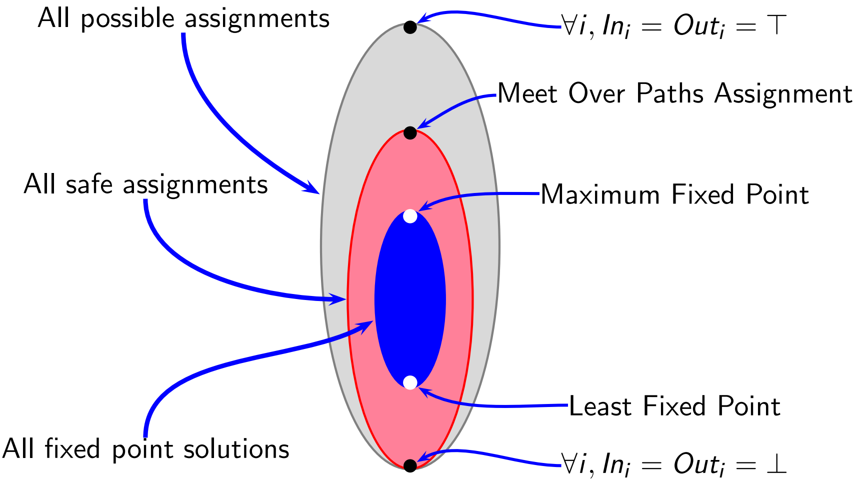
Possible Assignments as Solutions of Data Flow Analyses



Possible Assignments as Solutions of Data Flow Analyses

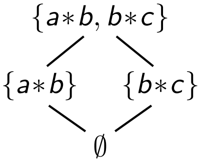


Possible Assignments as Solutions of Data Flow Analyses



Available Expr. Analysis Framework with Two Expressions

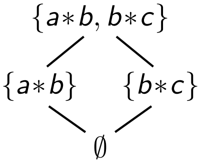
Lattice



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	$x - \{b*c\}$

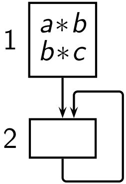
Available Expr. Analysis Framework with Two Expressions

Lattice



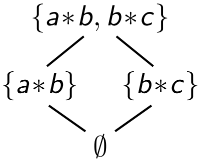
Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
$f_\top$	$\{a*b, b*c\}$	f_{id}	x
$f_\bot$	$\emptyset$	f_c	$x \cup \{a*b\}$
f_a	$\{a*b\}$	f_d	$x \cup \{b*c\}$
f_b	$\{b*c\}$	f_e	$x - \{a*b\}$
		f_f	$x - \{b*c\}$

Program



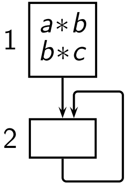
Available Expr. Analysis Framework with Two Expressions

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	$x - \{b*c\}$

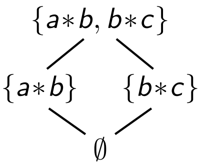
Program



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

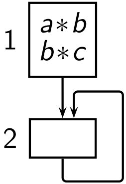
Available Expr. Analysis Framework with Two Expressions

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

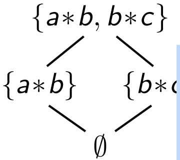


Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

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

Available Expr. Analysis Framework with Two Expressions

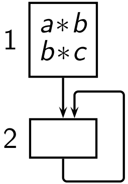
Lattice



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

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

Program

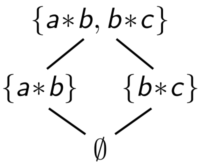


Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

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

Available Expr. Analysis Framework with Two Expressions

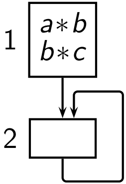
Lattice



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

Program

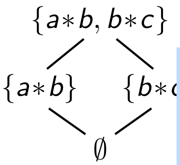


Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

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

Available Expr. Analysis Framework with Two Expressions

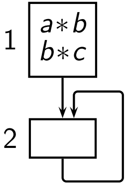
Lattice



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

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

Program

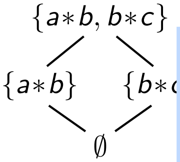


Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

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

Available Expr. Analysis Framework with Two Expressions

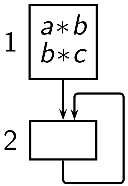
Lattice



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

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

Program

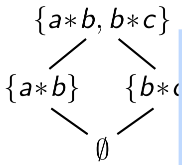


Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

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

Available Expr. Analysis Framework with Two Expressions

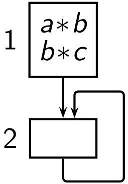
Lattice



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

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

Program

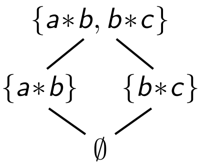


Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

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

Available Expr. Analysis Framework with Two Expressions

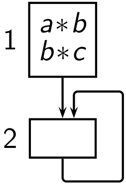
Lattice



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

- Not a fixed point assignment

Program



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

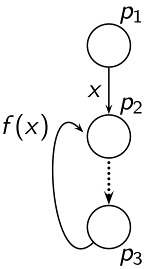
Some Possible Assignments						
	A_1	A_2	A_3	A_4	A_5	A_6
In_1	00	00	00	00	00	00
Out_1	11	00	11	11	11	11
In_2	11	00	00	10	01	01
Out_2	11	00	00	10	01	10

Existence of an MoP Assignment

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

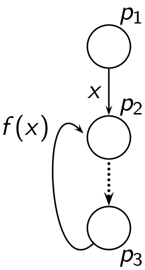
- If all paths reaching p are acyclic, then existence of solution trivially follows from the definition of the function space.
- If cyclic paths also reach p, then there are an infinite number of unbounded paths.
  $\Rightarrow$ Need to define loop closures.

Loop Closures of Flow Functions



Paths Terminating at p_2	Data Flow Value
p_1, p_2	x
p_1, p_2, p_3, p_2	$f(x)$
$p_1, p_2, p_3, p_2, p_3, p_2$	$f(f(x)) = f^2(x)$
$p_1, p_2, p_3, p_2, p_3, p_2, p_3, p_2$	$f(f(f(x))) = f^3(x)$
$\ldots$	$\ldots$

Loop Closures of Flow Functions

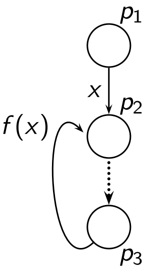


Paths Terminating at p_2	Data Flow Value
p_1, p_2	x
p_1, p_2, p_3, p_2	$f(x)$
$p_1, p_2, p_3, p_2, p_3, p_2$	$f(f(x)) = f^2(x)$
$p_1, p_2, p_3, p_2, p_3, p_2, p_3, p_2$	$f(f(f(x))) = f^3(x)$
$\ldots$	$\ldots$

- For static analysis we need to summarize the value at p_2 by a value which is safe after any iteration.

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

Loop Closures of Flow Functions



Paths Terminating at p_2	Data Flow Value
p_1, p_2	x
p_1, p_2, p_3, p_2	$f(x)$
$p_1, p_2, p_3, p_2, p_3, p_2$	$f(f(x)) = f^2(x)$
$p_1, p_2, p_3, p_2, p_3, p_2, p_3, p_2$	$f(f(f(x))) = f^3(x)$
...	...

- For static analysis we need to summarize the value at p_2 by a value which is safe after any iteration.

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

- f^* is called the loop closure of f.

Loop Closures in Bit Vector Frameworks

- Flow functions in bit vector frameworks have constant Gen and Kill

$$
\begin{aligned}
f^*(x) &= x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots \\
f^2(x) &= f\,(\text{Gen} \cup (x - \text{Kill})) \\
&= \text{Gen} \cup ((\text{Gen} \cup (x - \text{Kill})) - \text{Kill}) \\
&= \text{Gen} \cup ((\text{Gen} - \text{Kill}) \cup (x - \text{Kill})) \\
&= \text{Gen} \cup (\text{Gen} - \text{Kill}) \cup (x - \text{Kill}) \\
&= \text{Gen} \cup (x - \text{Kill}) \quad = \quad f(x) \\
f^*(x) &= x \sqcap f(x)
\end{aligned}
$$

Loop Closures in Bit Vector Frameworks

- Flow functions in bit vector frameworks have constant Gen and Kill

$$
\begin{aligned}
f^*(x) &= x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots \\
f^2(x) &= f\,(\text{Gen} \cup (x - \text{Kill})) \\
&= \text{Gen} \cup ((\text{Gen} \cup (x - \text{Kill})) - \text{Kill}) \\
&= \text{Gen} \cup ((\text{Gen} - \text{Kill}) \cup (x - \text{Kill})) \\
&= \text{Gen} \cup (\text{Gen} - \text{Kill}) \cup (x - \text{Kill}) \\
&= \text{Gen} \cup (x - \text{Kill}) \quad = \quad f(x) \\
f^*(x) &= x \sqcap f(x)
\end{aligned}
$$

- *Loop Closures of Bit Vector Frameworks are 2-bounded.*

Loop Closures in Bit Vector Frameworks

- Flow functions in bit vector frameworks have constant Gen and Kill

$$
\begin{aligned}
f^*(x) &= x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots \\
f^2(x) &= f\,(\text{Gen} \cup (x - \text{Kill})) \\
&= \text{Gen} \cup ((\text{Gen} \cup (x - \text{Kill})) - \text{Kill}) \\
&= \text{Gen} \cup ((\text{Gen} - \text{Kill}) \cup (x - \text{Kill})) \\
&= \text{Gen} \cup (\text{Gen} - \text{Kill}) \cup (x - \text{Kill}) \\
&= \text{Gen} \cup (x - \text{Kill}) \quad = \quad f(x) \\
f^*(x) &= x \sqcap f(x)
\end{aligned}
$$

- *Loop Closures of Bit Vector Frameworks are 2-bounded.*
- Intuition: Since Gen and Kill are constant, same things are generated or killed in every application of f.

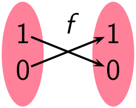
  Multiple applications of f are not required unless the input value changes.

Bounded Loop Closures May not be Computable

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

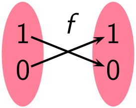
Bounded Loop Closures May not be Computable

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



Bounded Loop Closures May not be Computable

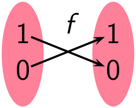
- If f is not monotonic, the computation may not converge



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

Bounded Loop Closures May not be Computable

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

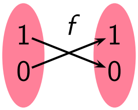


x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\dots$
1	0	1	0	1	$\dots$

$\Rightarrow f^*(x) = x \sqcap f(x) = 0$ Solution exists

Bounded Loop Closures May not be Computable

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



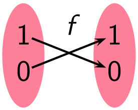
x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\dots$
1	0	1	0	1	$\dots$

$\Rightarrow f^*(x) = x \sqcap f(x) = 0$ Solution exists

- Iteratively computing the solution

Bounded Loop Closures May not be Computable

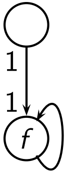
- If f is not monotonic, the computation may not converge



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

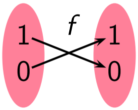
$\Rightarrow f^*(x) = x \sqcap f(x) = 0$ Solution exists

- Iteratively computing the solution



Bounded Loop Closures May not be Computable

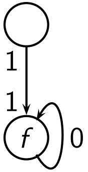
- If f is not monotonic, the computation may not converge



x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	...
1	0	1	0	1	...

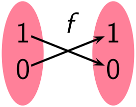
$\Rightarrow f^*(x) = x \sqcap f(x) = 0$ Solution exists

- Iteratively computing the solution



Bounded Loop Closures May not be Computable

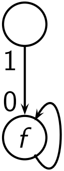
- If f is not monotonic, the computation may not converge



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

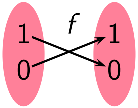
$\Rightarrow f^*(x) = x \sqcap f(x) = 0$ Solution exists

- Iteratively computing the solution



Bounded Loop Closures May not be Computable

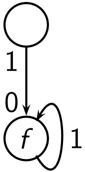
- If f is not monotonic, the computation may not converge



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

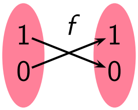
$\Rightarrow f^*(x) = x \sqcap f(x) = 0$ Solution exists

- Iteratively computing the solution



Bounded Loop Closures May not be Computable

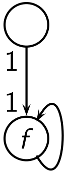
- If f is not monotonic, the computation may not converge



x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	...
1	0	1	0	1	...

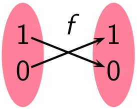
$\Rightarrow f^*(x) = x \sqcap f(x) = 0$ Solution exists

- Iteratively computing the solution



Bounded Loop Closures May not be Computable

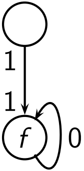
- If f is not monotonic, the computation may not converge



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

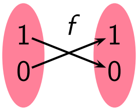
$\Rightarrow f^*(x) = x \sqcap f(x) = 0$ Solution exists

- Iteratively computing the solution



Bounded Loop Closures May not be Computable

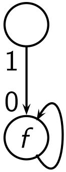
- If f is not monotonic, the computation may not converge



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

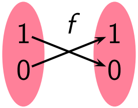
$\Rightarrow f^*(x) = x \sqcap f(x) = 0$ Solution exists

- Iteratively computing the solution



Bounded Loop Closures May not be Computable

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

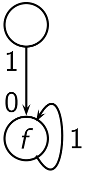


x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	...
1	0	1	0	1	...

$\Rightarrow f^*(x) = x \sqcap f(x) = 0$ Solution exists

- Iteratively computing the solution

  The values in the loop keep changing



More on Loop Closure Boundedness

Boundedness of f requires the existence of some k such that

$$f^*(x) = x \sqcap f(x) \sqcap f^2(x) \sqcap \ldots \sqcap f^{k-1}(x)$$

Given, monotonic f, loop closures are bounded because of any of the following:

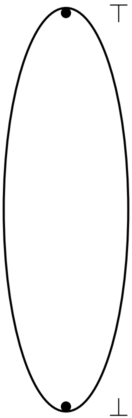
- $x \sqsubseteq f(x)$. All applications of f can be ignored
- $x \sqsupseteq f(x)$. In this case, $x, f(x), f^2(x), \ldots$ follow a descending chain. If descending chains are bounded, loop closures are bounded.
- x and $f(x)$ are incomparable. In this case $\displaystyle\prod_{i=0}^{j} f^j(x)$ follows a strictly descending chain. If descending chains are bounded, loop closures are bounded.

Existence and Computation of the Maximum Fixed Point

For monotonic $f : L \mapsto L$, if all descending chains are finite, then $MFP(f) = f^{k+1}(\top) = f^{k}(\top)$ such that $f^{j+1}(\top) \neq f^{j}(\top)$, $j < k$.

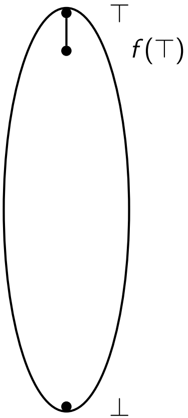
Existence and Computation of the Maximum Fixed Point

For monotonic $f : L \mapsto L$, if all descending chains are finite, then $MFP(f) = f^{k+1}(\top) = f^{k}(\top)$ such that $f^{j+1}(\top) \neq f^{j}(\top)$, $j < k$.



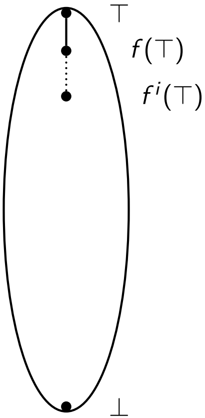
Existence and Computation of the Maximum Fixed Point

For monotonic $f : L \mapsto L$, if all descending chains are finite, 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

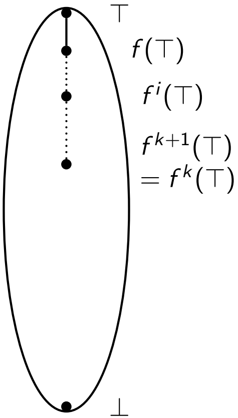
For monotonic $f : L \mapsto L$, if all descending chains are finite, then $MFP(f) = f^{k+1}(\top) = f^k(\top)$ such that $f^{j+1}(\top) \neq f^j(\top)$, $j < k$.



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

Existence and Computation of the Maximum Fixed Point

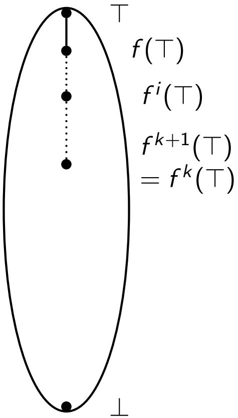
For monotonic $f : L \mapsto L$, if all descending chains are finite, then
$MFP(f) = f^{k+1}(\top) = f^k(\top)$ such that $f^{j+1}(\top) \neq f^j(\top)$, $j < k$.



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

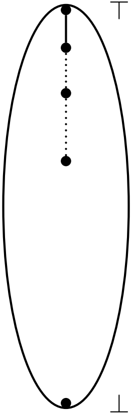
For monotonic $f : L \mapsto L$, if all descending chains are finite, then $MFP(f) = f^{k+1}(\top) = f^k(\top)$ such that $f^{j+1}(\top) \neq f^j(\top)$, $j < k$.



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

For monotonic $f : L \mapsto L$, if all descending chains are finite, then
$MFP(f) = f^{k+1}(\top) = f^k(\top)$ such that $f^{j+1}(\top) \neq f^j(\top)$, $j < k$.



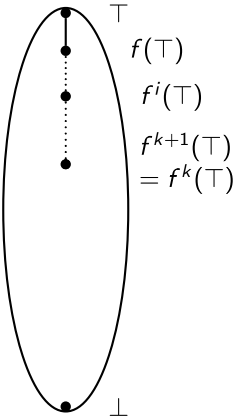
- $\top \sqsupseteq f(\top) \sqsupseteq f^2(\top) \sqsupseteq f^3(\top) \sqsupseteq f^4(\top) \sqsupseteq \dots$
- Since descending chains are finite, there must exist $f^k(\top)$ such that $f^{k+1}(\top) = f^k(\top)$ and $f^{j+1}(\top) \neq f^j(\top)$, $j < k$.
- If p is a fixed point of f then $p \sqsubseteq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

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

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

Existence and Computation of the Maximum Fixed Point

For monotonic $f : L \mapsto L$, if all descending chains are finite, 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 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 $p \sqsubseteq f^k(\top)$.

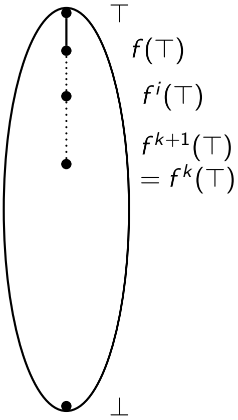
  Proof strategy: Induction on i for $f^i(\top)$

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

$$\begin{array}{rrcll} & f(p) & \sqsubseteq & f(f^i(\top)) & (f \text{ is monotonic}) \\ \Rightarrow & p & \sqsubseteq & f(f^i(\top)) & (f(p) = p) \\ \Rightarrow & p & \sqsubseteq & f^{i+1}(\top) & \end{array}$$

Existence and Computation of the Maximum Fixed Point

For monotonic $f : L \mapsto L$, if all descending chains are finite, 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 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 $f^i(\top) \sqsupseteq p$.
  - Proof:
  $$\begin{array}{rrcll} & f(p) & \sqsubseteq & f(f^i(\top)) & (f \text{ is monotonic}) \\ \Rightarrow & p & \sqsubseteq & f(f^i(\top)) & (f(p) = p) \\ \Rightarrow & p & \sqsubseteq & f^{i+1}(\top) & \end{array}$$
- $\Rightarrow f^{k+1}(\top)$ is the MFP.

Fixed Points Computation: Flow Functions Vs. Equations

- Recall that

  $MFP(f) = f^{k+1}(\top) = f^k(\top)$ 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

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

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

Fixed Points Computation: Flow Functions Vs. Equations

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

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

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

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

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

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

Fixed Points Computation: Flow Functions Vs. Equations

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

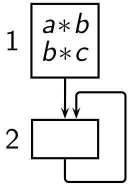
$$\mathcal{X} = \mathcal{F}(\mathcal{X})$$

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

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

Available Expr. Analysis Framework with Two Expressions

Program

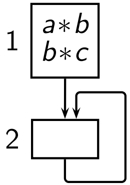


- Data Flow Equation $\mathcal{X} = \mathcal{F}(\mathcal{X})$ is

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

Available Expr. Analysis Framework with Two Expressions

Program



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

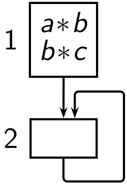
$$\mathcal{F}(\langle In_1, Out_1, In_2, Out_2\rangle) = \langle 00, 11, Out_2, Out_2\rangle$$

- The maximum fixed point assignment is

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

Available Expr. Analysis Framework with Two Expressions

Program



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

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

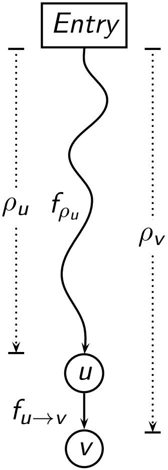
- The maximum fixed point assignment is

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

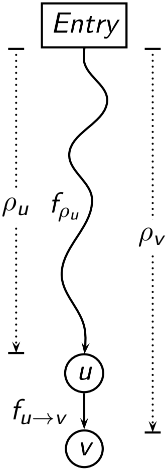
- The minimum fixed point assignment is

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

Safety of MFP Assignment: MFP $\sqsubseteq$ MoP

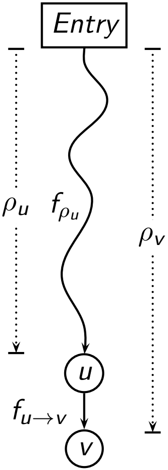


Safety of MFP Assignment: MFP $\sqsubseteq$ MoP



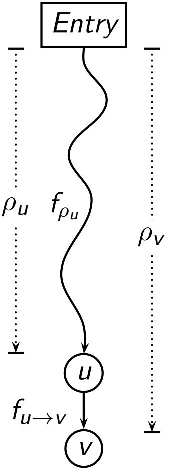
- $MoP(v) \quad = \displaystyle\bigsqcap_{\rho \in Paths(v)} f_\rho(BI)$

Safety of MFP Assignment: MFP $\sqsubseteq$ MoP



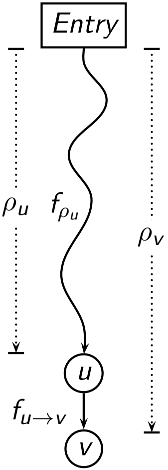
- $MoP(v) \quad = \prod_{\rho \in Paths(v)} f_\rho(BI)$
- Proof Obligation: $\forall \rho_v \; MFP(v) \sqsubseteq f_{\rho_v}(BI)$

Safety of MFP Assignment: MFP $\sqsubseteq$ MoP



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

Safety of MFP Assignment: MFP $\sqsubseteq$ MoP



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

  Base case: Path of length 0.

  $MFP(Entry) = MoP(Entry) = BI$

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

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

Part 8

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.

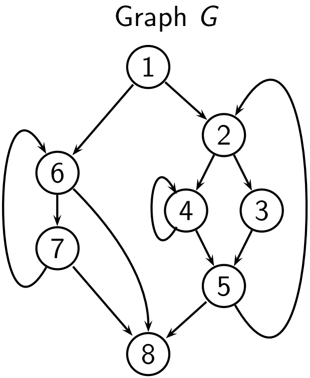
Elimination Methods of Performing Data Flow Analysis

Delayed computations of dependent data flow values of dependent nodes.

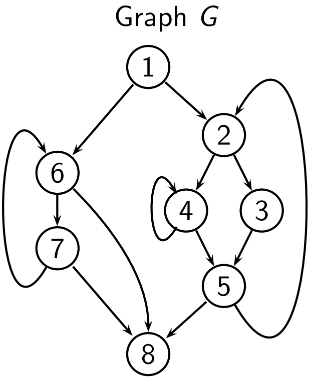
Find suitable single-entry regions.

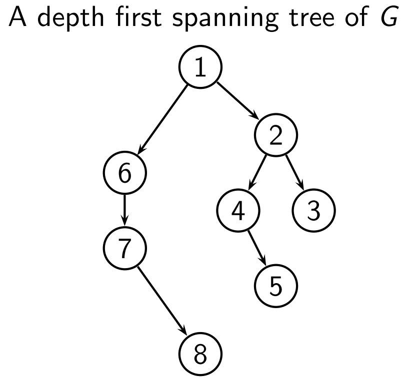
- *Interval Based Analysis*. Uses graph partitioning.
- T_1, T_2 *Based Analysis*. Uses graph parsing.

Classification of Edges in a Graph



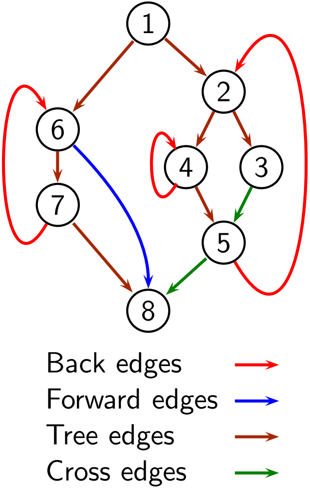
Classification of Edges in a Graph



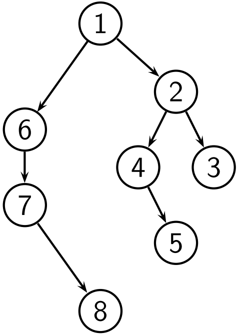


Classification of Edges in a Graph

Graph G

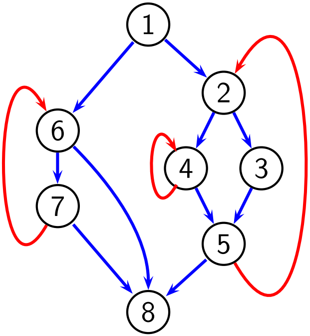


A depth first spanning tree of G

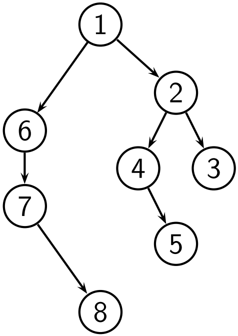


Classification of Edges in a Graph

Graph G



A depth first spanning tree of G



Back edges

Forward edges

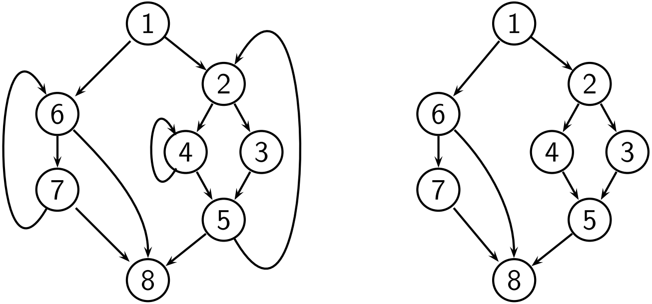
For data flow analysis, we club *tree*, *forward*, and *cross* edges into *forward* edges. Thus we have just forward or back edges in a control flow graph

Reverse Post Order Traversal

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

Graph G

G' obtained after removing back edges of G



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

Round Robin Iterative Algorithm

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

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 $\{\ \ change = false$
7 $\quad$ **for** $j = 1$ to $N-1$ **do**
8 $\quad \{\ \ temp = \displaystyle\bigsqcap_{p \in pred(j)} f_p(In_p)$
9 $\quad\quad$ **if** $temp \neq In_j$ **then**
10 $\quad\quad \{\ \ In_j = temp$
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)
- Reverse postorder (rpo) traversal for efficiency (line 7)

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 $\quad \{ \quad temp = \displaystyle\prod_{p \in pred(j)} f_p(In_p)$
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)
- 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 spaning 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 spaning 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 spaning 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

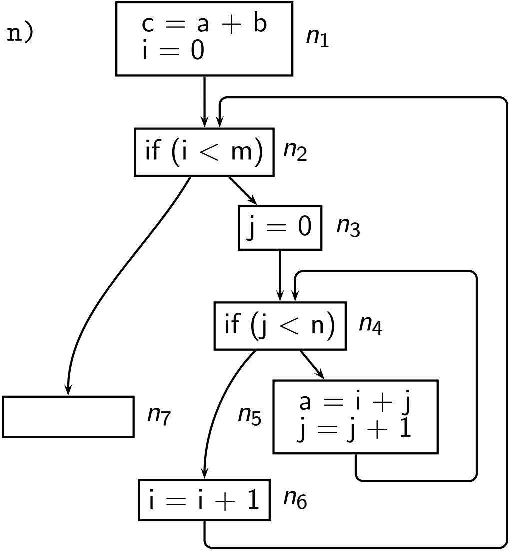
- What about bidirectional bit vector frameworks?
- What about other frameworks?

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;
      }
   }
```

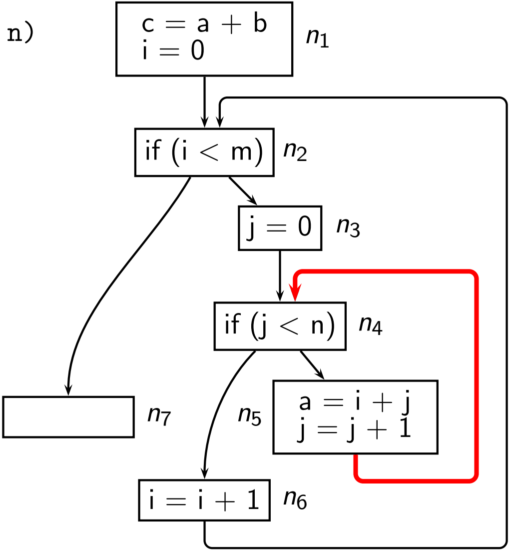
Example C Program with d(G,T) = 2

```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;
      }
   }
```



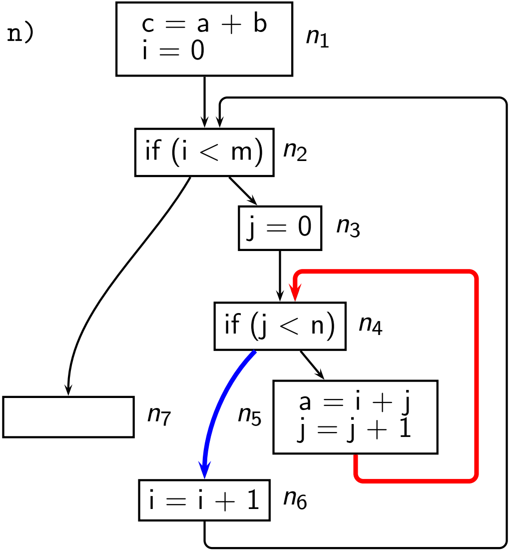
Example C Program with d(G,T) = 2

```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;
      }
   }
```



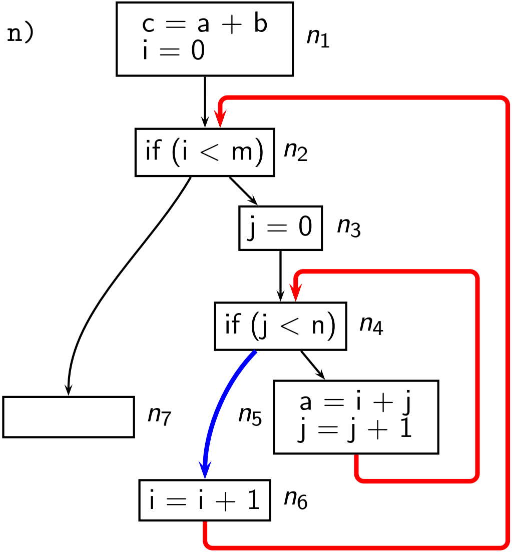
Example C Program with d(G,T) = 2

```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

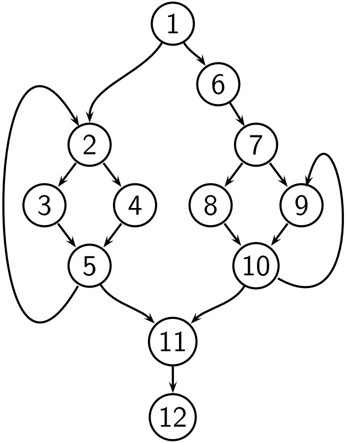
```
void fun(int m, int n)
{
   int i,j,a,b,c;
   c=a+b;
   i=0;
   while(i<m)
   {
         j=0;
         while(j<n)
         {
            a=i+j;
            j=j+1;
         }
         i=i+1;
      }
   }
```



3 + 1 iterations for available expressions analysis

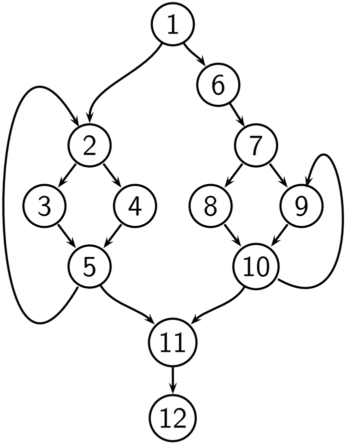
Complexity of Bidirectional Bit Vector Frameworks

Example: Consider the following CFG for PRE



Complexity of Bidirectional Bit Vector Frameworks

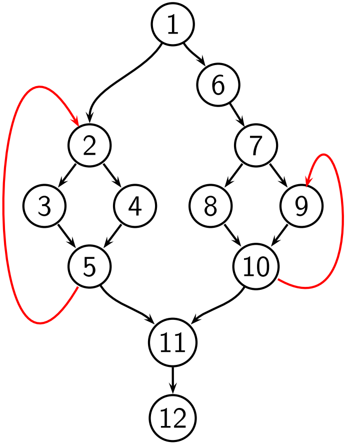
Example: Consider the following CFG for PRE



- Node numbers are in reverse post order

Complexity of Bidirectional Bit Vector Frameworks

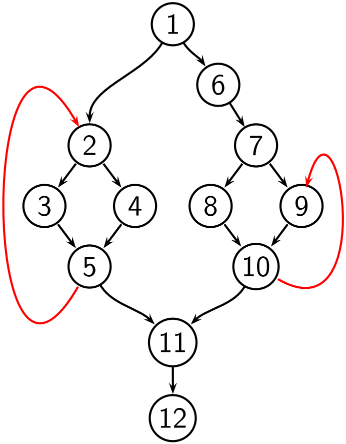
Example: Consider the following CFG for PRE



- 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

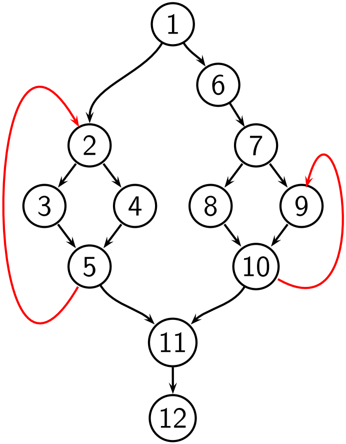
Example: Consider the following CFG for PRE



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

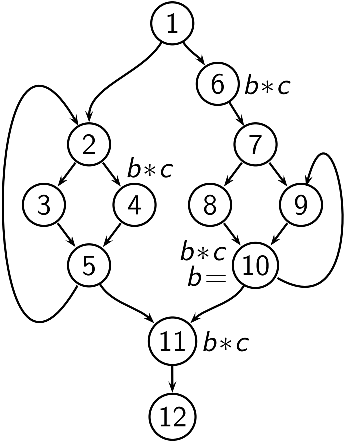
Complexity of Bidirectional Bit Vector Frameworks

Example: Consider the following CFG for PRE



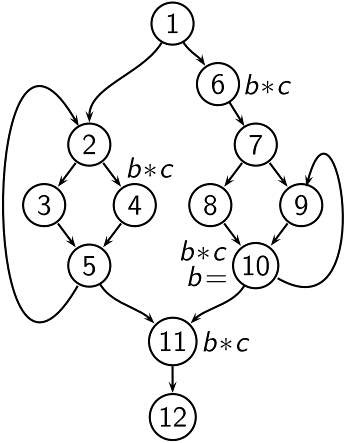
- Node numbers are in reverse post order
- Back edges in the graph are $n_5 \rightarrow n_2$ and $n_{10} \rightarrow n_9$.
- $d(G, T) = 1$
- Actual iterations : 5

Complexity of Bidirectional Bit Vector Frameworks



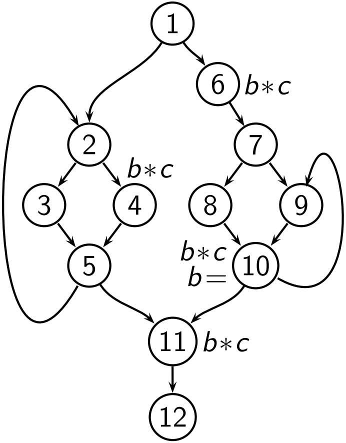
	Pairs of *Out*,*In* Values							
	Initia-lization	Changes in Iterations					Final values & transformation	
		#1	#2	#3	#4	#5		
	O,I	O,I	O,I	O,I	O,I	O,I	O,I	
12	0,1							
11	1,1							
10	1,1							
9	1,1							
8	1,1							
7	1,1							
6	1,1							
5	1,1							
4	1,1							
3	1,1							
2	1,1					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

Complexity of Bidirectional Bit Vector Frameworks



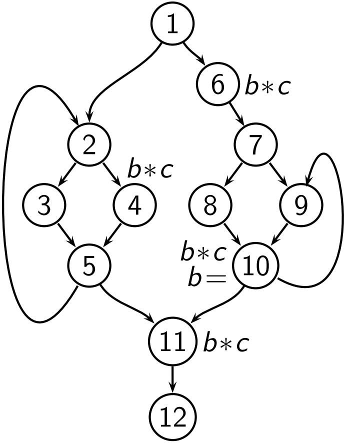
	Pairs of *Out*,*In* Values							
	Initia-lization	Changes in Iterations					Final values & transformation	
		#1	#2	#3	#4	#5		
	O,I	O,I	O,I	O,I	O,I	O,I	O,I	
12	0,1	0,0						
11	1,1	0,1						
10	1,1							
9	1,1							
8	1,1							
7	1,1							
6	1,1	1,0						
5	1,1							
4	1,1							
3	1,1							
2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

Complexity of Bidirectional Bit Vector Frameworks



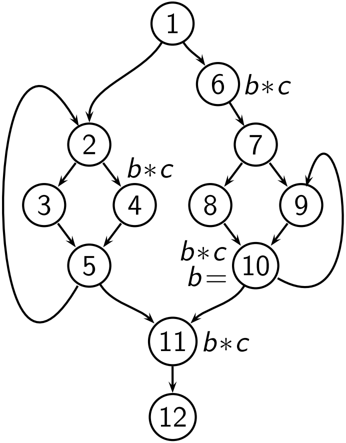
	Pairs of *Out*,*In* Values							
	Initialization	Changes in Iterations					Final values & transformation	
		#1	#2	#3	#4	#5		
	O,I	O,I	O,I	O,I	O,I	O,I	O,I	
12	0,1	0,0						
11	1,1	0,1						
10	1,1							
9	1,1							
8	1,1							
7	1,1							
6	1,1	1,0						
5	1,1							
4	1,1							
3	1,1							
2	1,1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

Complexity of Bidirectional Bit Vector Frameworks



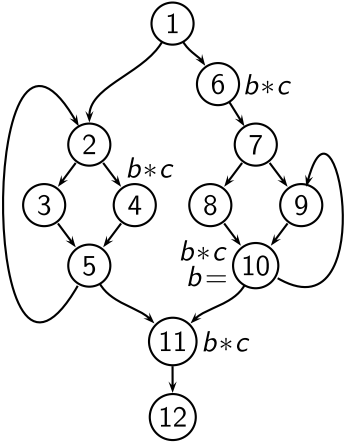
	Pairs of *Out*,*In* Values							
	Initia-lization	Changes in Iterations					Final values & transformation	
		#1	#2	#3	#4	#5		
	O,I	O,I	O,I	O,I	O,I	O,I	O,I	
12	0,1	0,0						
11	1,1	0,1						
10	1,1							
9	1,1							
8	1,1							
7	1,1							
6	1,1	1,0						
5	1,1			0,0				
4	1,1			0,1				
3	1,1			0,0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

Complexity of Bidirectional Bit Vector Frameworks



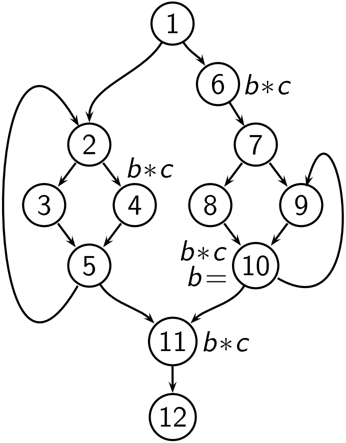
	Pairs of *Out*,*In* Values							
	Initia-lization	Changes in Iterations					Final values & transformation	
		#1	#2	#3	#4	#5		
	O,I	O,I	O,I	O,I	O,I	O,I	O,I	
12	0,1	0,0						
11	1,1	0,1			0,0			
10	1,1				0,1			
9	1,1				1,0			
8	1,1							
7	1,1				0,0			
6	1,1	1,0			0,0			
5	1,1			0,0				
4	1,1			0,1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



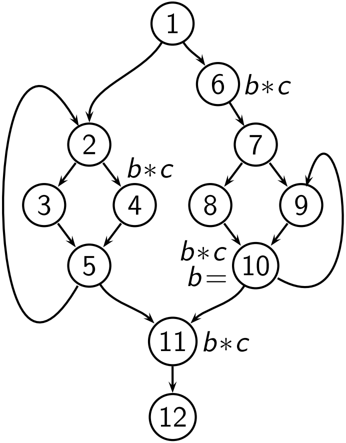
	Pairs of *Out*,*In* Values							
	Initia-lization	Changes in Iterations					Final values & transformation	
		#1	#2	#3	#4	#5		
	O,I	O,I	O,I	O,I	O,I	O,I	O,I	
12	0,1	0,0						
11	1,1	0,1			0,0			
10	1,1				0,1			
9	1,1				1,0			
8	1,1					1,0		
7	1,1				0,0			
6	1,1	1,0			0,0			
5	1,1			0,0				
4	1,1	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



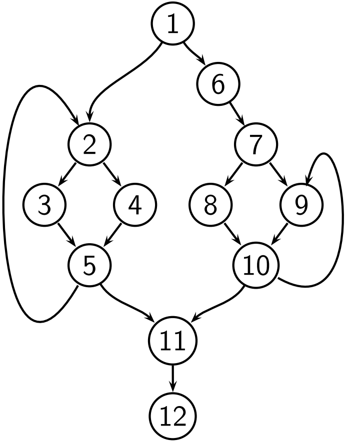
	Pairs of *Out*,*In* Values							
	Initia-lization	Changes in Iterations					Final values & transformation	
		#1	#2	#3	#4	#5		
	O,I	O,I	O,I	O,I	O,I	O,I	O,I	
12	0,1	0,0					0,0	
11	1,1	0,1			0,0		0,0	
10	1,1				0,1		0,1	
9	1,1				1,0		1,0	
8	1,1					1,0	1,0	
7	1,1				0,0		0,0	
6	1,1	1,0			0,0		0,0	
5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



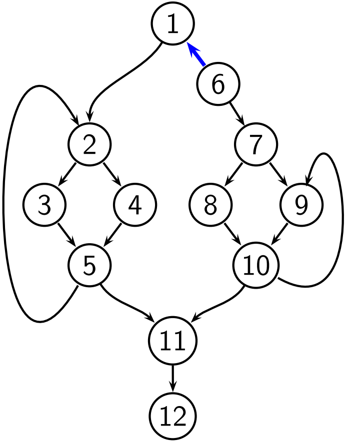
	Pairs of *Out*,*In* Values							
	Initia-lization	Changes in Iterations					Final values & transformation	
		#1	#2	#3	#4	#5		
	O,I	O,I	O,I	O,I	O,I	O,I	O,I	
12	0,1	0,0					0,0	
11	1,1	0,1			0,0		0,0	
10	1,1				0,1		0,1	Delete
9	1,1				1,0		1,0	Insert
8	1,1					1,0	1,0	Insert
7	1,1				0,0		0,0	
6	1,1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



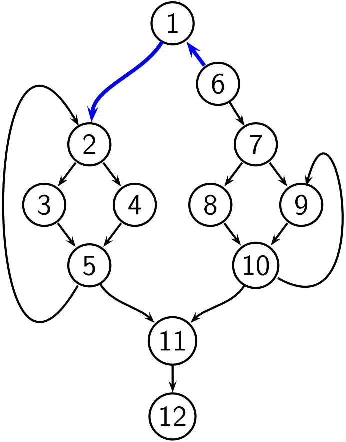
- $PavIn_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)

An Example of Information Flow in Our PRE Analysis



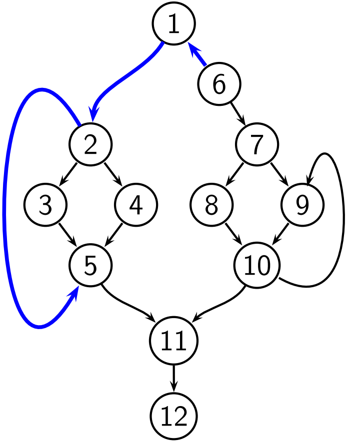
- $PavIn_6$ becomes 0 in the first itereation
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)

An Example of Information Flow in Our PRE Analysis



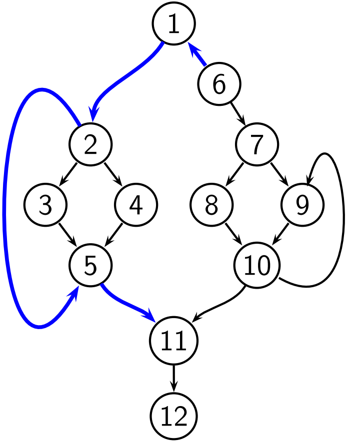
- *PavIn*$_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)

An Example of Information Flow in Our PRE Analysis



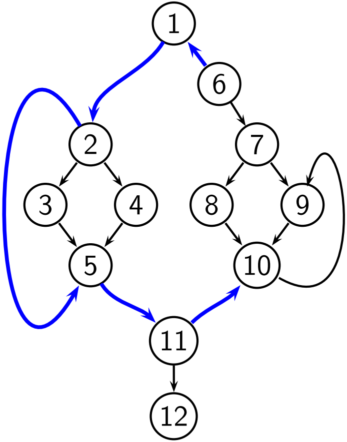
- $PavIn_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)

An Example of Information Flow in Our PRE Analysis



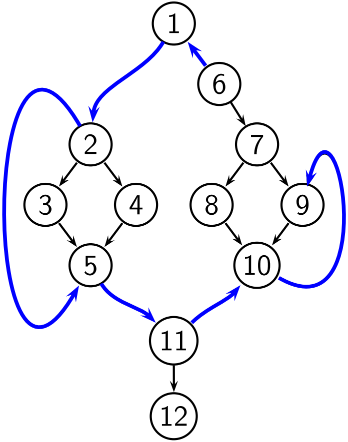
- $PavIn_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)

An Example of Information Flow in Our PRE Analysis



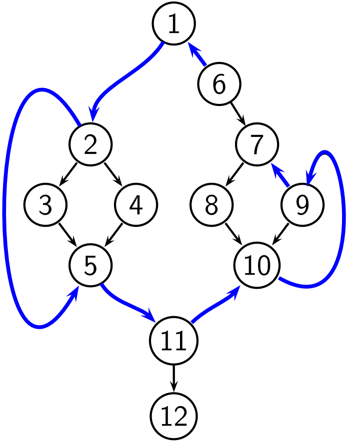
- $PavIn_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)

An Example of Information Flow in Our PRE Analysis



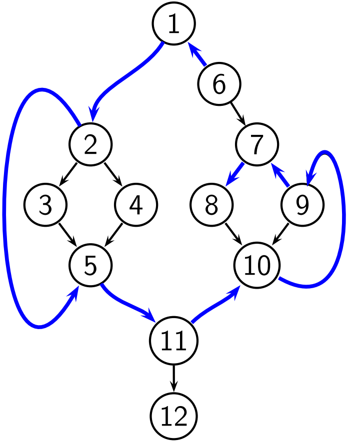
- $PavIn_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)

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

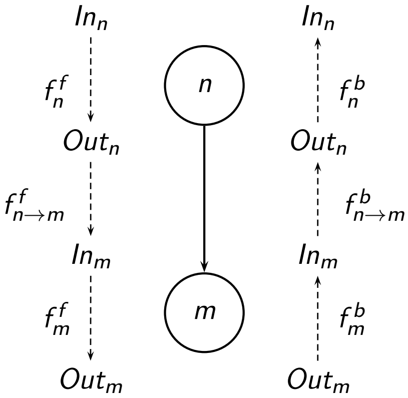
- A change in the data flow at a program point could be
  - *Generation of information*
    Change from $\top$ to a non-$\top$ due to local effect (i.e. $f(\top) \neq \top$)
  - *Propagation of information*
    Change from x to y such that $y \sqsubseteq x$ due to global effect

Information Flow and Information Flow Paths

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

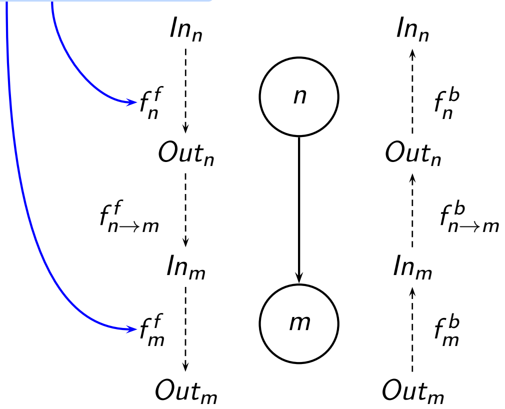
  Sequence of adjacent program points
  along which data flow values change
- A change in the data flow at a program point could be
  - *Generation of information*
    Change from $\top$ to a non-$\top$ due to local effect (i.e. $f(\top) \neq \top$)
  - *Propagation of information*
    Change from x to 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

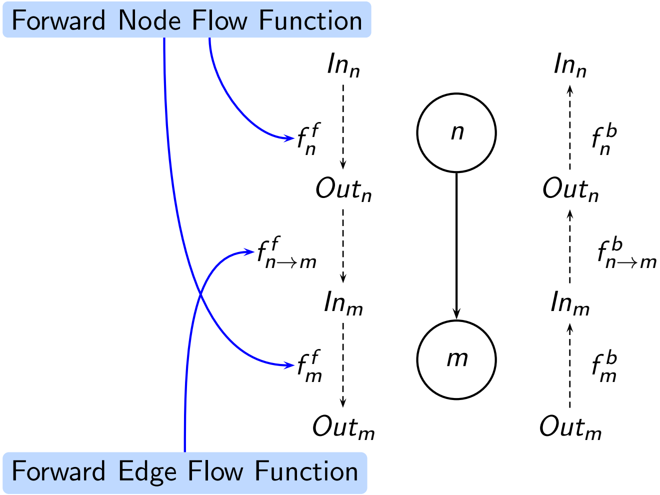


Edge and Node Flow Functions

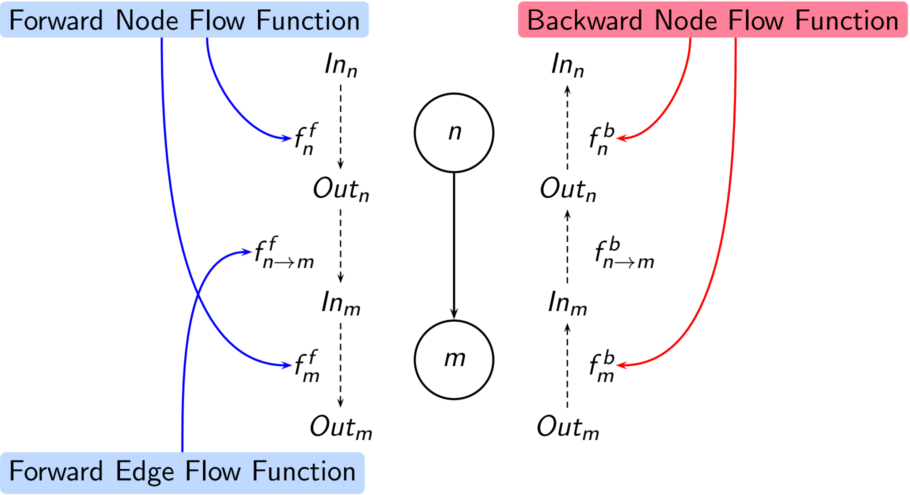
Forward Node Flow Function



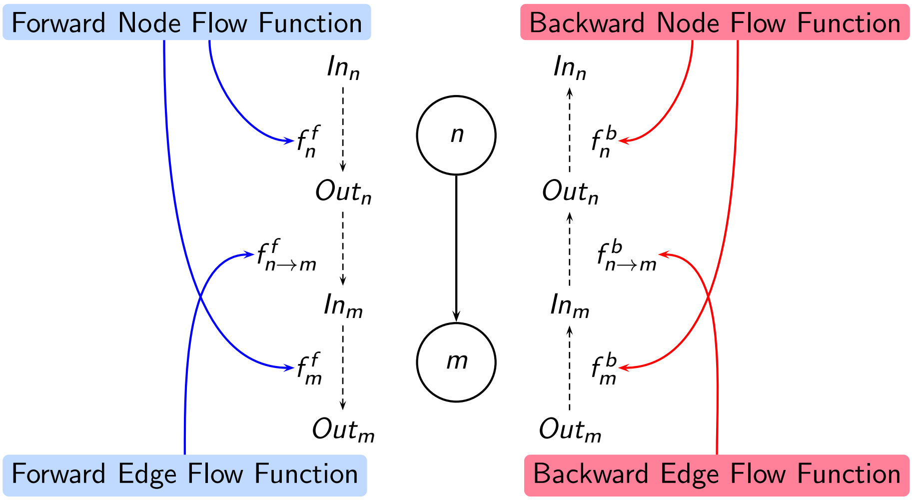
Edge and Node Flow Functions



Edge and Node Flow Functions



Edge and Node Flow Functions



General Data Flow Equations

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

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

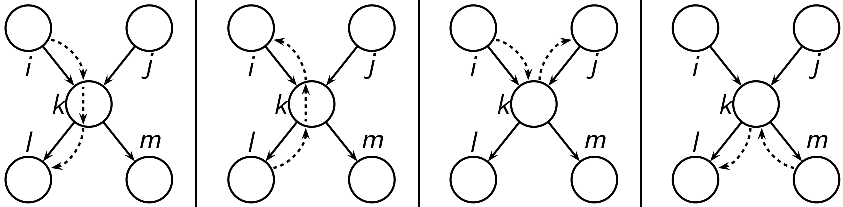
- Edge flow functions are typically identity

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

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

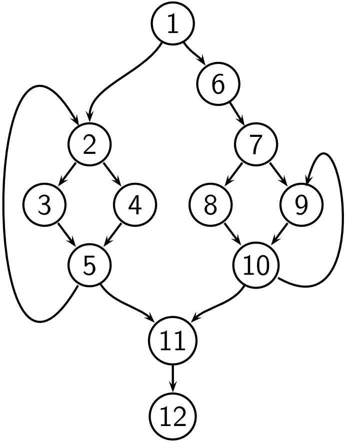
$$\forall x \in L,\ f(x) = \top$$

Modelling Information Flows Using Edge and Node Flow Functions



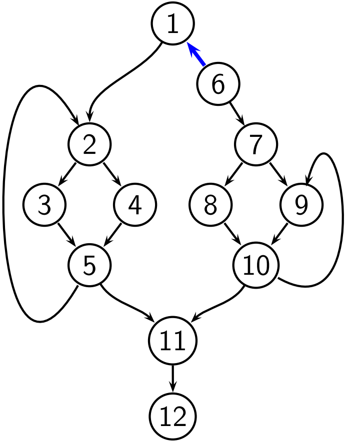
Forward	Backward	Bidirectional	Bidirectional
$f^f_{k \to l} \circ f^f_k \circ f^f_{i \to k}$	$f^b_{i \to k} \circ f^b_k \circ f^b_{k \to l}$	$f^f_{i \to k} \circ f^b_{k \to j}$	$f^b_{k \to m} \circ f^f_{k \to l}$

Information Flow Paths in PRE



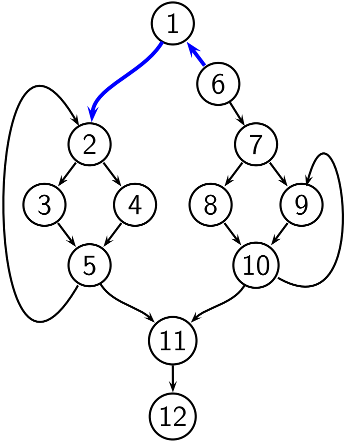
- Information could flow along arbitrary paths

Information Flow Paths in PRE



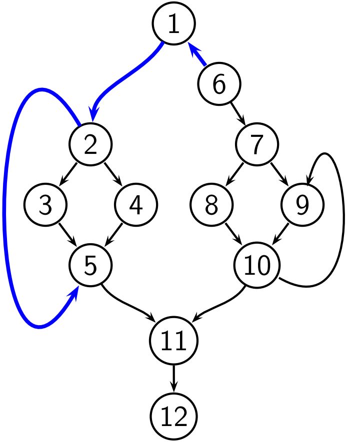
- Information could flow along arbitrary paths

Information Flow Paths in PRE



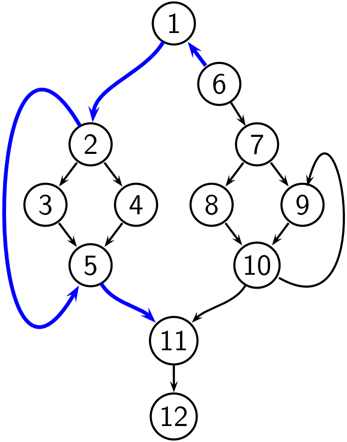
- Information could flow along arbitrary paths

Information Flow Paths in PRE



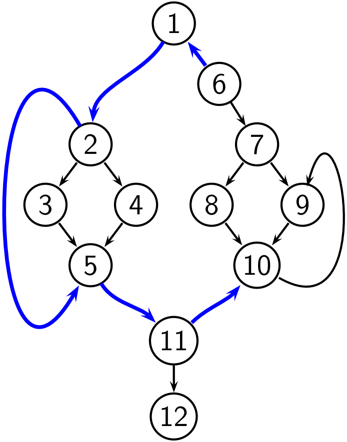
- Information could flow along arbitrary paths

Information Flow Paths in PRE



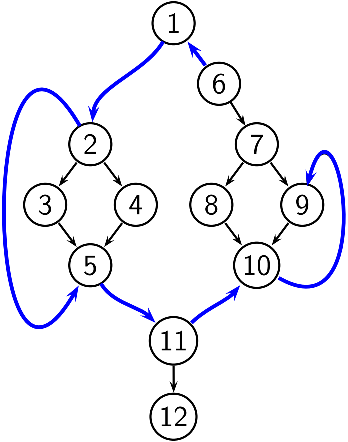
- Information could flow along arbitrary paths

Information Flow Paths in PRE



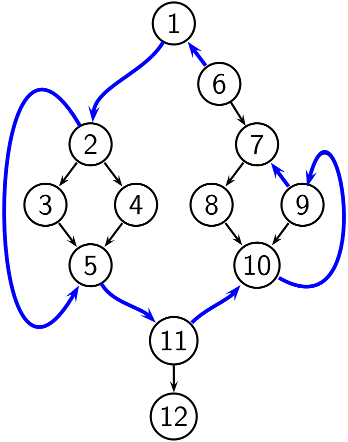
- Information could flow along arbitrary paths

Information Flow Paths in PRE



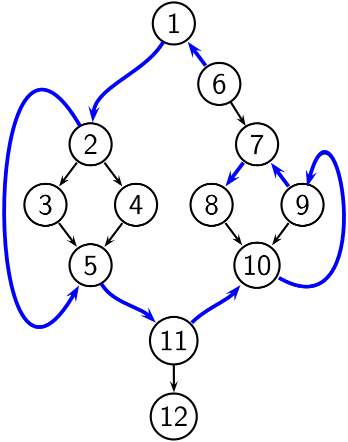
- Information could flow along arbitrary paths

Information Flow Paths in PRE



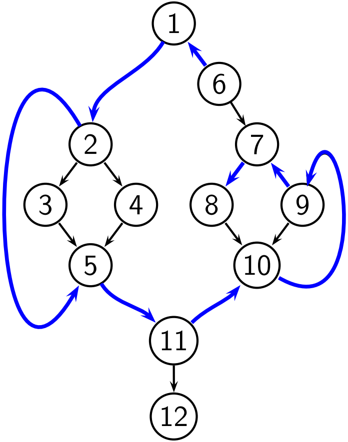
- Information could flow along arbitrary paths

Information Flow Paths in PRE



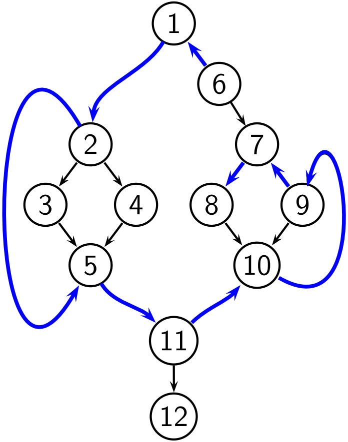
- Information could flow along arbitrary paths
- Theoretically predicted number : 144

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)

Lacuna with PRE Complexity

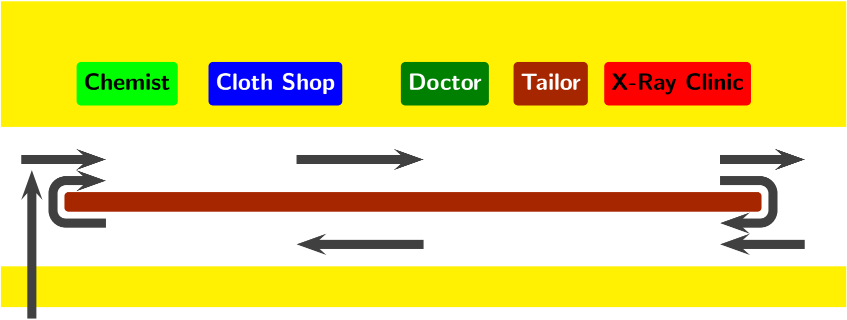
- Lacuna with PRE : Complexity $O(n^2)$ traversals.

  Practical graphs may have upto 50 nodes.

  - Predicted number of traversals : 2,500.
  - Practical number of traversals : ≤ 5.

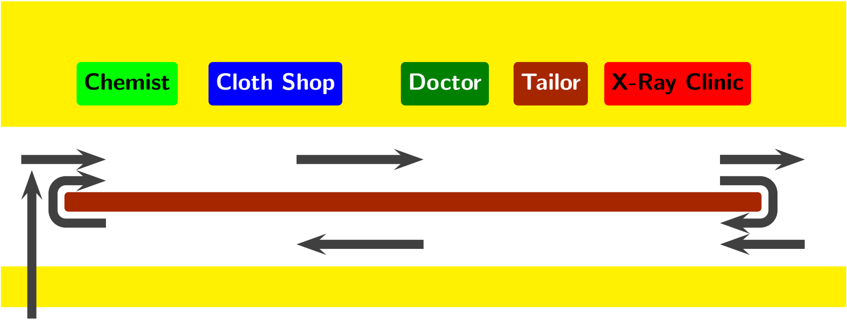
- No explanation for about 14 years despite dozens of efforts.
- Not much experimentation with performing advanced optimizations involving bidirectional dependency.

Complexity of Round Robin Iterative Method



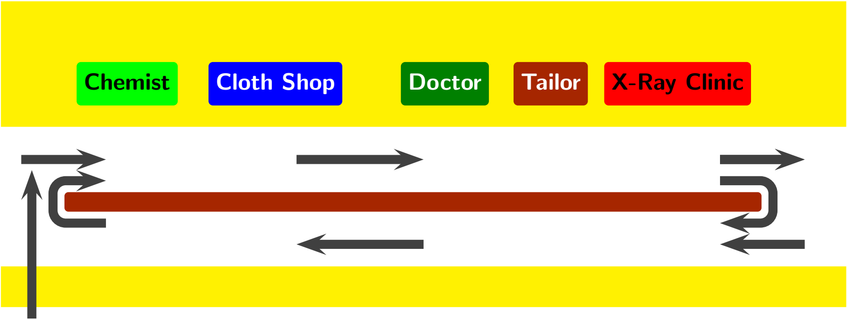
- Buy OTC (Over-The-Counter) medicine. No U-Turn 1 Trip

Complexity of Round Robin Iterative Method



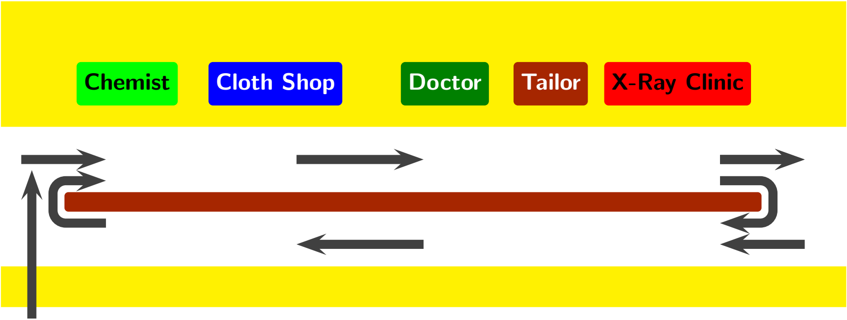
• Buy OTC (Over-The-Counter) medicine.	No U-Turn	1 Trip
• Buy cloth. Give it to the tailor for stitching.	No U-Turn	1 Trip

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

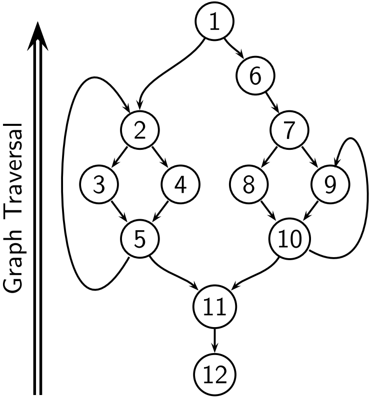
Information Flow Paths and Width of a Graph

- 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

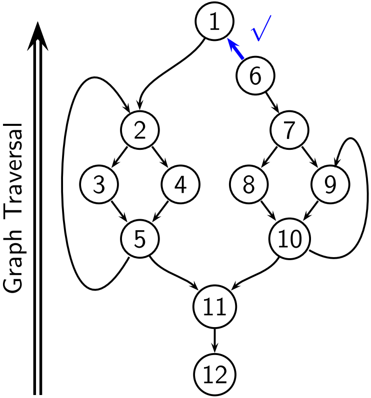
- A traversal $u \to v$ in an ifp is
  - *Compatible* if u is visited *before* v in the chosen graph traversal
  - *Incompatible* if u is visited *after* v in the chosen graph traversal
- Every incompatible edge traversal requires one additional iteration
- Width of a program flow graph with respect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



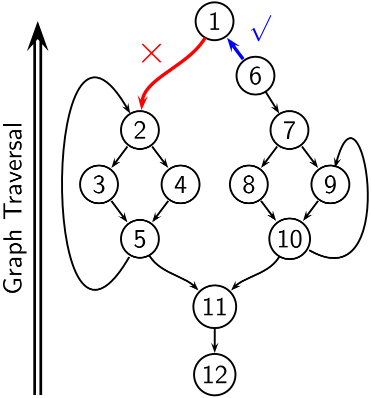
- Every "incompatible" edge traversal
  $\Rightarrow$ **One additional graph traversal**

Complexity of Bidirectional Bit Vector Frameworks



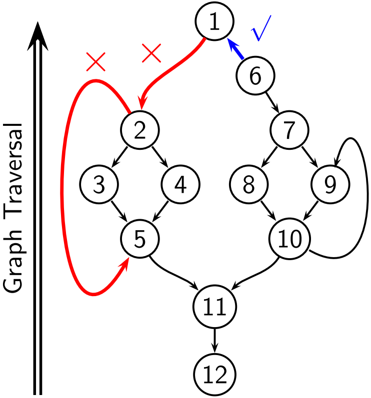
- **Every "incompatible" edge traversal $\Rightarrow$ One additional graph traversal**
- Max. Incompatible edge traversals $=$ *Width* of the graph $=$ **0?**
- Maximum number of traversals $=$ $1 +$ Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



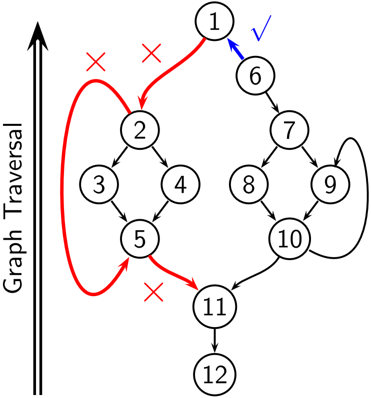
- $\boxed{\text{Every "incompatible" edge traversal } \Rightarrow \textbf{One additional graph traversal}}$
- Max. Incompatible edge traversals
  = *Width* of the graph = **1?**
- Maximum number of traversals =
  1 + Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



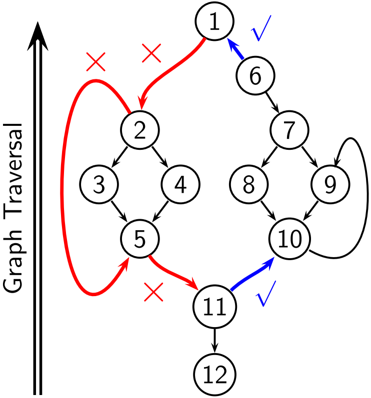
- Every "incompatible" edge traversal $\Rightarrow$ **One additional graph traversal**
- Max. Incompatible edge traversals $=$ *Width* of the graph $=$ **2?**
- Maximum number of traversals $=$ $1 +$ Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



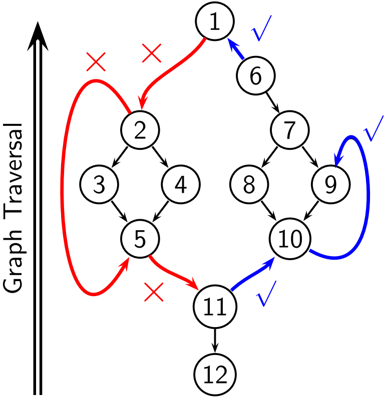
- $\boxed{\text{Every "incompatible" edge traversal} \Rightarrow \textbf{One additional graph traversal}}$
- Max. Incompatible edge traversals
  = *Width* of the graph = **3?**
- Maximum number of traversals =
  1 + Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



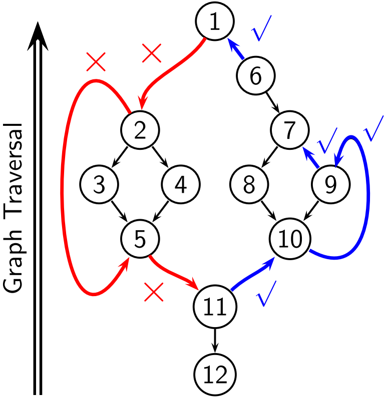
- Every "incompatible" edge traversal
  $\Rightarrow$ **One additional graph traversal**
- Max. Incompatible edge traversals
  $=$ *Width* of the graph $=$ **3?**
- Maximum number of traversals $=$
  $1 +$ Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



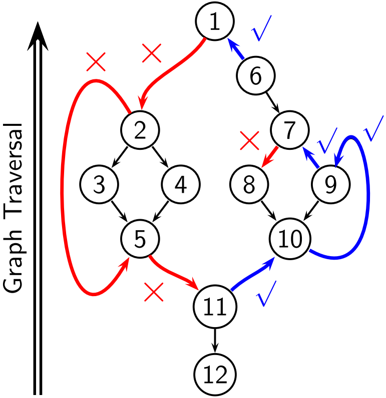
- **Every "incompatible" edge traversal**
  ⇒ One additional graph traversal
- Max. Incompatible edge traversals = *Width* of the graph = **3?**
- Maximum number of traversals = 1 + Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



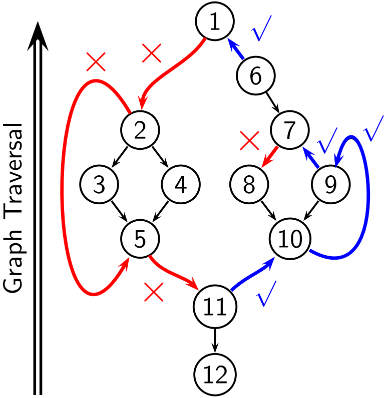
- Every "incompatible" edge traversal
  $\Rightarrow$ **One additional graph traversal**
- Max. Incompatible edge traversals
  $=$ *Width* of the graph $=$ **3?**
- Maximum number of traversals $=$
  $1 +$ Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



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

Complexity of Bidirectional Bit Vector Frameworks

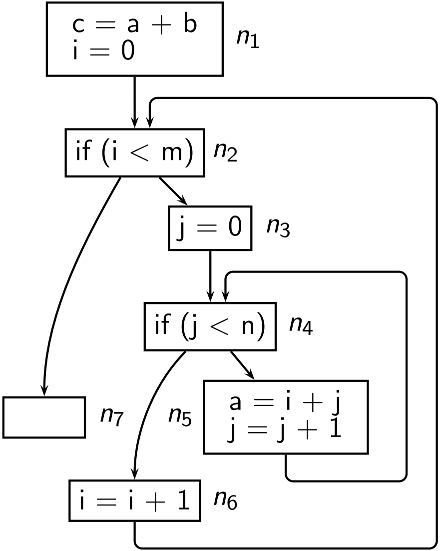


- Every "incompatible" edge traversal $\Rightarrow$ **One additional graph traversal**
- Max. Incompatible edge traversals $=$ *Width* of the graph $=$ **4**
- Maximum number of traversals $=$ $1 + 4 = 5$

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, Width $\leq$ Depth
  Width provides a tighter bound

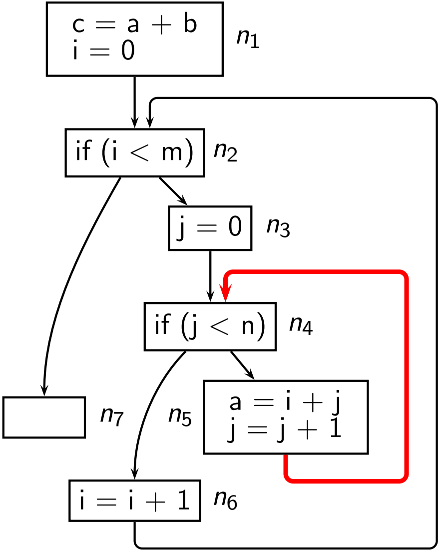
Width and Depth



Assuming reverse postorder traversal for available expressions analysis

- Depth = 2

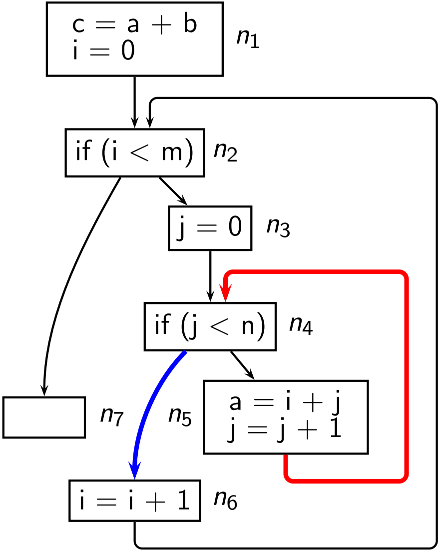
Width and Depth



Assuming reverse postorder traversal for available expressions analysis

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

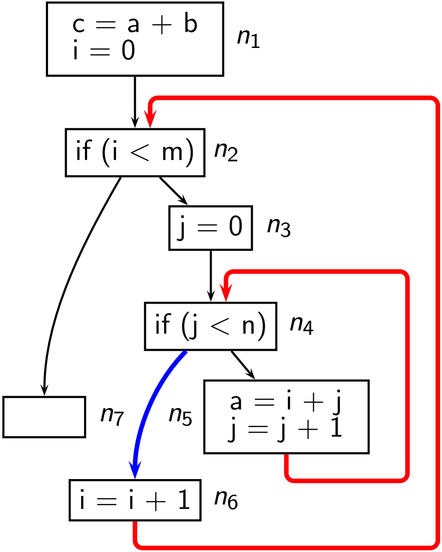
Width and Depth



Assuming reverse postorder traversal for available expressions analysis

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

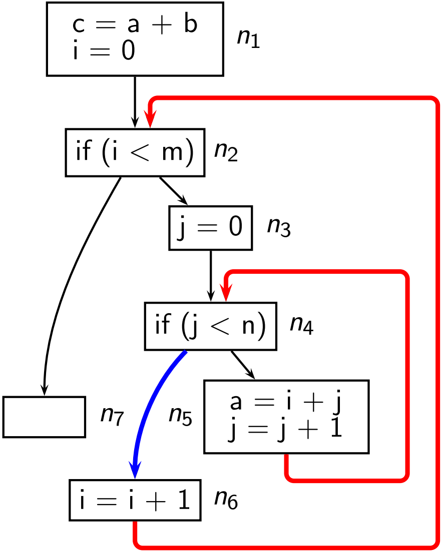
Width and Depth



Assuming reverse postorder traversal for available expressions analysis

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

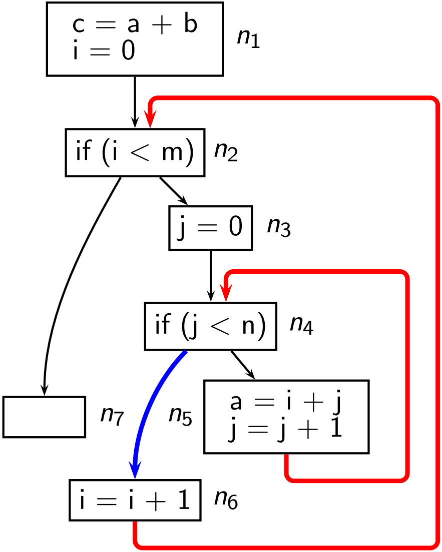
Width and Depth



Assuming reverse postorder traversal for available expressions analysis

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

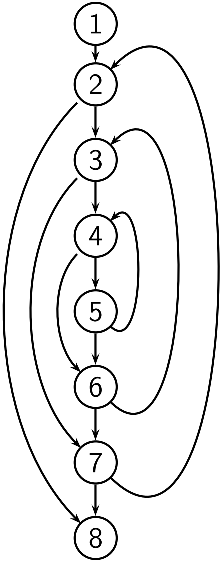
Width and Depth



Assuming reverse postorder traversal for available expressions analysis

- Depth = 2
- Information generation point n_5 kills expression "a + b"
- Information propagation path $n_5 \rightarrow n_4 \rightarrow n_5 \rightarrow n_2$
  No Gen or Kill for "a + b" along this path
- Width = 2
- What about "j + 1"?
- Not available on entry to the loop

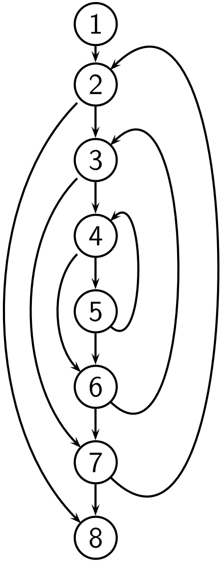
Width and Depth



Structures resulting from repeat-until loops with premature exits

- Depth = 3

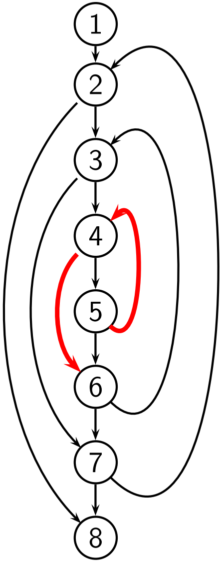
Width and Depth



Structures resulting from repeat-until loops with premature exits

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

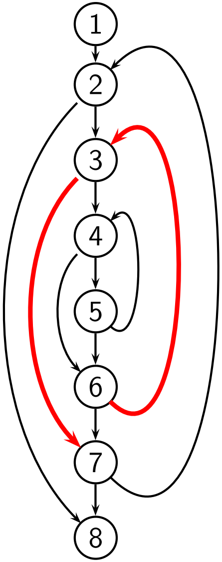
Width and Depth



Structures resulting from repeat-until loops with premature exits

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

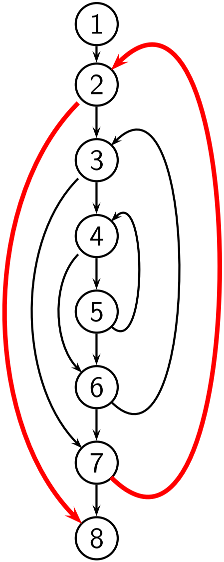
Width and Depth



Structures resulting from repeat-until loops with premature exits

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

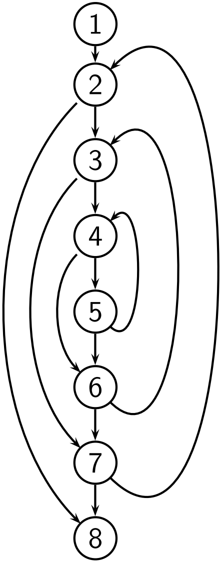
Width and Depth



Structures resulting from repeat-until loops with premature exits

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

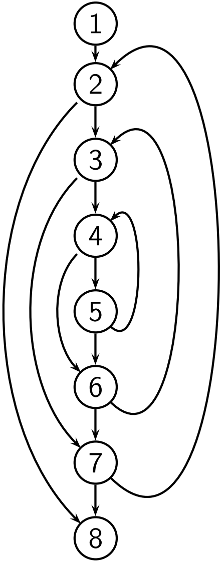
Width and Depth



Structures resulting from repeat-until loops with premature exits

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

Width and Depth



Structures resulting from repeat-until loops with premature exits

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

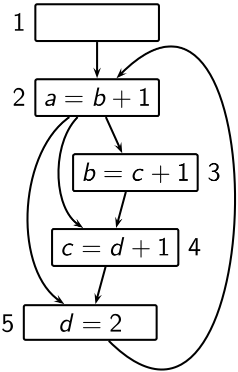
Show the trace of the analysis in the folloing format:

Step No.	Program Point Selected	Remaining Work list	Data Flow Value	Program Point(s) Added	Resulting Work list

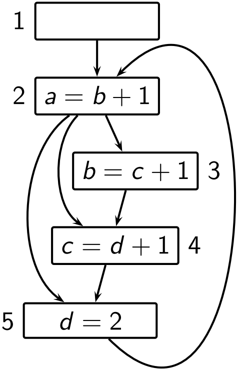
Part 9

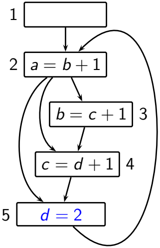
Precise Modelling of General Flows

Complexity of Constant Propagation?



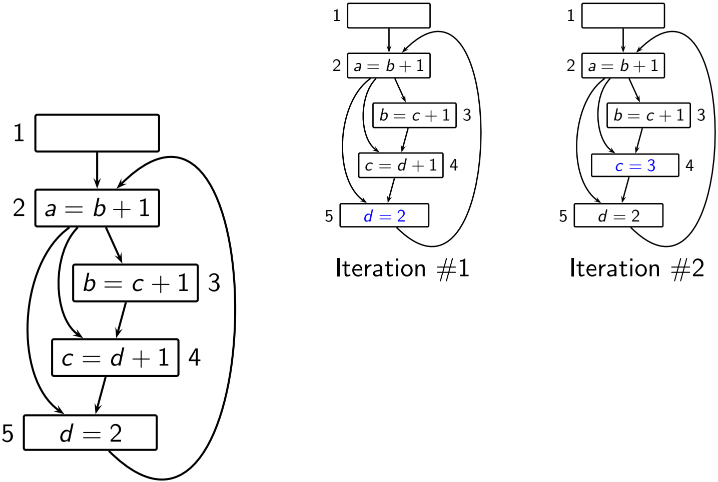
Complexity of Constant Propagation?



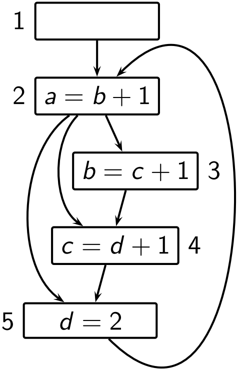


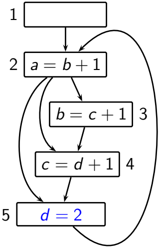
Iteration #1

Complexity of Constant Propagation?

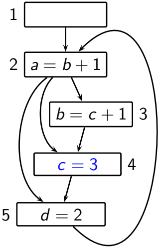


Complexity of Constant Propagation?

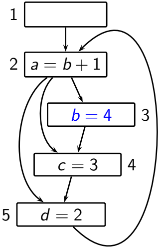




Iteration #1

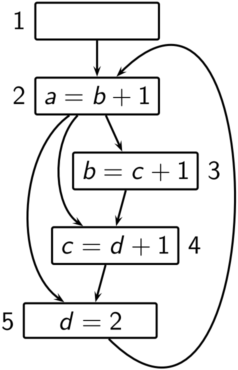


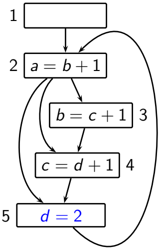
Iteration #2



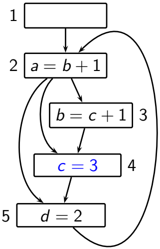
Iteration #3

Complexity of Constant Propagation?

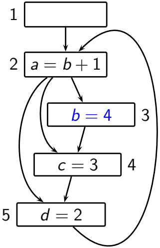




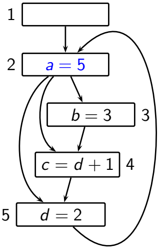
Iteration #1



Iteration #2



Iteration #3



Iteration #4

Larger Values of Loop Closure Bounds

- Fast Frameworks $\equiv$ 2-bounded frameworks (eg. bit vector frameworks)
  Both these conditions must be satisfied
  - *Separability*
    Data flow values of different entities are independent
  - *Constant or Identity Flow Functions*
    Flow functions for an entity are either constant or identity
- Non-fast frameworks
  At least one of the above conditions is violated

Separability

$f : L \mapsto L$ is $\langle \widehat{h}_1, \widehat{h}_2, \ldots, \widehat{h}_m \rangle$ where $\widehat{h}_i$ computes the value of $\widehat{x}_i$

Separability

$f : L \mapsto L$ is $\langle \widehat{h}_1, \widehat{h}_2, \ldots, \widehat{h}_m \rangle$ where $\widehat{h}_i$ computes the value of $\widehat{x}_i$

Separable	**Non-Separable**
Example: All bit vector frameworks	Example: Constant Propagation

Separability

$f : L \mapsto L$ is $\langle \widehat{h}_1, \widehat{h}_2, \ldots, \widehat{h}_m \rangle$ where $\widehat{h}_i$ computes the value of $\widehat{x}_i$

Separable

$$\langle\ \widehat{x}_1,\ \widehat{x}_2,\ \ldots,\ \widehat{x}_m\ \rangle$$
$$\Downarrow$$
$$f$$
$$\Downarrow$$
$$\langle\ \widehat{y}_1,\ \widehat{y}_2,\ \ldots,\ \widehat{y}_m\ \rangle$$

Example: All bit vector frameworks

Non-Separable

$$\langle\ \widehat{x}_1,\ \widehat{x}_2,\ \ldots,\ \widehat{x}_m\ \rangle$$
$$\Downarrow$$
$$f$$
$$\Downarrow$$
$$\langle\ \widehat{y}_1,\ \widehat{y}_2,\ \ldots,\ \widehat{y}_m\ \rangle$$

Example: Constant Propagation

Separability

$f : L \mapsto L$ is $\langle \widehat{h}_1, \widehat{h}_2, \ldots, \widehat{h}_m \rangle$ where $\widehat{h}_i$ computes the value of $\widehat{x}_i$

Separable

$\langle\ \widehat{x}_1,\ \widehat{x}_2,\ \ldots,\ \widehat{x}_m\ \rangle$

$\widehat{h}_2$

$\langle\ \widehat{y}_1,\ \widehat{y}_2,\ \ldots,\ \widehat{y}_m\ \rangle$

Example: All bit vector frameworks

Non-Separable

$\langle\ \widehat{x}_1,\ \widehat{x}_2,\ \ldots,\ \widehat{x}_m\ \rangle$

$\Downarrow$

f

$\Downarrow$

$\langle\ \widehat{y}_1,\ \widehat{y}_2,\ \ldots,\ \widehat{y}_m\ \rangle$

Example: Constant Propagation

Separability

$f : L \mapsto L$ is $\langle \widehat{h}_1, \widehat{h}_2, \ldots, \widehat{h}_m \rangle$ where $\widehat{h}_i$ computes the value of $\widehat{x}_i$

Separable	**Non-Separable**
$\langle\ \widehat{x}_1,\ \widehat{x}_2,\ \ldots,\ \widehat{x}_m\ \rangle$	$\langle\ \widehat{x}_1,\ \widehat{x}_2,\ \ldots,\ \widehat{x}_m\ \rangle$
$\widehat{h}_2$	f
$\langle\ \widehat{y}_1,\ \widehat{y}_2,\ \ldots,\ \widehat{y}_m\ \rangle$	$\langle\ \widehat{y}_1,\ \widehat{y}_2,\ \ldots,\ \widehat{y}_m\ \rangle$
$\widehat{h} : \widehat{L} \mapsto \widehat{L}$	

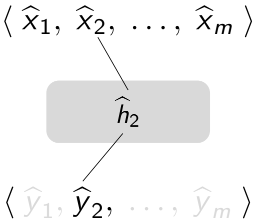
Example: All bit vector frameworks

Example: Constant Propagation

Separability

$f : L \mapsto L$ is $\langle \widehat{h}_1, \widehat{h}_2, \ldots, \widehat{h}_m \rangle$ where $\widehat{h}_i$ computes the value of $\widehat{x}_i$

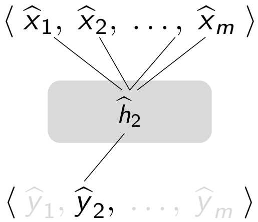
Separable



$\widehat{h} : \widehat{L} \mapsto \widehat{L}$

Example: All bit vector frameworks

Non-Separable



Example: Constant Propagation

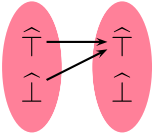
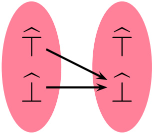
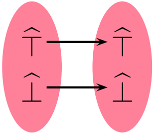
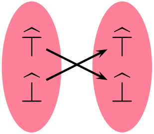
Separability

$f : L \mapsto L$ is $\langle \widehat{h}_1, \widehat{h}_2, \ldots, \widehat{h}_m \rangle$ where $\widehat{h}_i$ computes the value of $\widehat{x}_i$

Separable	**Non-Separable**
$\langle\ \widehat{x}_1,\ \widehat{x}_2,\ \ldots,\ \widehat{x}_m\ \rangle$	$\langle\ \widehat{x}_1,\ \widehat{x}_2,\ \ldots,\ \widehat{x}_m\ \rangle$
$\widehat{h}_2$	$\widehat{h}_2$
$\langle\ \widehat{y}_1,\ \widehat{y}_2,\ \ldots,\ \widehat{y}_m\ \rangle$	$\langle\ \widehat{y}_1,\ \widehat{y}_2,\ \ldots,\ \widehat{y}_m\ \rangle$
$\widehat{h} : \widehat{L} \mapsto \widehat{L}$	$\widehat{h} : L \mapsto \widehat{L}$
Example: All bit vector frameworks	Example: Constant Propagation

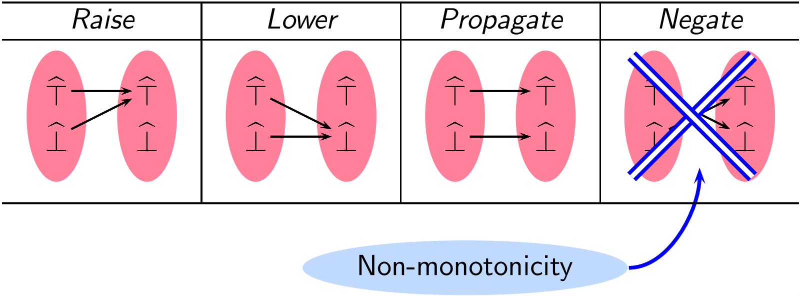
Separability of Bit Vector Frameworks

- $\widehat{L}$ is $\{0,1\}$, L is $\{0,1\}^m$
- $\widehat{\sqcap}$ is either boolean AND or boolean OR
- $\widehat{\top}$ and $\widehat{\bot}$ are 0 or 1 depending on $\widehat{\sqcap}$.
- $\widehat{h}$ is a *bit function* and could be one of the following:

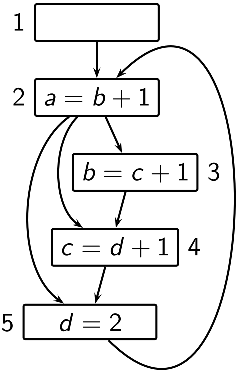
Raise	*Lower*	*Propagate*	*Negate*
			

Separability of Bit Vector Frameworks

- $\widehat{L}$ is $\{0,1\}$, L is $\{0,1\}^m$
- $\widehat{\sqcap}$ is either boolean AND or boolean OR
- $\widehat{\top}$ and $\widehat{\bot}$ are 0 or 1 depending on $\widehat{\sqcap}$.
- $\widehat{h}$ is a *bit function* and could be one of the following:



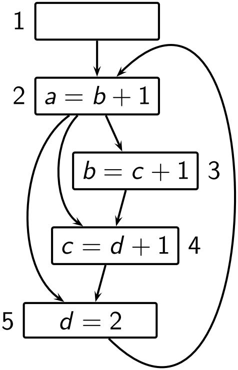
Larger Values of Loop Closure Bounds



Composite flow function for the loop is

$$f(\langle v_a, v_b, v_c, v_d\rangle) = \langle v_b + 1, v_c + 1, v_d + 1, 2\rangle$$

Larger Values of Loop Closure Bounds

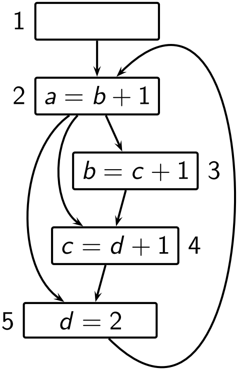


Composite flow function for the loop is

$$f(\langle v_a, v_b, v_c, v_d \rangle) = \langle v_b + 1, v_c + 1, v_d + 1, 2 \rangle$$

f is not 2-bounded because:

Larger Values of Loop Closure Bounds



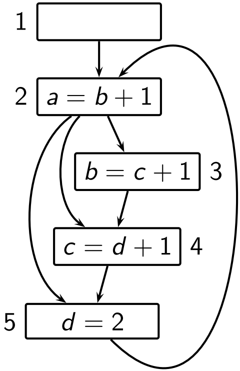
Composite flow function for the loop is

$$f(\langle v_a, v_b, v_c, v_d \rangle) = \langle v_b + 1, v_c + 1, v_d + 1, 2 \rangle$$

f is not 2-bounded because:

$$f(\langle \widehat{\top}, \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle) \quad = \quad \langle \widehat{\top}, \widehat{\top}, \widehat{\top}, 2 \rangle$$

Larger Values of Loop Closure Bounds



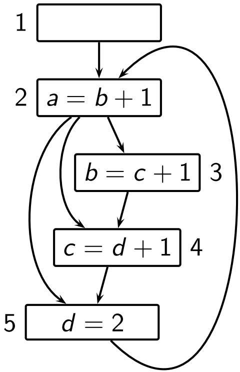
Composite flow function for the loop is

$$f(\langle v_a, v_b, v_c, v_d \rangle) = \langle v_b + 1, v_c + 1, v_d + 1, 2 \rangle$$

f is not 2-bounded because:

$$\begin{array}{rcl} f(\langle \widehat{\top}, \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle) & = & \langle \widehat{\top}, \widehat{\top}, \widehat{\top}, 2 \rangle \\ f^2(\langle \widehat{\top}, \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle) & = & \langle \widehat{\top}, \widehat{\top}, 3, 2 \rangle \end{array}$$

Larger Values of Loop Closure Bounds



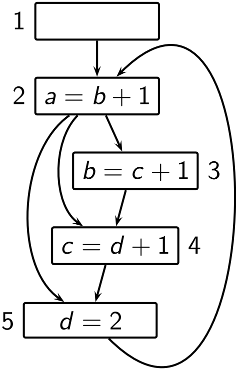
Composite flow function for the loop is

$$f(\langle v_a, v_b, v_c, v_d \rangle) = \langle v_b + 1, v_c + 1, v_d + 1, 2 \rangle$$

f is not 2-bounded because:

$$\begin{array}{rcl} f(\langle \widehat{\top}, \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle) & = & \langle \widehat{\top}, \widehat{\top}, \widehat{\top}, 2 \rangle \\ f^2(\langle \widehat{\top}, \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle) & = & \langle \widehat{\top}, \widehat{\top}, 3, 2 \rangle \\ f^3(\langle \widehat{\top}, \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle) & = & \langle \widehat{\top}, 4, 3, 2 \rangle \end{array}$$

Larger Values of Loop Closure Bounds



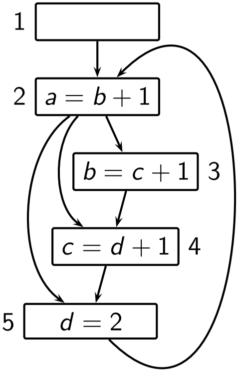
Composite flow function for the loop is

$$f(\langle v_a, v_b, v_c, v_d \rangle) = \langle v_b + 1, v_c + 1, v_d + 1, 2 \rangle$$

f is not 2-bounded because:

$$\begin{array}{rcl}
f(\langle \widehat{\top}, \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle) & = & \langle \widehat{\top}, \widehat{\top}, \widehat{\top}, 2 \rangle \\
f^2(\langle \widehat{\top}, \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle) & = & \langle \widehat{\top}, \widehat{\top}, 3, 2 \rangle \\
f^3(\langle \widehat{\top}, \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle) & = & \langle \widehat{\top}, 4, 3, 2 \rangle \\
f^4(\langle \widehat{\top}, \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle) & = & \langle 5, 4, 3, 2 \rangle
\end{array}$$

Larger Values of Loop Closure Bounds



Composite flow function for the loop is

$$f(\langle v_a, v_b, v_c, v_d\rangle) = \langle v_b + 1, v_c + 1, v_d + 1, 2\rangle$$

f is not 2-bounded because:

$$\begin{aligned}
f(\langle \widehat{\top}, \widehat{\top}, \widehat{\top}, \widehat{\top}\rangle) &= \langle \widehat{\top}, \widehat{\top}, \widehat{\top}, 2\rangle \\
f^2(\langle \widehat{\top}, \widehat{\top}, \widehat{\top}, \widehat{\top}\rangle) &= \langle \widehat{\top}, \widehat{\top}, 3, 2\rangle \\
f^3(\langle \widehat{\top}, \widehat{\top}, \widehat{\top}, \widehat{\top}\rangle) &= \langle \widehat{\top}, 4, 3, 2\rangle \\
f^4(\langle \widehat{\top}, \widehat{\top}, \widehat{\top}, \widehat{\top}\rangle) &= \langle 5, 4, 3, 2\rangle \\
f^5(\langle \widehat{\top}, \widehat{\top}, \widehat{\top}, \widehat{\top}\rangle) &= \langle 5, 4, 3, 2\rangle
\end{aligned}$$

Modelling Flow Functions for General Flows

- General flow functions can be written as

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

where Gen and Kill have constant and dependent parts

$$\begin{aligned} \mathsf{Gen}_n(X) &= ConstGen_n \cup DepGen_n(X) \\ \mathsf{Kill}_n(X) &= ConstKill_n \cup DepKill_n(X) \end{aligned}$$

Modelling Flow Functions for General Flows

- General flow functions can be written as

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

  where Gen and Kill have constant and dependent parts

$$\begin{aligned}\mathrm{Gen}_n(X) &= \mathit{ConstGen}_n \cup \mathit{DepGen}_n(X)\\ \mathrm{Kill}_n(X) &= \mathit{ConstKill}_n \cup \mathit{DepKill}_n(X)\end{aligned}$$

- The dependent parts take care of
  - dependence across different entities as well as
  - dependence on the value of the same entity in the argument X

Modelling Flow Functions for General Flows

- General flow functions can be written as

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

where Gen and Kill have constant and dependent parts

$$\begin{aligned} \text{Gen}_n(X) \quad &= \quad ConstGen_n \cup DepGen_n(X) \\ \text{Kill}_n(X) \quad &= \quad ConstKill_n \cup DepKill_n(X) \end{aligned}$$

- The dependent parts take care of
  - dependence across different entities as well as
  - dependence on the value of the same entity in the argument X
- Bit vector frameworks are a special case

$$DepGen_n(X) = DepKill_n(X) = \emptyset$$

Part 10

Extra Topics

Undecidability of Data Flow Analysis

- 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 $2, 1, 1, 3$.

Modified Post's Correspondence Problem (MPCP)

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

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

Modified Post's Correspondence Problem (MPCP)

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

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

Modified Post's Correspondence Problem (MPCP)

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

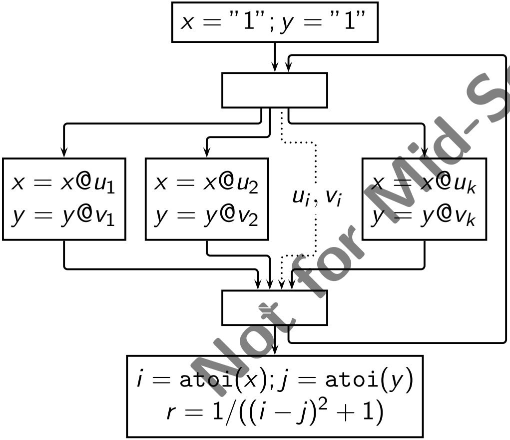
$$u_1 u_{i_1} u_{i_2} \dots u_{i_m} = v_1 v_{i_1} v_{i_2} \dots v_{i_m}$$

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

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

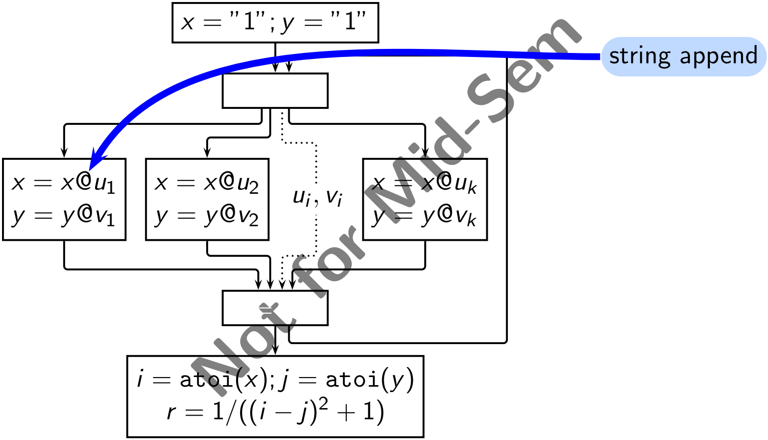
Hecht's MPCP to Constant Propagation Reduction

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



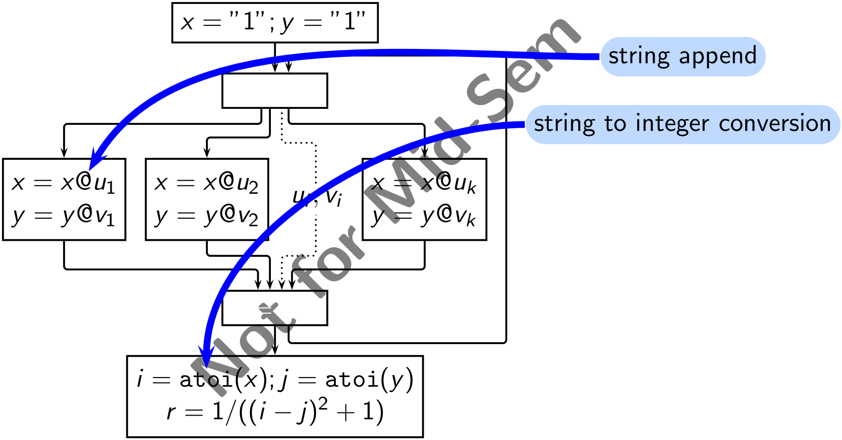
Hecht's MPCP to Constant Propagation Reduction

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



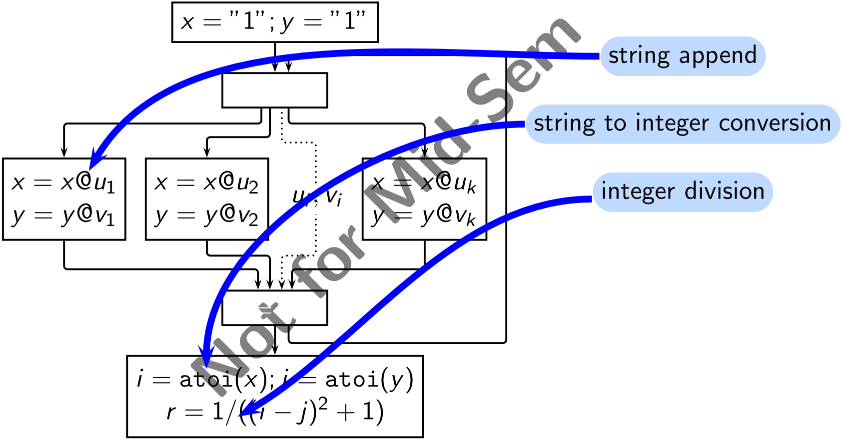
Hecht's MPCP to Constant Propagation Reduction

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



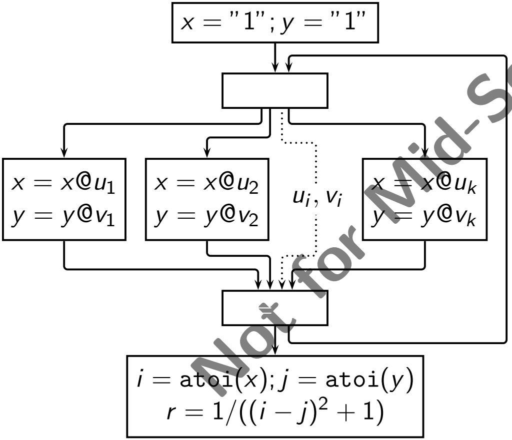
Hecht's MPCP to Constant Propagation Reduction

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



Hecht's MPCP to Constant Propagation Reduction

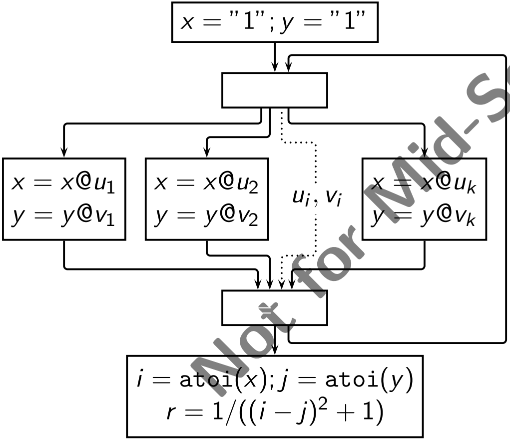
Given: An instance of MPCP with $\Sigma = \{0, 1\}$.



- $i == j \Rightarrow r = 1$
- $i \;!= j \Rightarrow r = 0$

Hecht's MPCP to Constant Propagation Reduction

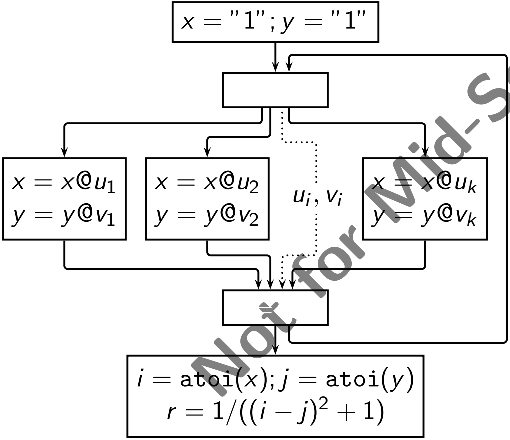
Given: An instance of MPCP with $\Sigma = \{0, 1\}$.



- $i == j \Rightarrow r = 1$
- $i \mathrel{!=} j \Rightarrow r = 0$
- If there exists an algorithm which can determine that

Hecht's MPCP to Constant Propagation Reduction

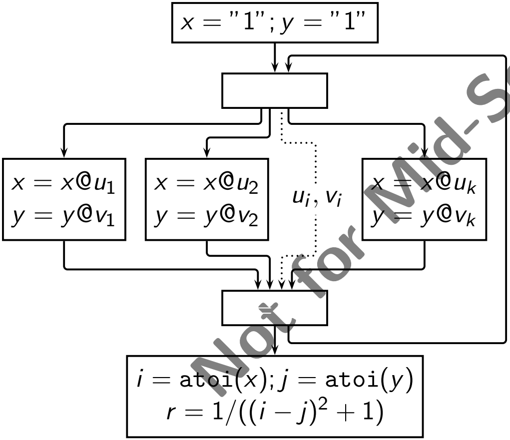
Given: An instance of MPCP with $\Sigma = \{0, 1\}$.



- i == $j \Rightarrow r = 1$
  i != $j \Rightarrow r = 0$
- If there exists an algorithm which can determine that
  - $r = 1$ along some path
    $\Rightarrow x$ == y
    $\Rightarrow$ MPCP instance has a solution

Hecht's MPCP to Constant Propagation Reduction

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

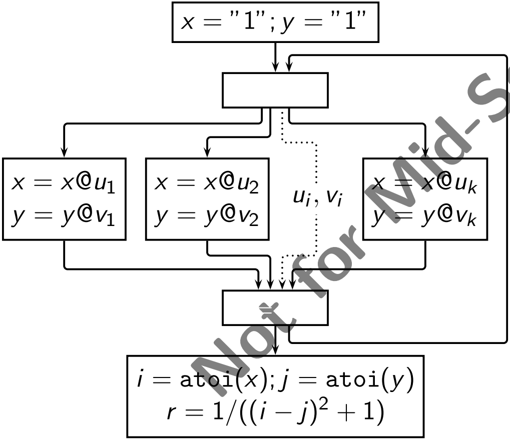


- i `==` $j \Rightarrow r = 1$
  i `!=` $j \Rightarrow r = 0$
- If there exists an algorithm which can determine that
  - $r = 1$ along some path
    $\Rightarrow x$ `==` y
    $\Rightarrow$ MPCP instance has a solution
  - $r = 0$ along every path
    $\Rightarrow x$ `!=` y
    $\Rightarrow$ MPCP instance does not have a solution

$\Rightarrow$ MPCP is decidable

Hecht's MPCP to Constant Propagation Reduction

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



- i == $j \Rightarrow r = 1$
  i != $j \Rightarrow r = 0$
- If there exists an algorithm which can determine that
  - $r = 1$ along some path
    $\Rightarrow x$ == y
    $\Rightarrow$ MPCP instance has a solution
  - $r = 0$ along every path
    $\Rightarrow x$!= y
    $\Rightarrow$ MPCP instance does not have a solution

$\Rightarrow$ MPCP is decidable

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

Tarski's Fixed Point Theorem

Given monotonic $f : L \mapsto L$ 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 $f : L \mapsto L$ where L is a complete lattice

Define

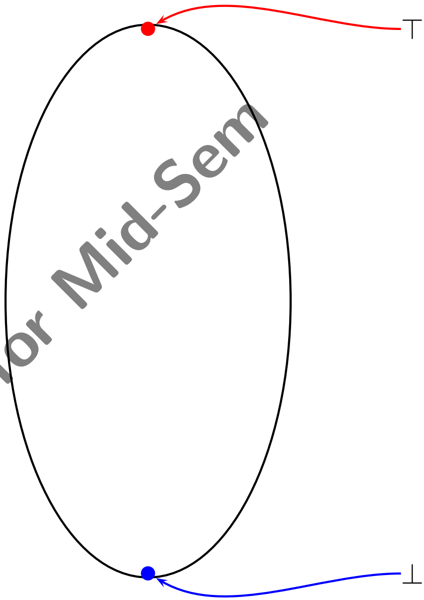
$$
\begin{array}{lrcl}
p \text{ is a fixed point of } f: & Fix(f) & = & \{p \mid f(p) = p\} \\
f \text{ is reductive at } p: & Red(f) & = & \{p \mid f(p) \sqsubseteq p\} \\
f \text{ is extensive at } p: & Ext(f) & = & \{p \mid f(p) \sqsupseteq p\}
\end{array}
$$

Then

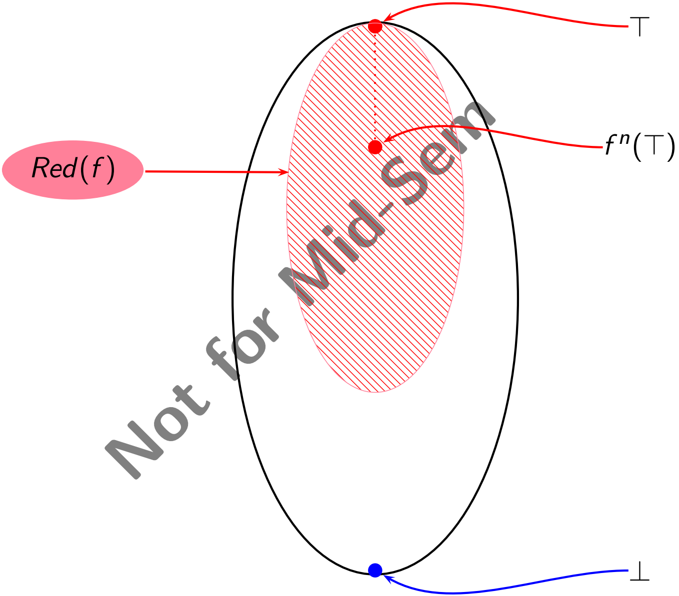
$$
\begin{array}{rcl}
LFP(f) & = & \bigsqcap Red(f) \in Fix(f) \\
MFP(f) & = & \bigsqcup Ext(f) \in Fix(f)
\end{array}
$$

Guarantees only existence, not computability of fixed points.

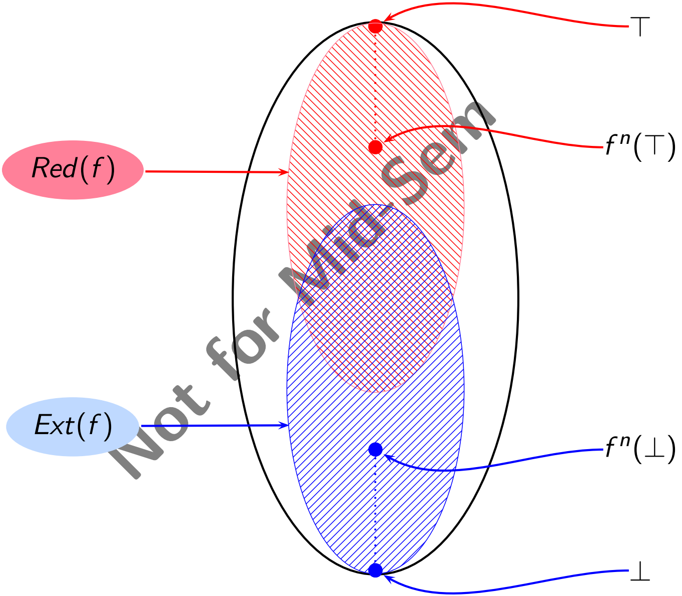
Fixed Points of a Function



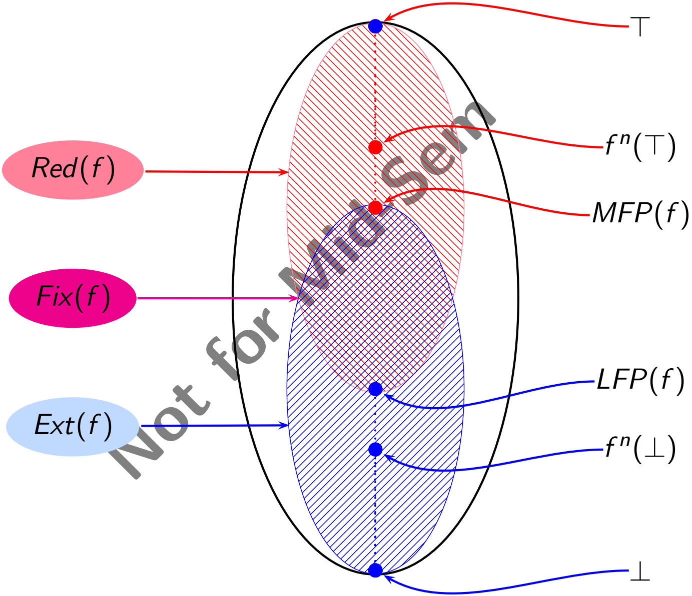
Fixed Points of a Function



Fixed Points of a Function

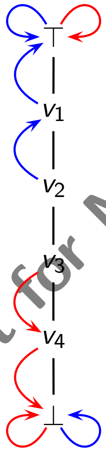


Fixed Points of a Function



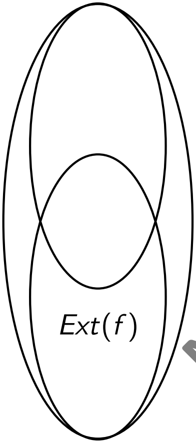
Examples of Reductive and Extensive Sets

Finite L Monotonic $f : L \mapsto L$

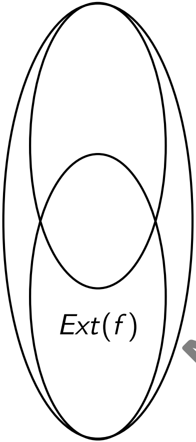


$$
\begin{aligned}
Red(f) &= \{\top, v_3, v_4, \bot\} \\
Ext(f) &= \{\top, v_1, v_2, \bot\} \\
Fix(f) &= Red(f) \cap Ext(f) \\
&= \{\top, \bot\} \\
MFP(f) &= lub\,(Ext(f)) \\
&= lub\,(Fix(f)) \\
&= \top \\
LFP(f) &= glb\,(Red(f)) \\
&= glb\,(Fix(f)) \\
&= \bot
\end{aligned}
$$

Existence of MFP: Proof of Tarski's Fixed Point Theorem

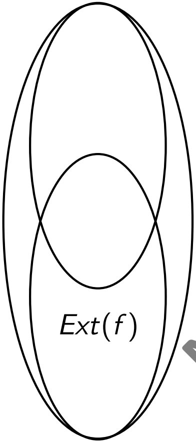


Existence of MFP: Proof of Tarski's Fixed Point Theorem



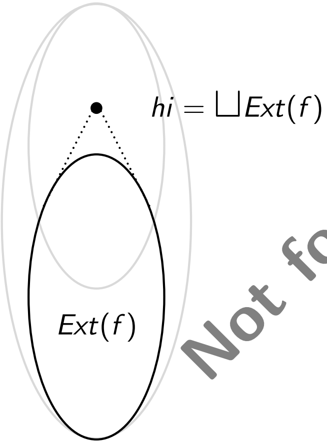
1. Claim 1: Let $X \subseteq L$.
   $\forall x \in X,\ p \sqsupseteq x \Rightarrow p \sqsupseteq \bigsqcup(X)$.

Existence of MFP: Proof of Tarski's Fixed Point Theorem



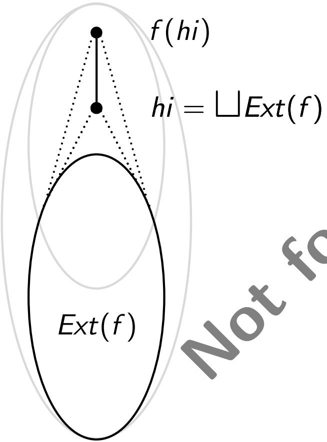
1. Claim 1: Let $X \subseteq L$.
   $\forall x \in X,\ p \sqsupseteq x \Rightarrow p \sqsupseteq \bigsqcup(X)$.
2. In the following we use $Ext(f)$ as X.

Existence of MFP: Proof of Tarski's Fixed Point Theorem



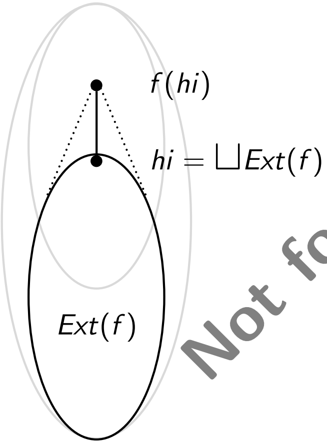
1. Claim 1: Let $X \subseteq L$.
   $\forall x \in X,\ p \sqsupseteq x \Rightarrow p \sqsupseteq \bigsqcup(X)$.
2. In the following we use $Ext(f)$ as X.
3. $\forall p \in Ext(f),\ hi \sqsupseteq p$

Existence of MFP: Proof of Tarski's Fixed Point Theorem



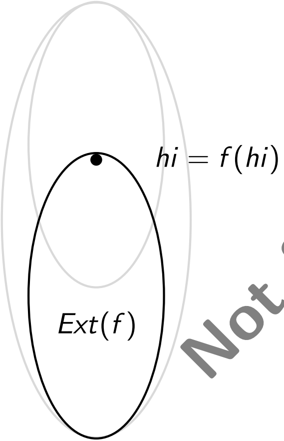
1. Claim 1: Let $X \subseteq L$.
   $\forall x \in X,\ p \sqsupseteq x \Rightarrow p \sqsupseteq \bigsqcup(X)$.
2. In the following we use $Ext(f)$ as X.
3. $\forall p \in Ext(f),\ hi \sqsupseteq p$
4. $hi \sqsupseteq p$
   $\Rightarrow f(hi) \sqsupseteq f(p) \sqsupseteq p$ (monotonicity)
   $\Rightarrow f(hi) \sqsupseteq hi$ (claim 1)

Existence of MFP: Proof of Tarski's Fixed Point Theorem



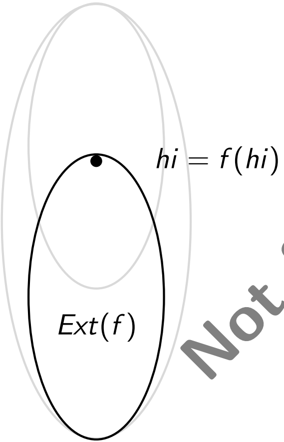
1. Claim 1: Let $X \subseteq L$.
   $\forall x \in X,\ p \sqsupseteq x \Rightarrow p \sqsupseteq \bigsqcup(X)$.
2. In the following we use $Ext(f)$ as X.
3. $\forall p \in Ext(f),\ hi \sqsupseteq p$
4. $hi \sqsupseteq p$
   $\Rightarrow f(hi) \sqsupseteq f(p) \sqsupseteq p$ (monotonicity)
   $\Rightarrow f(hi) \sqsupseteq hi$ (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 $f : L \mapsto L$

Existence and Computation of the Maximum Fixed Point

- For monotonic $f : L \mapsto L$
  - Existence: $MFP(f) = \bigsqcup Ext(f) \in Fix(f)$
    Requires L to be complete.

Existence and Computation of the Maximum Fixed Point

- For monotonic $f : L \mapsto L$
  - 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 $f : L \mapsto L$
  - 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 $f : L \mapsto L$
  - 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 $f : L \mapsto L$
  - 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

k-Bounded Frameworks

Fast Frameworks ($k = 2$)

Rapid Frameworks

Bit Vector Frameworks

$f^2(x) = f(x)$

$f^2(x) \sqsupseteq f(x)$

$f^2(x) \sqsupseteq f(x) \sqcap x$

$f^*(x) = x \sqcap f(x) \sqcap f^2(x) \sqcap \ldots \sqcap f^{k-1}(x)$

Necessary but not sufficient

Necessary and sufficient

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