Theoretical Abstractions in Data Flow Analysis

Uday Khedker

(www.cse.iitb.ac.in/~uday)

Department of Computer Science and Engineering,

Indian Institute of Technology, Bombay

Aug 2024

Copyright

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

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

  (Indian edition published by Ane Books in 2013)

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

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

Outline

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

The Need for a More General Setting

What We Have Seen So Far . . .

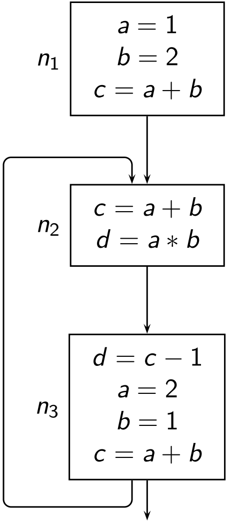
Analysis	Entity	Attribute at p	Paths	
Live variables	Variables	Use	Starting at p	Some
Available expressions	Expressions	Availability	Reaching p	All
Partially available expressions	Expressions	Availability	Reaching p	Some
Anticipable expressions	Expressions	Use	Starting at p	All
Reaching definitions	Definitions	Availability	Reaching p	Some

The Need for a More General Setting

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

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

An Introduction to Constant Propagation (Example from M S Hecht [1977])



An Introduction to Constant Propagation (Example from M S Hecht [1977])

n_1:
$a = 1$
$b = 2$
$c = a + b$

n_2:
$c = a + b$
$d = a * b$

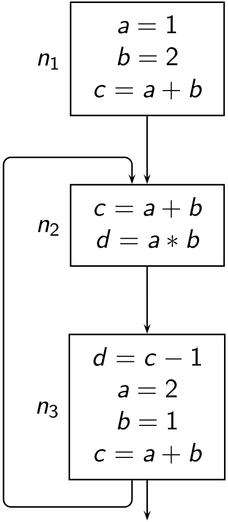
n_3:
$d = c - 1$
$a = 2$
$b = 1$
$c = a + b$

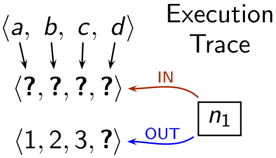
Execution Trace

$\langle a,\ b,\ c,\ d \rangle$

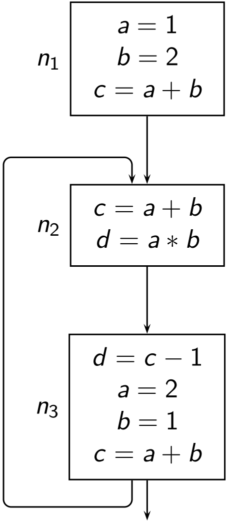
$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle \xleftarrow{\text{IN}} \boxed{n_1}$

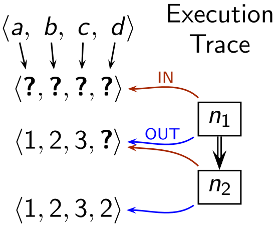
An Introduction to Constant Propagation (Example from M S Hecht [1977])



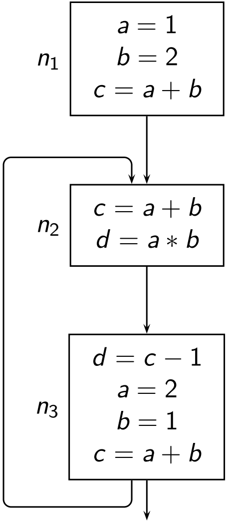


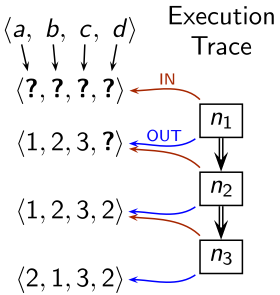
An Introduction to Constant Propagation (Example from M S Hecht [1977])



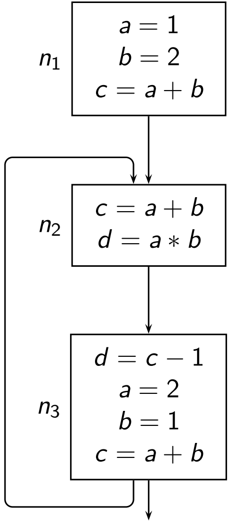


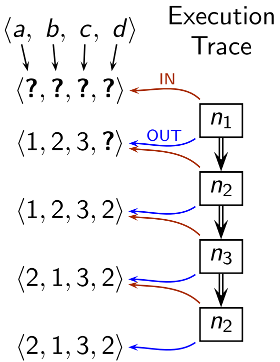
An Introduction to Constant Propagation (Example from M S Hecht [1977])



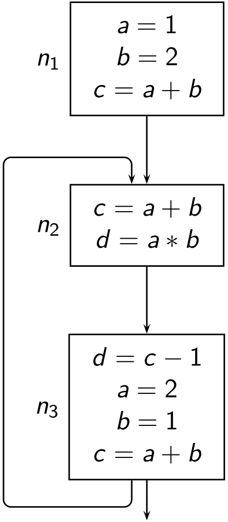


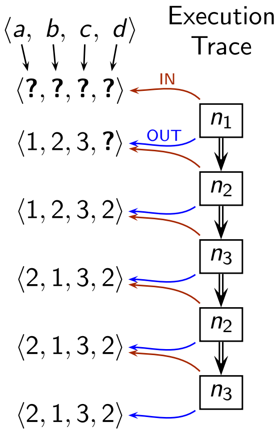
An Introduction to Constant Propagation (Example from M S Hecht [1977])



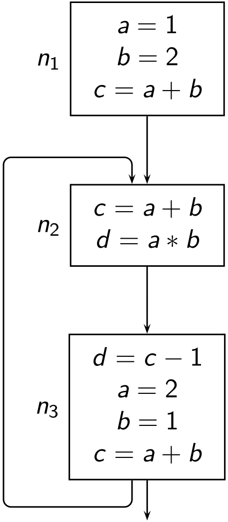


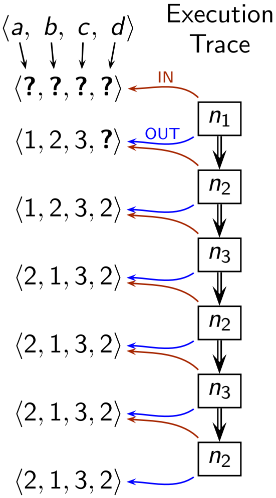
An Introduction to Constant Propagation (Example from M S Hecht [1977])



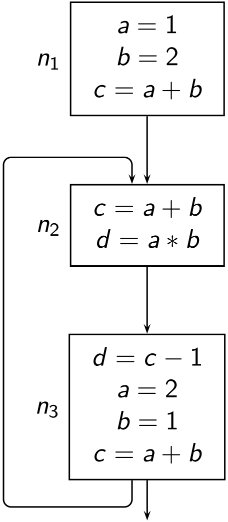


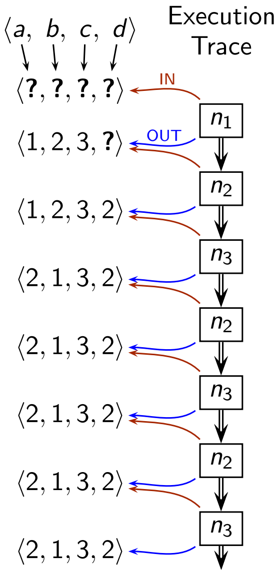
An Introduction to Constant Propagation (Example from M S Hecht [1977])



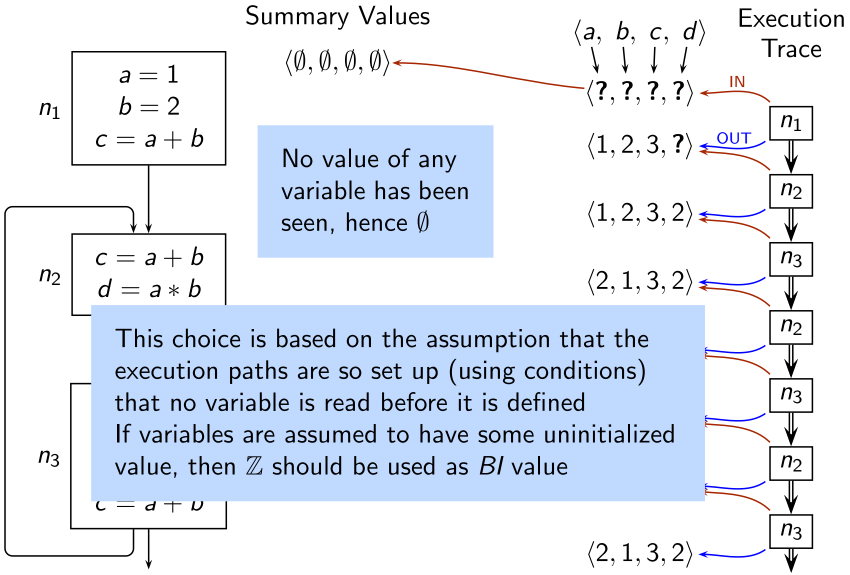


An Introduction to Constant Propagation (Example from M S Hecht [1977])





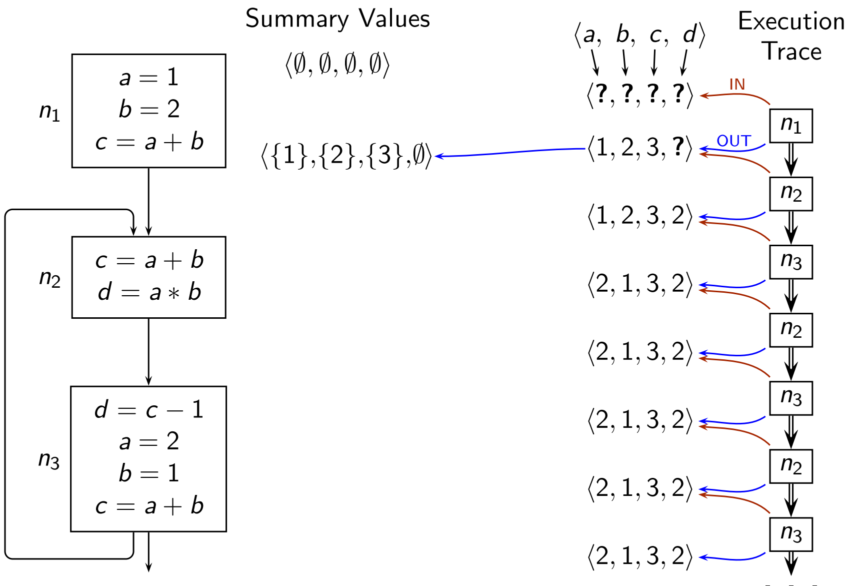
An Introduction to Constant Propagation (Example from M S Hecht [1977])



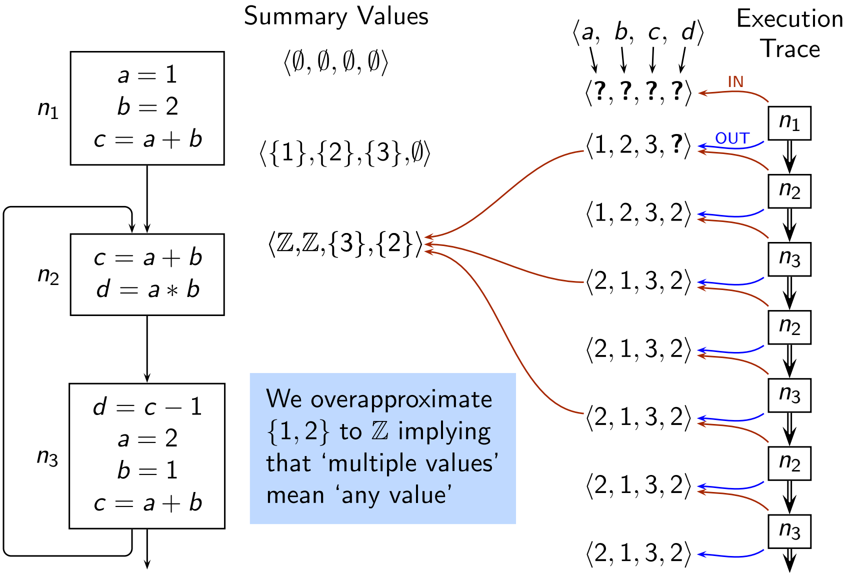
No value of any variable has been seen, hence $\emptyset$

This choice is based on the assumption that the execution paths are so set up (using conditions) that no variable is read before it is defined
If variables are assumed to have some uninitialized value, then $\mathbb{Z}$ should be used as *BI* value

An Introduction to Constant Propagation (Example from M S Hecht [1977])

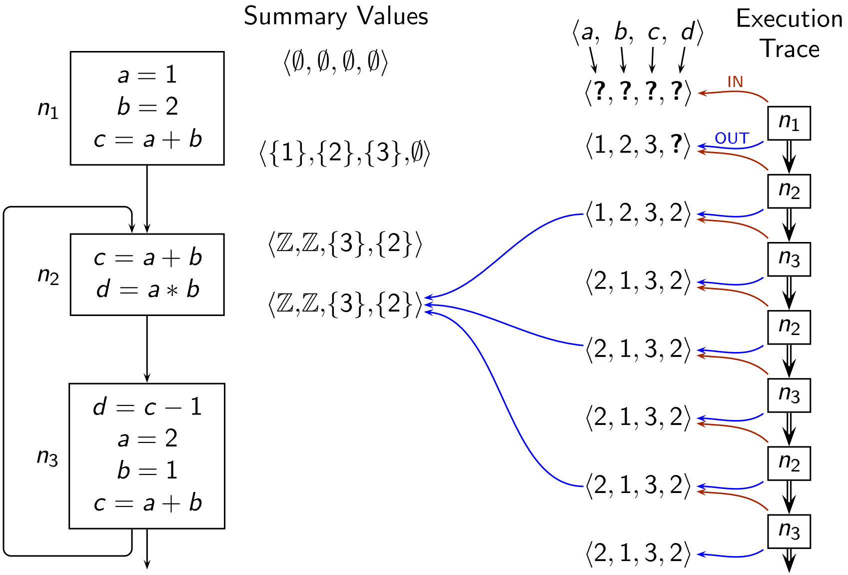


An Introduction to Constant Propagation (Example from M S Hecht [1977])

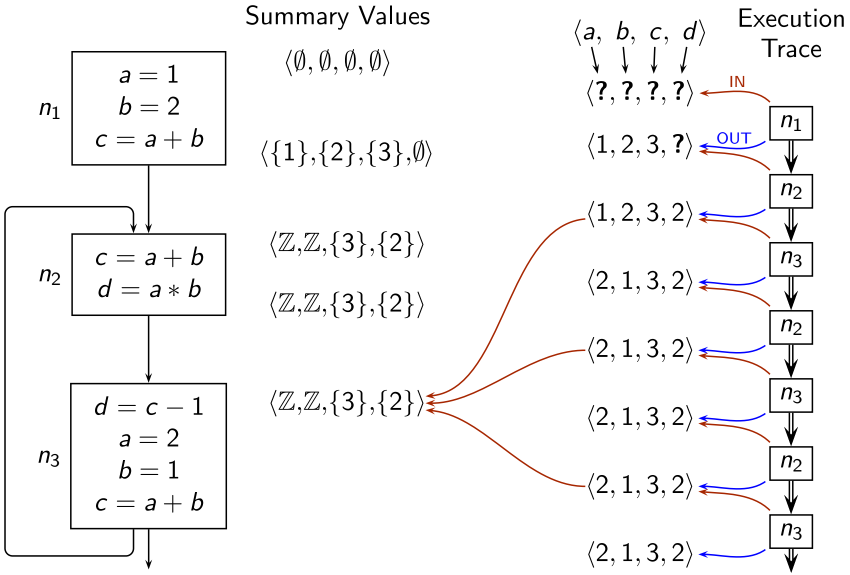


We overapproximate $\{1, 2\}$ to $\mathbb{Z}$ implying that 'multiple values' mean 'any value'

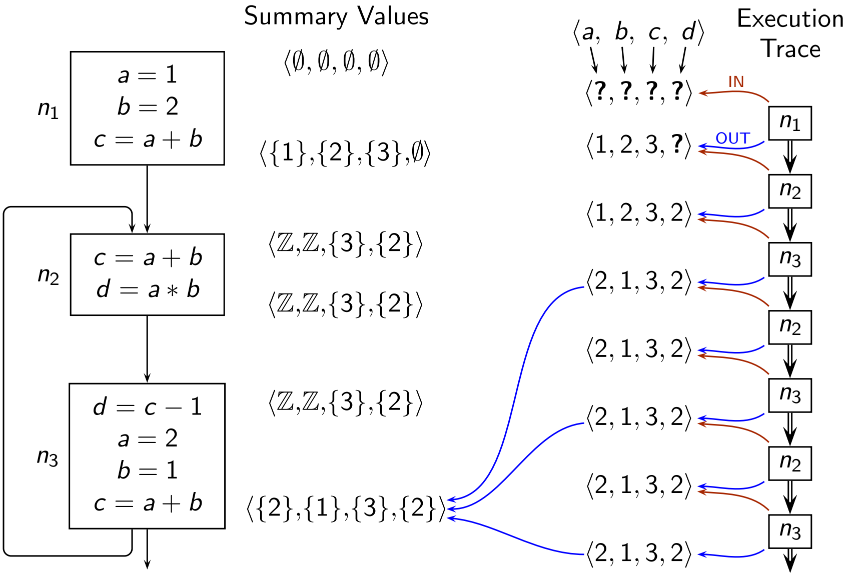
An Introduction to Constant Propagation (Example from M S Hecht [1977])



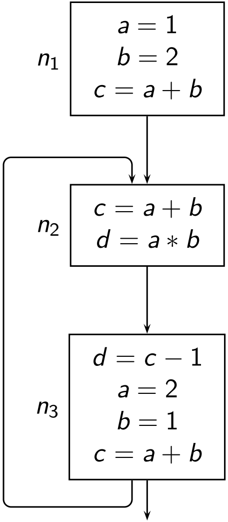
An Introduction to Constant Propagation (Example from M S Hecht [1977])



An Introduction to Constant Propagation (Example from M S Hecht [1977])



An Introduction to Constant Propagation (Example from M S Hecht [1977])



Summary Values

$\langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$

$\langle \{1\},\{2\},\{3\},\emptyset \rangle$

$\langle \mathbb{Z},\mathbb{Z},\{3\},\{2\} \rangle$

$\langle \mathbb{Z},\mathbb{Z},\{3\},\{2\} \rangle$

$\langle \mathbb{Z},\mathbb{Z},\{3\},\{2\} \rangle$

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

Desired Solution

Difference #1: Data Flow Values

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

  Unlike bit vector frameworks, value η_i is not 0 or 1 (i.e. true or false). Instead, it is one of the following:
  - $\emptyset$ indicating that no values is known for v_i
  - $\mathbb{Z}$ indicating that variable v_i could have multiple values
  - Set $\{c_1\}$ if the value of v_i is known to be c_1 at compile time

Difference #2: Dependence of Data Flow Values Across Entities

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

Difference #2: Dependence of Data Flow Values Across Entities

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

Difference #2: Dependence of Data Flow Values Across Entities

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

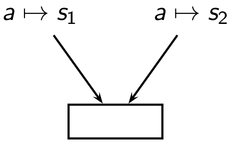
Difference #2: Dependence of Data Flow Values Across Entities

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

No

Difference #3: Confluence Operation

- Confluence operation $a \mapsto s_1 \sqcap a \mapsto s_2$



$\sqcap$	$a \mapsto \emptyset$	$a \mapsto \mathbb{Z}$	$a \mapsto \{c_1\}$
$a \mapsto \emptyset$	$a \mapsto \emptyset$	$a \mapsto \mathbb{Z}$	$a \mapsto \{c_1\}$
$a \mapsto \mathbb{Z}$	$a \mapsto \mathbb{Z}$	$a \mapsto \mathbb{Z}$	$a \mapsto \mathbb{Z}$
$a \mapsto \{c_2\}$	$a \mapsto \{c_2\}$	$a \mapsto \mathbb{Z}$	If $c_1 = c_2$ $a \mapsto \{c_1\}$ Otherwise $a \mapsto \mathbb{Z}$

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

  What are its properties?

Difference #4: Flow Functions for Constant Propagation

- Flow function for $r = a_1 * a_2$

$$a_1 \mapsto s_1 \quad \downarrow \quad a_2 \mapsto s_2$$

$$\boxed{r = a_1 * a_2}$$

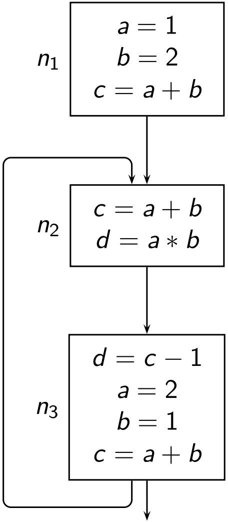
mult	$a_1 \mapsto \emptyset$	$a_1 \mapsto \mathbb{Z}$	$a_1 \mapsto \{c_1\}$
$a_2 \mapsto \emptyset$	$r \mapsto \emptyset$	$r \mapsto \mathbb{Z}$	$r \mapsto \emptyset$
$a_2 \mapsto \mathbb{Z}$	$r \mapsto \mathbb{Z}$	$r \mapsto \mathbb{Z}$	$r \mapsto \mathbb{Z}$
$a_2 \mapsto \{c_2\}$	$r \mapsto \emptyset$	$r \mapsto \mathbb{Z}$	$r \mapsto \{c_1 * c_2\}$

- This cannot be expressed in the form

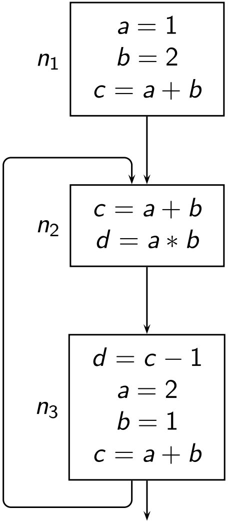
$$f_n(X) = \text{Gen}_n \cup (X - \text{Kill}_n)$$

where Gen_n and Kill_n are constant effects of block n

Difference #5: Solution Computed by the Iterative Method



Difference #5: Solution Computed by the Iterative Method



Iteration #1

$\langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$

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

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

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

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

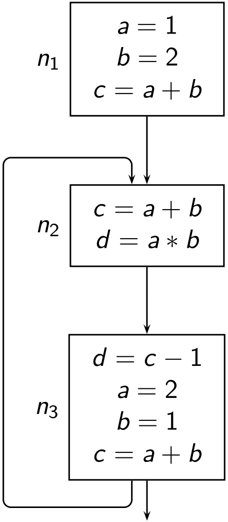
$\langle 2, 1, 3, 2 \rangle$

For convenience, we omit the braces for singleton sets

Difference #5: Solution Computed by the Iterative Method

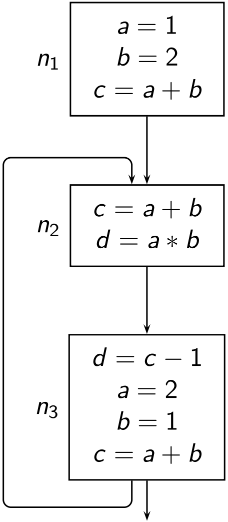
Node	Statements	Iteration #1	Iteration #2
n_1 (in)		$\langle\emptyset,\emptyset,\emptyset,\emptyset\rangle$	$\langle\emptyset,\emptyset,\emptyset,\emptyset\rangle$
n_1	$a = 1$ $b = 2$ $c = a + b$		
n_1 (out)		$\langle 1,2,3,\emptyset\rangle$	$\langle 1,2,3,\emptyset\rangle$
n_2 (in)		$\langle 1,2,3,\emptyset\rangle$	$\langle\mathbb{Z},\mathbb{Z},3,2\rangle$
n_2	$c = a + b$ $d = a * b$		
n_2 (out)		$\langle 1,2,3,2\rangle$	$\langle\mathbb{Z},\mathbb{Z},\mathbb{Z},\mathbb{Z}\rangle$
n_3 (in)		$\langle 1,2,3,2\rangle$	$\langle\mathbb{Z},\mathbb{Z},\mathbb{Z},\mathbb{Z}\rangle$
n_3	$d = c - 1$ $a = 2$ $b = 1$ $c = a + b$		
n_3 (out)		$\langle 2,1,3,2\rangle$	$\langle 2,1,3,\mathbb{Z}\rangle$

Difference #5: Solution Computed by the Iterative Method



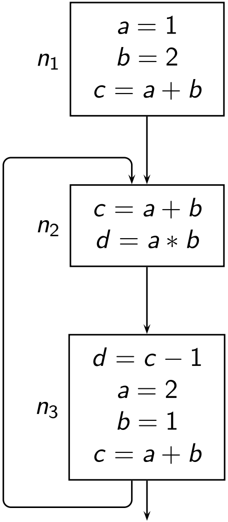
Iteration #1	Iteration #2	Iteration #3
$\langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$	$\langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$	$\langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$
$\langle 1, 2, 3, \emptyset \rangle$	$\langle 1, 2, 3, \emptyset \rangle$	$\langle 1, 2, 3, \emptyset \rangle$
$\langle 1, 2, 3, \emptyset \rangle$	$\langle \mathbb{Z}, \mathbb{Z}, 3, 2 \rangle$	$\langle \mathbb{Z}, \mathbb{Z}, 3, \mathbb{Z} \rangle$
$\langle 1, 2, 3, 2 \rangle$	$\langle \mathbb{Z}, \mathbb{Z}, \mathbb{Z}, \mathbb{Z} \rangle$	$\langle \mathbb{Z}, \mathbb{Z}, \mathbb{Z}, \mathbb{Z} \rangle$
$\langle 1, 2, 3, 2 \rangle$	$\langle \mathbb{Z}, \mathbb{Z}, \mathbb{Z}, \mathbb{Z} \rangle$	$\langle \mathbb{Z}, \mathbb{Z}, \mathbb{Z}, \mathbb{Z} \rangle$
$\langle 2, 1, 3, 2 \rangle$	$\langle 2, 1, 3, \mathbb{Z} \rangle$	$\langle 2, 1, 3, \mathbb{Z} \rangle$

Difference #5: Solution Computed by the Iterative Method



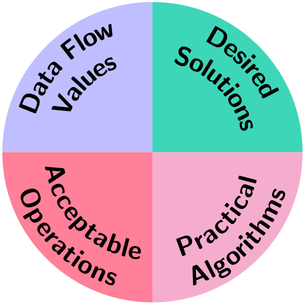
	Iteration #1	Iteration #2	Iteration #3	Desired solution
	$\langle\emptyset,\emptyset,\emptyset,\emptyset\rangle$	$\langle\emptyset,\emptyset,\emptyset,\emptyset\rangle$	$\langle\emptyset,\emptyset,\emptyset,\emptyset\rangle$	$\langle\emptyset,\emptyset,\emptyset,\emptyset\rangle$
	$\langle 1,2,3,\emptyset\rangle$	$\langle 1,2,3,\emptyset\rangle$	$\langle 1,2,3,\emptyset\rangle$	$\langle 1,2,3,\emptyset\rangle$
	$\langle 1,2,3,\emptyset\rangle$	$\langle \mathbb{Z},\mathbb{Z},3,2\rangle$	$\langle \mathbb{Z},\mathbb{Z},3,\mathbb{Z}\rangle$	$\langle \mathbb{Z},\mathbb{Z},3,2\rangle$
	$\langle 1,2,3,2\rangle$	$\langle \mathbb{Z},\mathbb{Z},\mathbb{Z},\mathbb{Z}\rangle$	$\langle \mathbb{Z},\mathbb{Z},\mathbb{Z},\mathbb{Z}\rangle$	$\langle \mathbb{Z},\mathbb{Z},3,2\rangle$
	$\langle 1,2,3,2\rangle$	$\langle \mathbb{Z},\mathbb{Z},\mathbb{Z},\mathbb{Z}\rangle$	$\langle \mathbb{Z},\mathbb{Z},\mathbb{Z},\mathbb{Z}\rangle$	$\langle \mathbb{Z},\mathbb{Z},3,2\rangle$
	$\langle 2,1,3,2\rangle$	$\langle 2,1,3,\mathbb{Z}\rangle$	$\langle 2,1,3,\mathbb{Z}\rangle$	$\langle 2,1,3,2\rangle$

Difference #5: Solution Computed by the Iterative Method



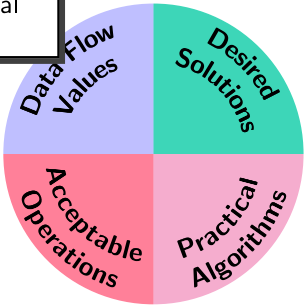
Iteration #1	Iteration #2	Iteration #3	Desired solution
$\langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$	$\langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$	$\langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$	$\langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$
$\langle 1, 2, 3, \emptyset \rangle$	$\langle 1, 2, 3, \emptyset \rangle$	$\langle 1, 2, 3, \emptyset \rangle$	$\langle 1, 2, 3, \emptyset \rangle$
$\langle 1, 2, 3, \emptyset \rangle$	$\langle \mathbb{Z}, \mathbb{Z}, 3, 2 \rangle$	$\langle \mathbb{Z}, \mathbb{Z}, 3, \mathbb{Z} \rangle$	$\langle \mathbb{Z}, \mathbb{Z}, 3, 2 \rangle$
$\langle 1, 2, 3, 2 \rangle$	$\langle \mathbb{Z}, \mathbb{Z}, \mathbb{Z}, \mathbb{Z} \rangle$	$\langle \mathbb{Z}, \mathbb{Z}, \mathbb{Z}, \mathbb{Z} \rangle$	$\langle \mathbb{Z}, \mathbb{Z}, 3, 2 \rangle$
$\langle 1, 2, 3, 2 \rangle$	$\langle \mathbb{Z}, \mathbb{Z}, \mathbb{Z}, \mathbb{Z} \rangle$	$\langle \mathbb{Z}, \mathbb{Z}, \mathbb{Z}, \mathbb{Z} \rangle$	$\langle \mathbb{Z}, \mathbb{Z}, 3, 2 \rangle$
$\langle 2, 1, 3, 2 \rangle$	$\langle 2, 1, 3, \mathbb{Z} \rangle$	$\langle 2, 1, 3, \mathbb{Z} \rangle$	$\langle 2, 1, 3, 2 \rangle$

Issues in Data Flow Analysis



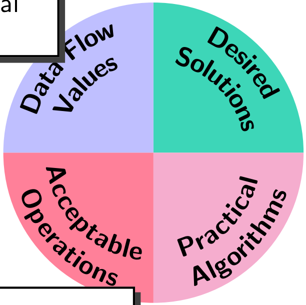
Issues in Data Flow Analysis

- Semantics
- Representation
- Approximation: Partial Order, Lattices



Issues in Data Flow Analysis

- Semantics
- Representation
- Approximation: Partial Order, Lattices

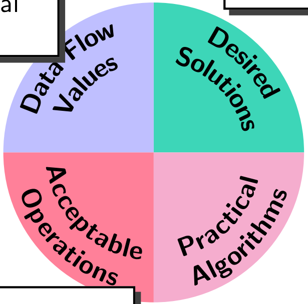


- Merge: Commutativity, Associativity, Idempotence
- Flow Functions: Monotonicity, Distributivity, Boundedness, Separability

Issues in Data Flow Analysis

- Semantics
- Representation
- Approximation: Partial Order, Lattices

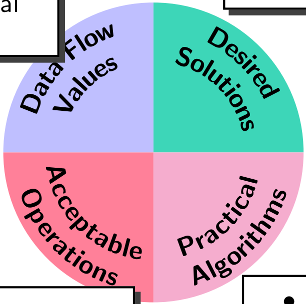
- Existence, Computability
- Soundness, Precision



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

Issues in Data Flow Analysis

- Semantics
- Representation
- Approximation: Partial Order, Lattices



- Existence, Computability
- Soundness, Precision

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

- Complexity, efficiency
- Convergence
- Initialization

Data Flow Values: An Overview

Data Flow Values: An Outline of Our Discussion

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

A Digression on Lattices

Partially Ordered Sets

Sets in which elements can be compared and ordered

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

Partially Ordered Sets

Sets in which elements can be compared and ordered

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

Example of partial order

Pre-requisite relation between courses

Order of taking exams

Order of answering questions in an exam

Total order

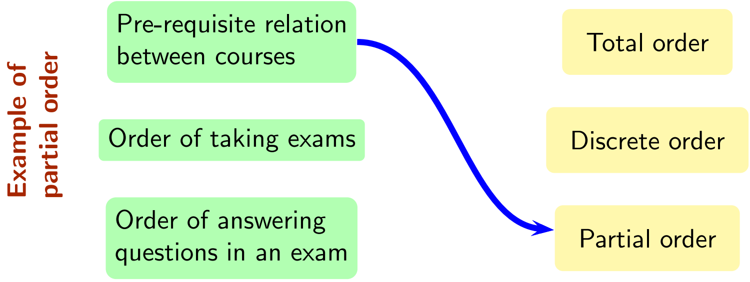
Discrete order

Partial order

Partially Ordered Sets

Sets in which elements can be compared and ordered

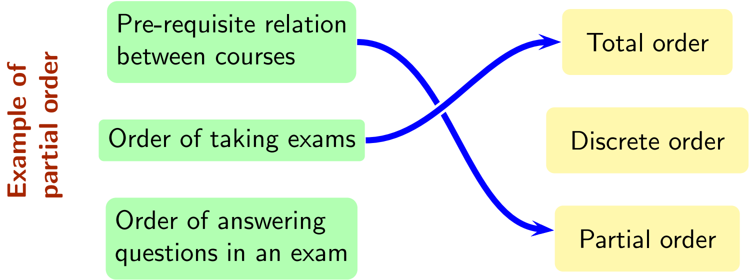
- *Total order*. Every element is comparable with every element (including itself)
- *Discrete order*. Every element is comparable only with itself and not with any other element
- *Partial order*. An element is comparable with itself and some other elements but not necessarily with all elements



Partially Ordered Sets

Sets in which elements can be compared and ordered

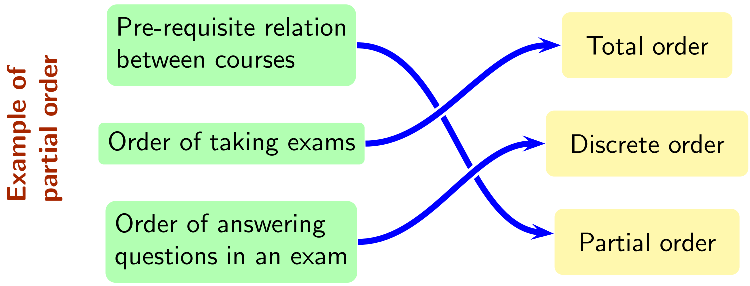
- *Total order*. Every element is comparable with every element (including itself)
- *Discrete order*. Every element is comparable only with itself and not with any other element
- *Partial order*. An element is comparable with itself and some other elements but not necessarily with all elements



Partially Ordered Sets

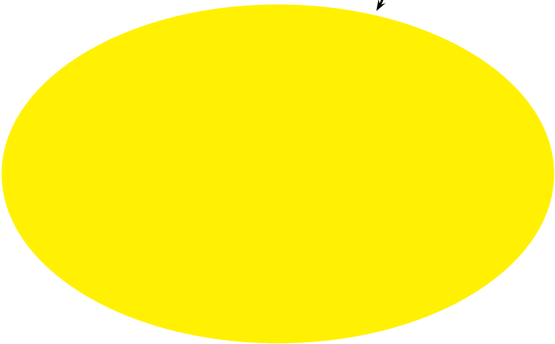
Sets in which elements can be compared and ordered

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



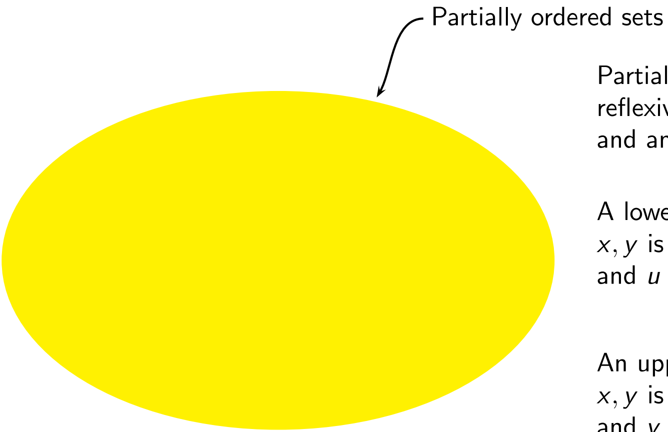
Partially Ordered Sets and Lattices

Partially ordered sets



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

Partially Ordered Sets and Lattices

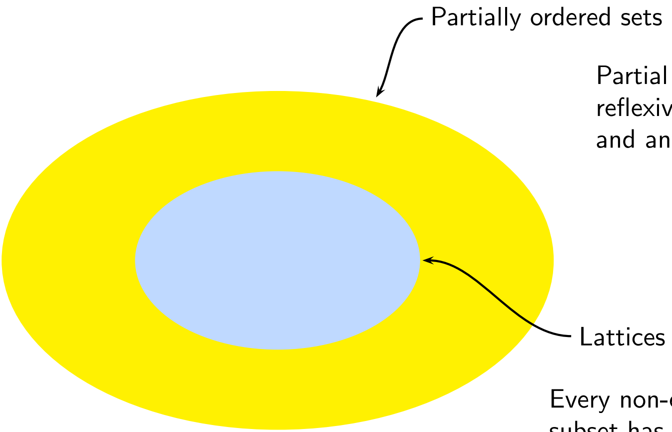


Partial order $\sqsubseteq$ is reflexive, transitive, and antisymmetric

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

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

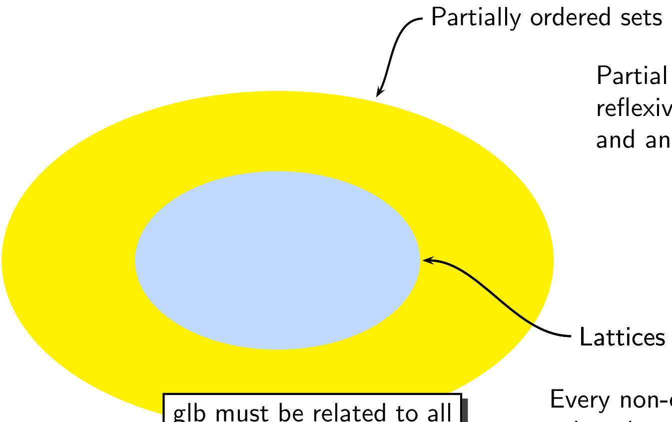
Partially Ordered Sets and Lattices



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

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

Partially Ordered Sets and Lattices



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

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

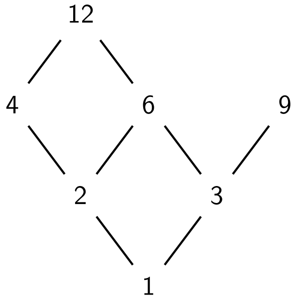
Every non-empty finite subset has a greatest lower bound (glb) and a least upper bound (lub)

Partially Ordered Sets

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

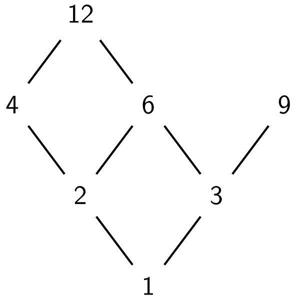
Partially Ordered Sets

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



Partially Ordered Sets

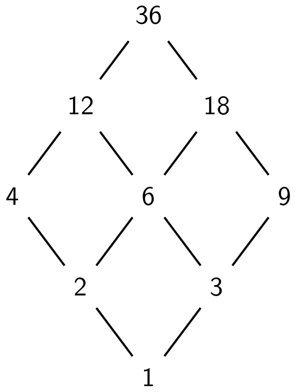
Set $\{1, 2, 3, 4, 6, 9, 12\}$ with $\sqsubseteq$ relation as "divides" (i.e. $a \sqsubseteq b$ iff a divides b)



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

Lattice

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



Examples of Orderings on Strings

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

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

    Example: Arrangement of words in a dictionary
    ball $\preceq$ bat $\preceq$ bath
    - "ball" and "bat" have a common prefix "ba" and l $\preceq$ t
    - "bat" is a proper prefix of "bath"

Complete Lattice

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

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

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

Complete Lattice

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

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

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

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

Complete Lattice

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

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

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

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

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

Complete Lattice

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

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

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

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

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

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

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

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

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

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

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

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

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

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

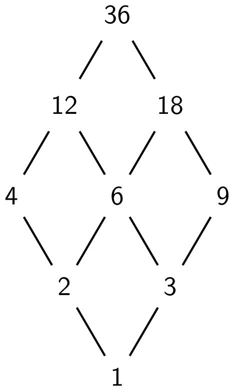
    Every element of $\mathbb{Z} \cup \{\infty, -\infty\}$ is vacuously a lower bound of every element in $\emptyset$ (because there is no element in $\emptyset$)

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

    (Every element of $\mathbb{Z} \cup \{\infty, -\infty\}$ is vacuously an upper bound too, of every element in $\emptyset$)

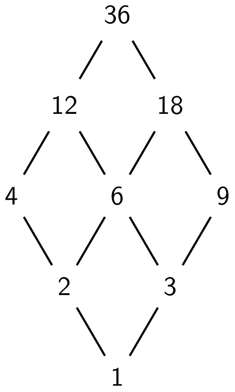
Operations on Lattices

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



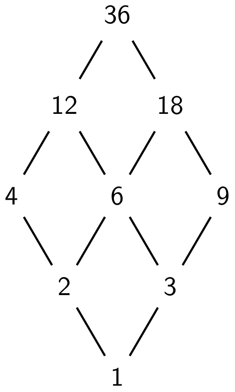
Operations on Lattices

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



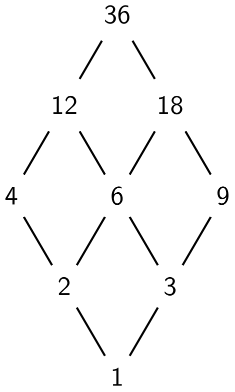
Operations on Lattices

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



Operations on Lattices

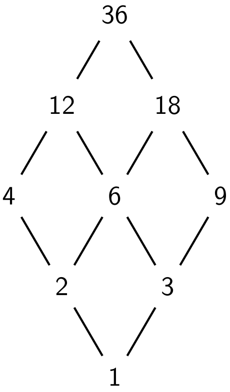
- Meet ($\sqcap$) and Join ($\sqcup$)
  - $x \sqcap y$ computes the glb of x and y
    $z = x \sqcap y \Rightarrow z \sqsubseteq x \wedge z \sqsubseteq y$
  - $x \sqcup y$ computes the lub of x and y
    $z = x \sqcup y \Rightarrow z \sqsupseteq x \wedge z \sqsupseteq y$
  - $\sqcap$ and $\sqcup$ are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



Operations on Lattices

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

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

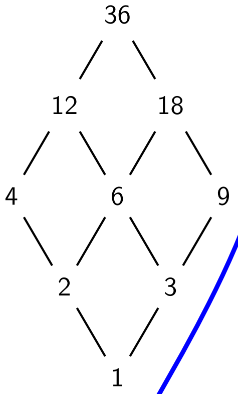


Operations on Lattices

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

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

Greatest common divisor



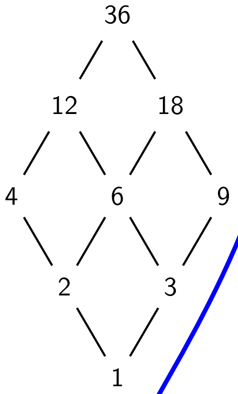
$x \sqcap y = gcd(x, y)$

Operations on Lattices

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

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

Greatest common divisor



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

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

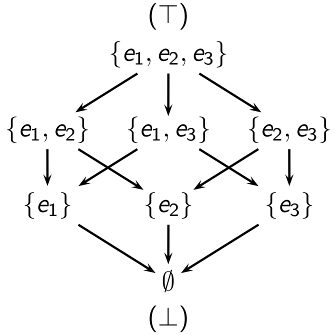
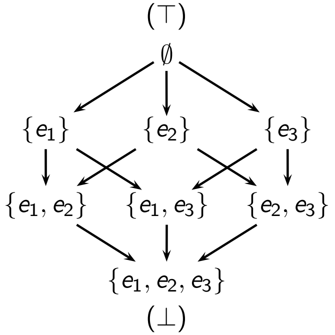
Lowest common multiple

Partial Order and Operations

- For a lattice, $\sqsubseteq$ relation induces $\sqcap$ and $\sqcup$ opperations, and vice-versa
- The choices of $\sqsubseteq$, $\sqcap$, and $\sqcup$ cannot be arbitrary
  They have to be
  - consistent with each other, and
  - definable in terms of each other
- For some variants of lattices, $\sqcap$ or $\sqcup$ may not exist
  Yet the requirement of its consistency with $\sqsubseteq$ cannot be violated

Finite Lattices are Complete

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

Available Expressions Analysis	Partially Available Expressions Analysis
	

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

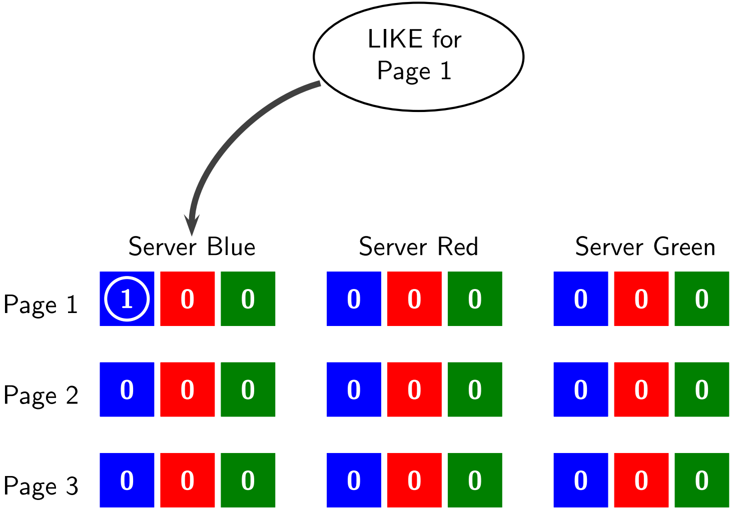
- If L satisfies acc and dcc, then
  - L has finite height, and
  - L is complete
- A complete lattice need not have finite height
  (i.e. strict chains may not be finite)
  Example:
  Lattice of integers under $\leq$ relation with ∞ as $\top$ and $-\infty$ as $\bot$

An Example of Lattices: Maintaining LIKE Counts on Cloud

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

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

An Example of Lattices: Maintaining LIKE Counts on Cloud

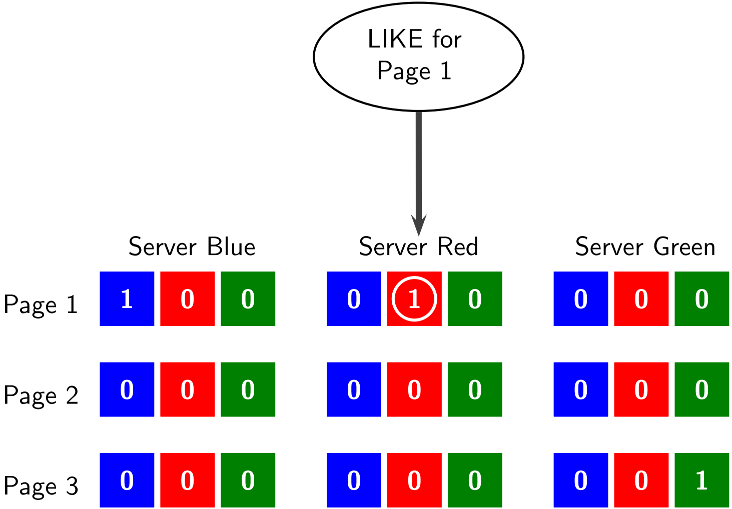


An Example of Lattices: Maintaining LIKE Counts on Cloud

LIKE for Page 3

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

An Example of Lattices: Maintaining LIKE Counts on Cloud

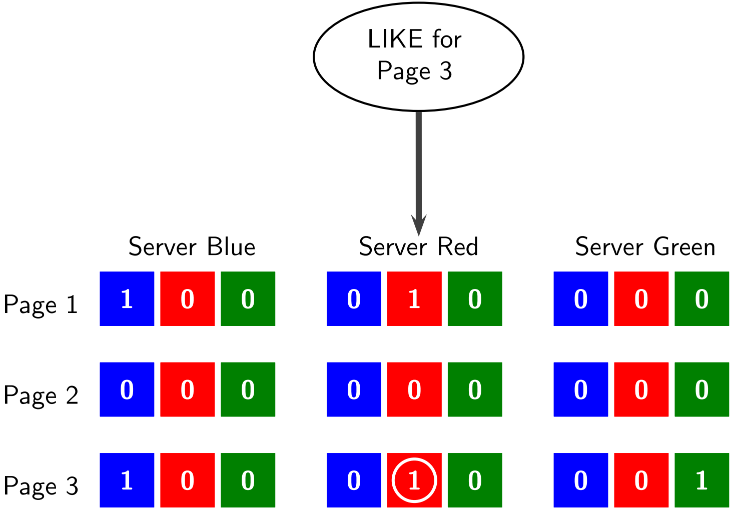


An Example of Lattices: Maintaining LIKE Counts on Cloud

LIKE for Page 3

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

An Example of Lattices: Maintaining LIKE Counts on Cloud



An Example of Lattices: Maintaining LIKE Counts on Cloud

LIKE for Page 3

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

An Example of Lattices: Maintaining LIKE Counts on Cloud

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

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

An Example of Lattices: Maintaining LIKE Counts on Cloud

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

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

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

- Tuples merged with max operation

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

An Example of Lattices: Maintaining LIKE Counts on Cloud

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

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

An Example of Lattices: Maintaining LIKE Counts on Cloud

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

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

An Example of Lattices: Maintaining LIKE Counts on Cloud

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

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

An Example of Lattices: Maintaining LIKE Counts on Cloud

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

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

An Example of Lattices: Maintaining LIKE Counts on Cloud

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

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

An Example of Lattices: Maintaining LIKE Counts on Cloud

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

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

An Example of Lattices: Maintaining LIKE Counts on Cloud

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

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

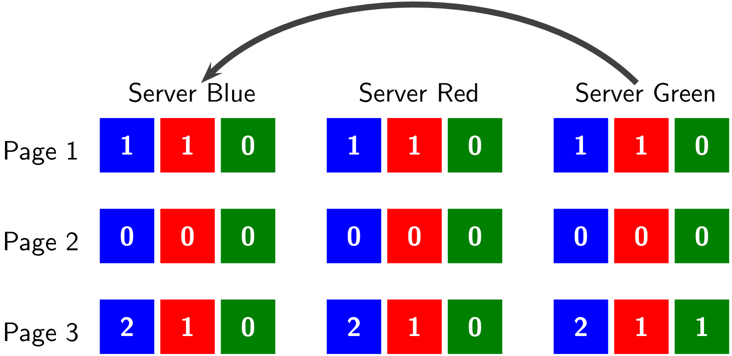
An Example of Lattices: Maintaining LIKE Counts on Cloud

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

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

An Example of Lattices: Maintaining LIKE Counts on Cloud

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



An Example of Lattices: Maintaining LIKE Counts on Cloud

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

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

An Example of Lattices: Maintaining LIKE Counts on Cloud

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

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

An Example of Lattices: Maintaining LIKE Counts on Cloud

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

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

An Example of Lattices: Maintaining LIKE Counts on Cloud

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

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

Constructing Lattices

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

Variants of Powerset Lattices

Consider set S

- The set 2^S with the partial order $\supseteq$ is a lattice with $\top = \emptyset$ and $\bot = S$

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

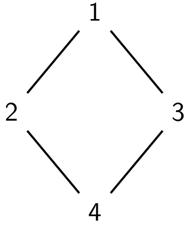
  We used such a lattice in live variables analysis, reaching definitions analysis, and partially available expressions analysis

- The set 2^S with the partial order $\subseteq$ is a lattice with $\top = S$ and $\bot = \emptyset$

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

  We used such a lattice in available expressions analysis and anticipable expressions analysis

Cartesian Product of Lattices



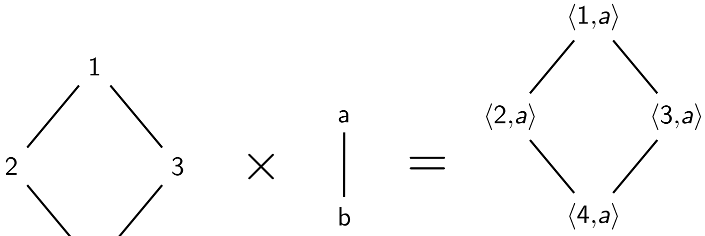
$\langle L_N, \sqsubseteq_N, \sqcap_N, \sqcup_N \rangle$

$\times$

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

$=$

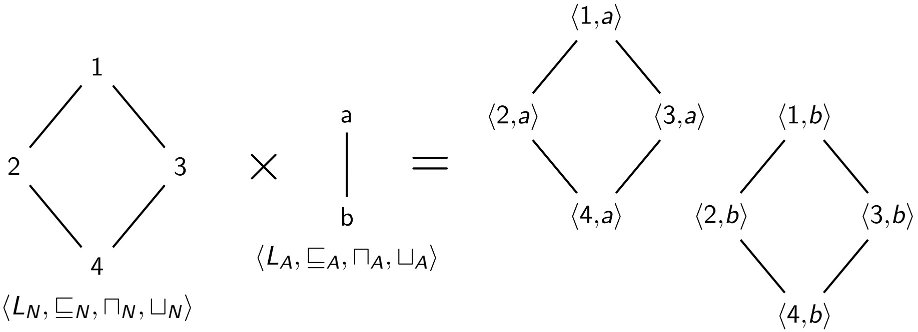
Cartesian Product of Lattices



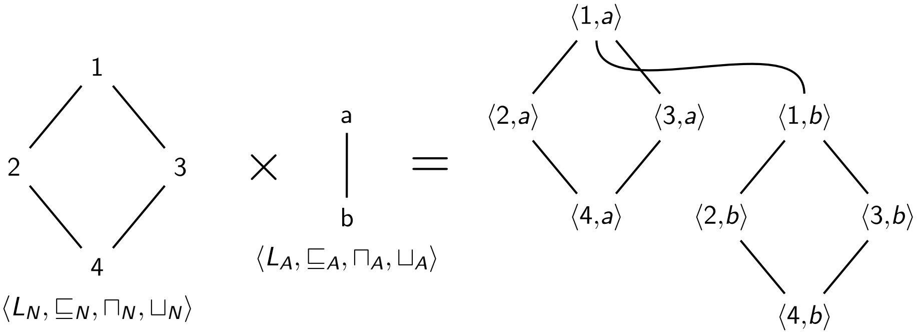
$\langle L_N, \sqsubseteq_N, \sqcap_N, \sqcup_N \rangle$

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

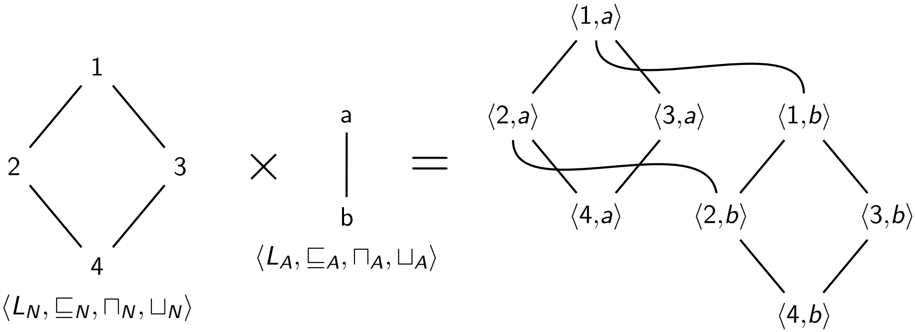
Cartesian Product of Lattices



Cartesian Product of Lattices

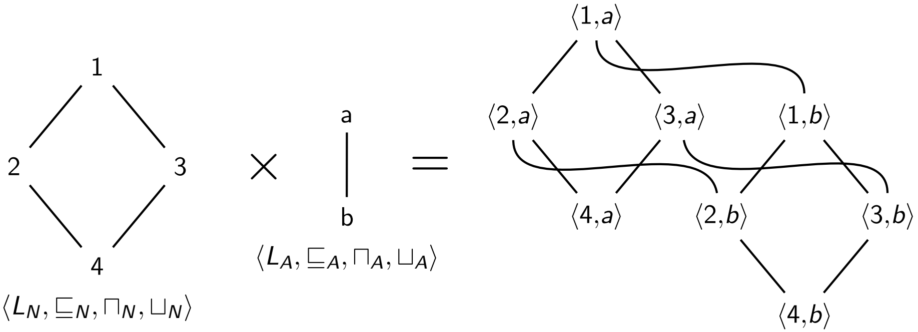


Cartesian Product of Lattices

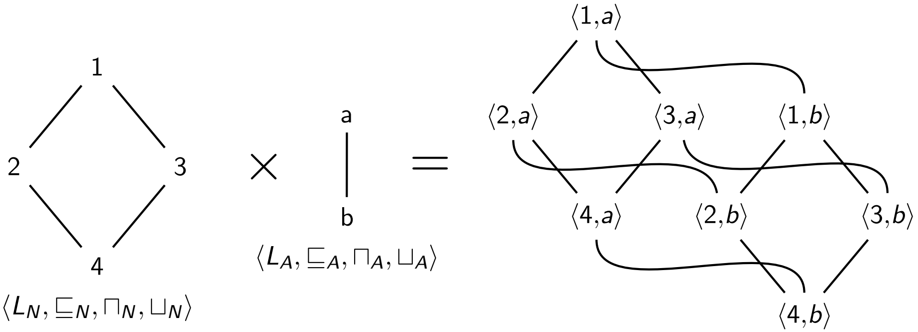


$\langle L_N, \sqsubseteq_N, \sqcap_N, \sqcup_N \rangle$ $\times$ $\langle L_A, \sqsubseteq_A, \sqcap_A, \sqcup_A \rangle$ $=$

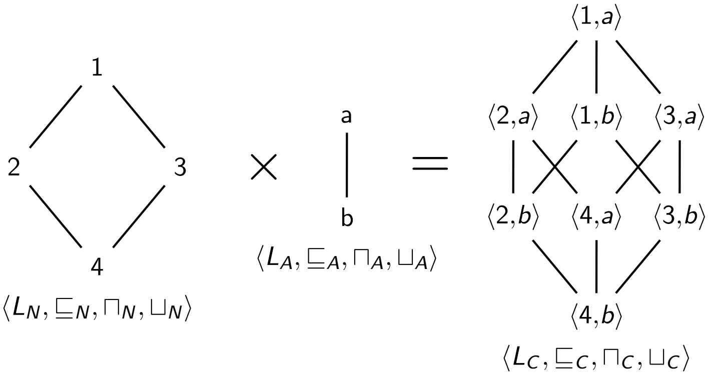
Cartesian Product of Lattices



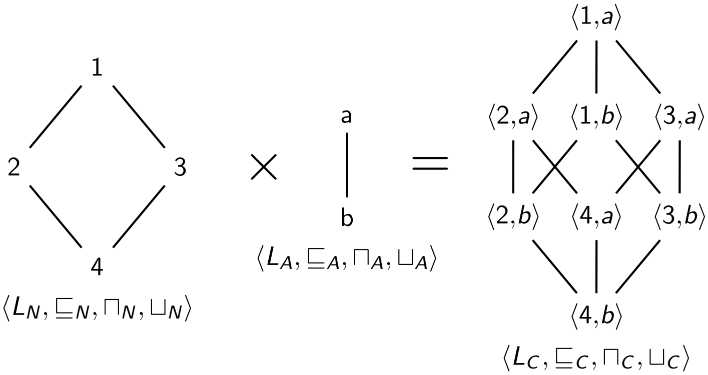
Cartesian Product of Lattices



Cartesian Product of Lattices



Cartesian Product of Lattices



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

Variants of Product Lattices

$L \subseteq L_x \times L_y, \quad \{(x_1, y_1), (x_2, y_2)\} \subseteq L, \quad \{x_1, x_2\} \subseteq L_x$, and $\{y_1, y_2\} \subseteq L_y$

- *Cartesian Product*

$$\begin{aligned}(x_1, y_1) \sqsubseteq (x_2, y_2) &\Leftrightarrow x_1 \sqsubseteq_x x_2 \wedge y_1 \sqsubseteq_y y_2 \\ (x_1, y_1) \sqcap (x_2, y_2) &= x_1 \sqcap_x x_2 \wedge y_1 \sqcap_y y_2 \\ (x_1, y_1) \sqcup (x_2, y_2) &= x_1 \sqcup_x x_2 \wedge y_1 \sqcup_y y_2\end{aligned}$$

- *Interval Product*

Variants of Product Lattices

$L \subseteq L_x \times L_y, \quad \{(x_1, y_1), (x_2, y_2)\} \subseteq L, \quad \{x_1, x_2\} \subseteq L_x$, and $\{y_1, y_2\} \subseteq L_y$

- *Cartesian Product*

$$\begin{aligned}(x_1, y_1) \sqsubseteq (x_2, y_2) &\Leftrightarrow x_1 \sqsubseteq_x x_2 \wedge y_1 \sqsubseteq_y y_2 \\ (x_1, y_1) \sqcap (x_2, y_2) &= x_1 \sqcap_x x_2 \wedge y_1 \sqcap_y y_2 \\ (x_1, y_1) \sqcup (x_2, y_2) &= x_1 \sqcup_x x_2 \wedge y_1 \sqcup_y y_2\end{aligned}$$

- *Interval Product*

$$\begin{aligned}(x_1, y_1) \sqsubseteq (x_2, y_2) &\Leftrightarrow x_1 \sqsubseteq_x x_2 \wedge y_1 \sqsupseteq_y y_2 \\ (x_1, y_1) \sqcap (x_2, y_2) &= x_1 \sqcap_x x_2 \wedge y_1 \sqcup_y y_2 \\ (x_1, y_1) \sqcup (x_2, y_2) &= x_1 \sqcup_x x_2 \wedge y_1 \sqcap_y y_2\end{aligned}$$

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

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

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

Set of Mappings as a Lattice

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

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

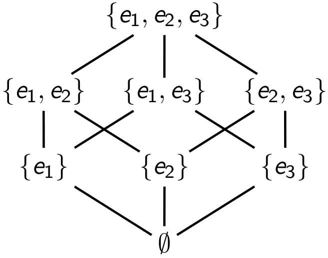
$$
\begin{aligned}
X \sqsubseteq Y \;&\Leftrightarrow\; \forall a \in A.\, (a, x) \in X \wedge (a, y) \in Y \wedge x \sqsubseteq_1 y \\
X \sqcap Y \;&=\; \big\{(a, x \sqcap_1 y) \mid a \in A, (a, x) \in X, (a, y) \in Y\big\} \\
X \sqcup Y \;&=\; \big\{(a, x \sqcup_1 y) \mid a \in A, (a, x) \in X, (a,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

Note: $(a, x) \in X$ is same as $a \mapsto x \in X$ when X is a function

Data Flow Values: Details

The Set of Data Flow Values For Available Expressions Analysis

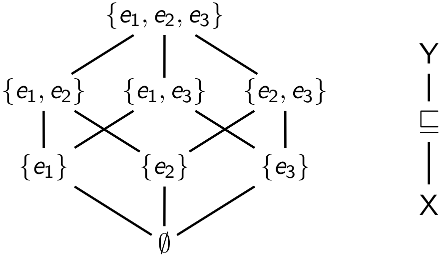
- The powerset of the universal set of expressions
- Partial order is the subset relation



Set View of the Lattice

The Set of Data Flow Values For Available Expressions Analysis

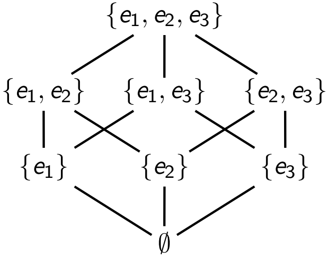
- The powerset of the universal set of expressions
- Partial order is the subset relation



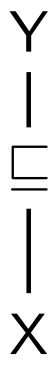
Set View of the Lattice

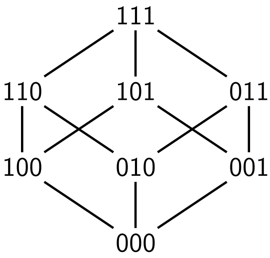
The Set of Data Flow Values For Available Expressions Analysis

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



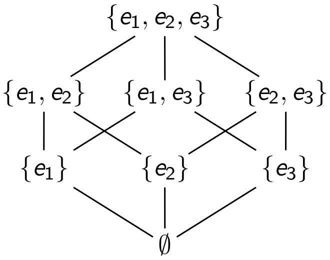
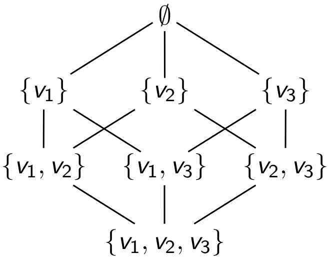
Set View of the Lattice



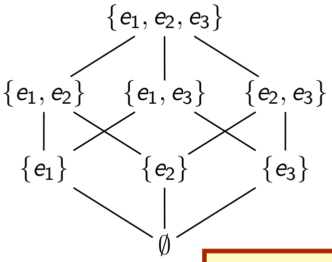
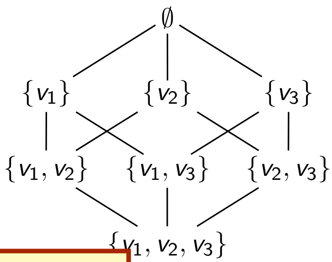


Bit Vector View

Setting Up Lattices

Available Expressions Analysis	Live Variables Analysis
	
$\sqsubseteq$ is $\subseteq$	$\sqsubseteq$ is $\supseteq$
$\sqcap$ is $\cap$	$\sqcap$ is $\cup$

Setting Up Lattices

Available Expressions Analysis	Live Variables Analysis
	
$\sqsubseteq$ is $\subseteq$	$\sqsubseteq$ is $\supseteq$
$\sqcap$ is $\cap$	$\sqcap$ is $\cup$

We choose an orientation such that the result of the confluence (i.e. $\sqcap$) appears "below" the two operands

Mappings View of Lattices in Bit Vector Frameworks

Bit vector frameworks have $L = A \rightarrow \{1, 0\}$

- Live variables analysis. $L = \mathbb{V}\text{ar} \rightarrow \{1, 0\}$

  Given $\mathbb{V}\text{ar} = \{a, b, c\}$, set $\{a, c\} = \{a \mapsto 1, b \mapsto 0, c \mapsto 1\}$ or 101 in bit vector notation

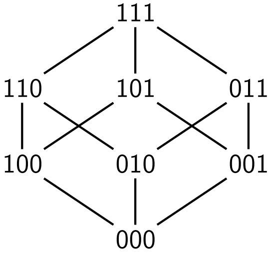
- Available variables analysis. $L = \mathbb{E}\text{xpr} \rightarrow \{1, 0\}$

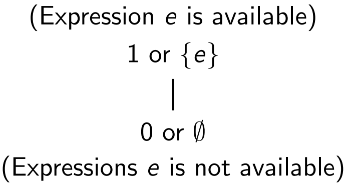
  (also partially available expressions or anticipable expressions analysis)

- Reaching definitions analysis. $L = \mathbb{V}\text{ar} \times \mathbb{N} \rightarrow \{1, 0\}$ where $\mathbb{N}$ is the set of nodes in the program

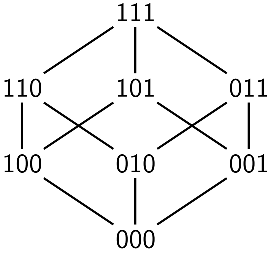
  Given $\mathbb{V}\text{ar} = \{a, b, c\}$, $\mathbb{N} = \{1, 2\}$, the set of definitions $\{a_1, c_2\}$ is a mapping $\{(a, 1) \mapsto 1, (a, 2) \mapsto 0, (b, 1) \mapsto 0, (b, 2) \mapsto 0, (c, 1) \mapsto 0, (c, 2) \mapsto 1\}$

Product View of Lattices in Bit Vector Frameworks

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




Product View of Lattices in Bit Vector Frameworks

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

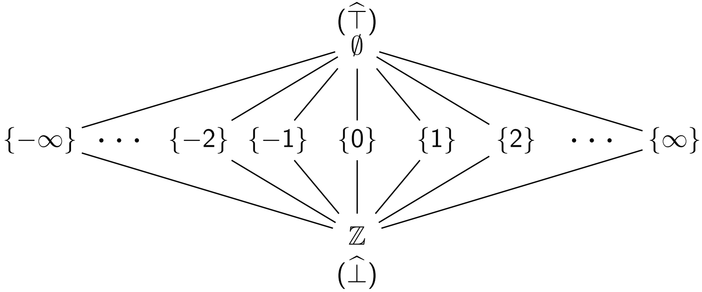
Cartesian products if sets are used, vectors (or tuples) if bit are used

- $L = \widehat{L} \times \widehat{L} \times \widehat{L}$ 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
$\vert$	$\vert$
0	1
$(\widehat{\bot})$	$(\widehat{\bot})$
$\sqcap$ is $\cap$ or Boolean AND	$\sqcap$ is $\cup$ or Boolean OR

Component Lattice for Integer Constant Propagation



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

$\widehat{\sqcap}$	$a \mapsto \emptyset$	$a \mapsto \mathbb{Z}$	$a \mapsto \{c_1\}$
$a \mapsto \emptyset$	$a \mapsto \emptyset$	$a \mapsto \mathbb{Z}$	$a \mapsto \{c_1\}$
$a \mapsto \mathbb{Z}$	$a \mapsto \mathbb{Z}$	$a \mapsto \mathbb{Z}$	$a \mapsto \mathbb{Z}$
$a \mapsto \{c_2\}$	$a \mapsto \{c_2\}$	$a \mapsto \mathbb{Z}$	$c_1 = c_2 ? a \mapsto \{c_1\} : a \mapsto \mathbb{Z}$

Two Views of Lattice in Constant Propagation

For constant propagation there are two views

- Set of values view $L = \mathbb{V}\mathrm{ar} \to 2^{\mathbb{Z}}$

  Given $\mathbb{V}\mathrm{ar} = \{a, b, c\}$, examples of data flow values are
  - $\{a \mapsto \{3\}, b \mapsto \emptyset, c \mapsto \{5\}\}$
  - $\{a \mapsto \mathbb{Z}, b \mapsto \{15\}, c \mapsto \mathbb{Z}\}$
- Single value view $L = \mathbb{V}\mathrm{ar} \to \mathbb{Z} \cup \{\widehat{\top}, \widehat{\bot}\}$

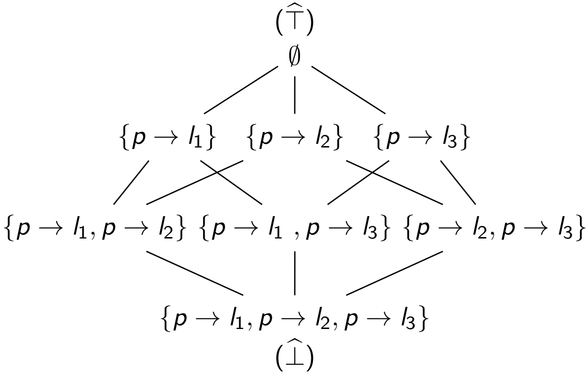
  The data flow values in the above example are represented by
  - $\{a \mapsto 3, b \mapsto \widehat{\top}, c \mapsto 5\}$
  - $\{a \mapsto \widehat{\bot}, b \mapsto 15, c \mapsto \widehat{\bot}\}$

Component Lattice for May Points-To Analysis

- Relation between pointer variables and locations in the memory

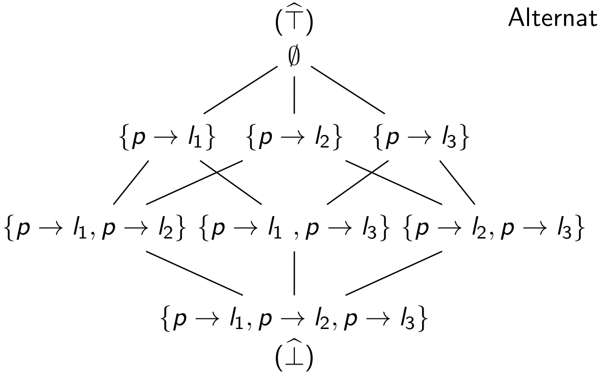
Component Lattice for May Points-To Analysis

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

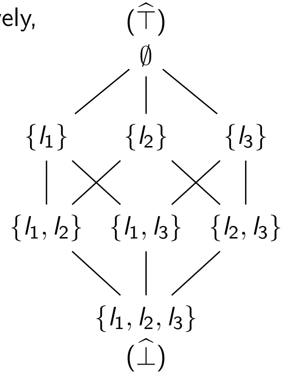


Component Lattice for May Points-To Analysis

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

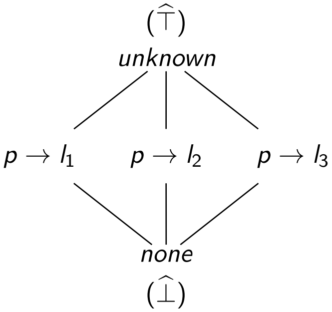


Alternatively,

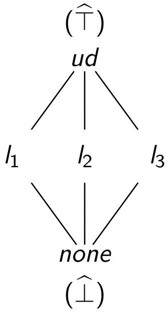


Component Lattice for Must Points-To Analysis

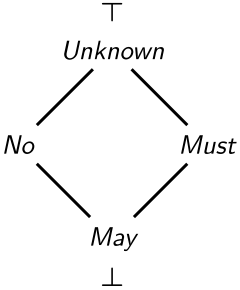
- A pointer can point to at most one location



Alternatively,



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. $(p \mapsto l, \mathit{May})$, $(p \mapsto l, \mathit{Must})$, $(p \mapsto l, \mathit{No})$, or $(p \mapsto l, \mathit{Unknown})$
- Type Inferencing for Dynamically Checked Languages

What Does A Lattice Represent?

- The concept of a lattice seems to fit needs of data flow analysis
- What semantics does it actually represent?
  How does it relate to soundness and precision?

The Concept of Conservative Approximation

- x approximates y conservatively *iff*

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

- Validity of approximation is context specific

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

The Concept of Conservative Approximation

- x approximates y conservatively *iff*

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

- Validity of approximation is context specific

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

  - Approximating Money

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

The Concept of Conservative Approximation

- x approximates y conservatively *iff*

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

- Validity of approximation is context specific

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

  - Approximating Money

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

  - Approximating Time

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

Two Important Objectives in Data Flow Analysis

- The discovered data flow information should be
  - *Sound*. Computed information should cover all run time behaviours
  - *Precise*. Information that does not correspond to any run time behaviour should be minimized

Two Important Objectives in Data Flow Analysis

- The discovered data flow information should be
  - *Sound*. Computed information should cover all run time behaviours
  - *Precise*. Information that does not correspond to any run time behaviour should be minimized
- The intended use of data flow information ($\equiv$ context) determines validity of approximation of data flow information

Conservative Approximation of Uncertain Information for Soundness

Live Variables

Static property at program point n

YES

NO

Conservative Approximation of Uncertain Information for Soundness

Live Variables

Static property at program point n	Dynamic behaviour at program point n
	YES along each path
YES	
	YES along some NO along others
NO	
	NO along each path

Conservative Approximation of Uncertain Information for Soundness

Live Variables		Available Expressions
Static property at program point n	Dynamic behaviour at program point n	Static property at program point n
	YES along each path	
YES		YES
	YES along some NO along others	
NO		NO
	NO along each path	

Conservative Approximation of Uncertain Information for Soundness

Live Variables		Available Expressions
Static property at program point n	Dynamic behaviour at program point n	Static property at program point n

YES ∪ NO = YES

YES along each path

Conservative

YES

YES along some
NO along others

YES

NO

Definite

NO along each path

NO

Conservative Approximation of Uncertain Information for Soundness

Live Variables

Available Expressions

Static property at program point n

Dynamic behaviour at program point n

Static property at program point n

YES $\cup$ NO = YES

YES along each path

YES

Conservative

Spurious inclusion

YES along some
NO along others

YES

NO

Definite

NO along each path

NO

Conservative Approximation of Uncertain Information for Soundness

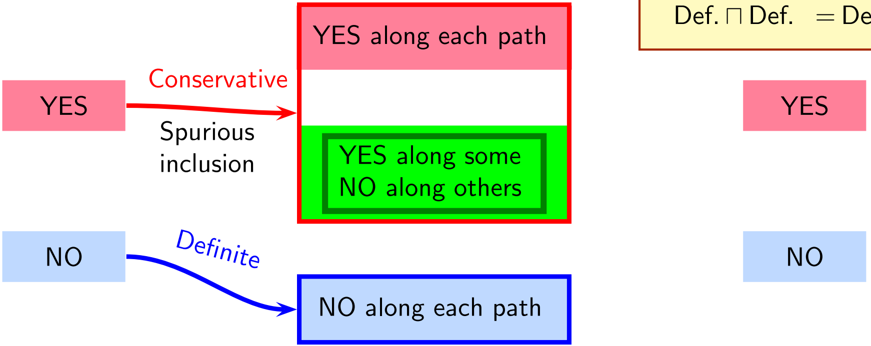
Live Variables

Available Expressions

Static property at program point n

Dynamic behaviour at program point n

$$
\begin{array}{rcl}
\text{Cons.} \sqcap \text{Cons.} & = & \text{Cons.} \\
\text{Cons.} \sqcap \text{Def.} & = & \text{Cons.} \\
\text{Def.} \sqcap \text{Cons.} & = & \text{Cons.} \\
\text{Def.} \sqcap \text{Def.} & = & \text{Def.}
\end{array}
$$



Conservative Approximation of Uncertain Information for Soundness

Live Variables		Available Expressions
Static property at program point n	Dynamic behaviour at program point n	Static property at program point n
	YES along each path	
YES	YES along some NO along others	YES
NO	NO along each path	NO

YES $\cap$ NO $=$ NO

Definite

Conservative

Conservative Approximation of Uncertain Information for Soundness

Live Variables		Available Expressions
Static property at program point n	Dynamic behaviour at program point n	Static property at program point n
	YES along each path	
YES		YES
	YES along some NO along others	
NO	NO along each path	NO

YES $\cap$ NO $=$ NO

Definite

Conservative

Spurious exclusion

Conservative Approximation of Uncertain Information for Soundness

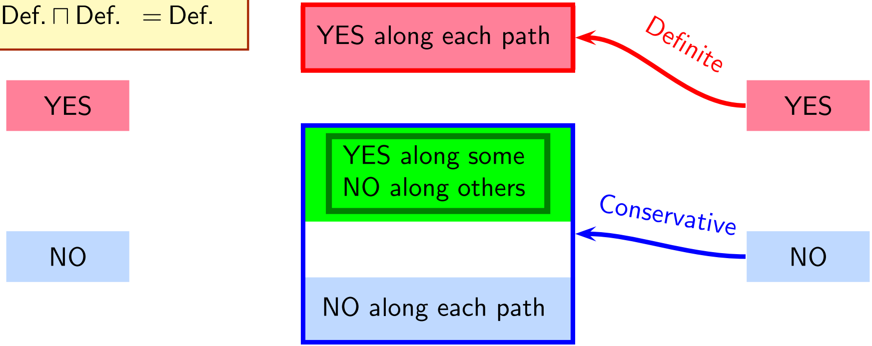
Live Variables

Available Expressions

Cons.	$\sqcap$	Cons.	=	Cons.
Cons.	$\sqcap$	Def.	=	Cons.
Def.	$\sqcap$	Cons.	=	Cons.
Def.	$\sqcap$	Def.	=	Def.

Dynamic behaviour at program point *n*

Static property at program point *n*



Conservative Approximation of Uncertain Information for Soundness

Live Variables

$$\begin{aligned}
\text{Cons.} \sqcap \text{Cons.} &= \text{Cons.}\\
\text{Cons.} \sqcap \text{Def.} &= \text{Cons.}\\
\text{Def.} \sqcap \text{Cons.} &= \text{Cons.}\\
\text{Def.} \sqcap \text{Def.} &= \text{Def.}
\end{aligned}$$

Dynamic behaviour at program point n

Available Expressions

$$\begin{aligned}
\text{Cons.} \sqcap \text{Cons.} &= \text{Cons.}\\
\text{Cons.} \sqcap \text{Def.} &= \text{Cons.}\\
\text{Def.} \sqcap \text{Cons.} &= \text{Cons.}\\
\text{Def.} \sqcap \text{Def.} &= \text{Def.}
\end{aligned}$$

YES along each path

Definite

YES

YES

Conservative

YES along some
NO along others

Conservative

NO

NO

Definite

NO along each path

Conservative Approximation of Uncertain Information for Soundness

Live Variables

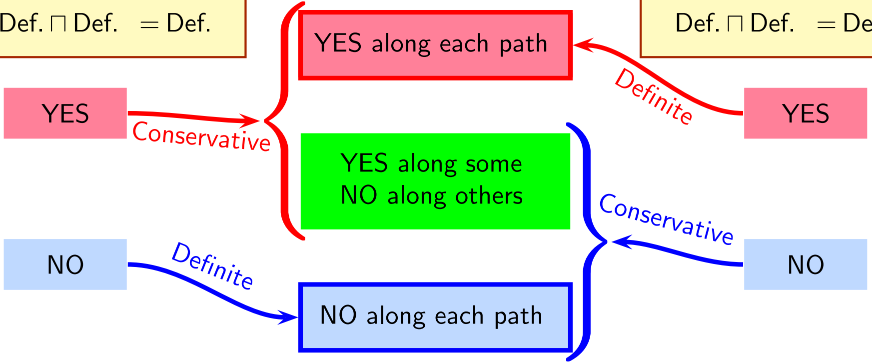
Cons. $\sqcap$ Cons.	=	Cons.	
Cons. $\sqcap$ Def.	=	Cons.	
Def. $\sqcap$ Cons.	=	Cons.	
Def. $\sqcap$ Def.	=	Def.	

Definite $\Rightarrow$ necessarily holds along each path

Conservative $\Rightarrow$ holds along some but not necessarily all

Available Expressions

Cons. $\sqcap$ Cons.	=	Cons.	
Cons. $\sqcap$ Def.	=	Cons.	
Def. $\sqcap$ Cons.	=	Cons.	
Def. $\sqcap$ Def.	=	Def.	



Conservative Approximation of Uncertain Information for Soundness

Live Variables

Available Expressions

Static p... erty at
progra... oint n

- Definite → Conservative **Harmless**
  - Information that must hold along all paths is assumed to hold along some but not all paths
  - Occurs because some spurious paths are considered (along which the information does not hold)
- Conservative → Definite **Harmful**
  - Information that holds along only some paths is assumed to hold along all paths
  - Occurs because some genuine paths are missed (along which the information holds)

Conservative, Definite, Every Path, Some Path . . .

- Definite $\rightarrow$ Conservative Harmless
  - Overapproximation of sets when confluence is union
    With $\cup$, inclusion of spurious paths may cause overapproximation
  - Underapproximation of sets when confluence is intersection
    With $\cap$, inclusion of spurious paths may cause underapproximation

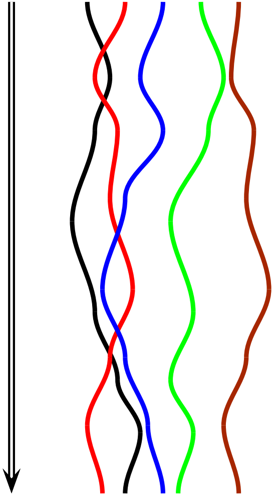
Including spurious paths may cause imprecision

- Conservative $\rightarrow$ Definite Harmful
  - Underapproximation of sets when confluence is union
    With $\cup$, exclusion of genuine paths may cause underapproximation
  - Overapproximation of sets when confluence is intersection
    With $\cap$, exclusion of genuine paths may cause overapproximation

Excluding genuine paths may cause unsoundness

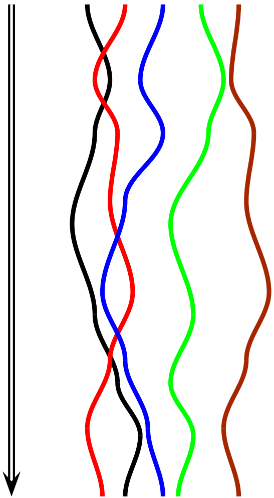
Static Analysis Computes Abstractions of Execution Paths

Execution Paths

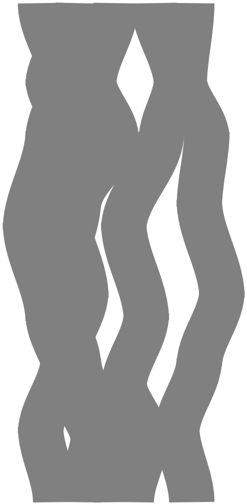


Static Analysis Computes Abstractions of Execution Paths

Execution Paths

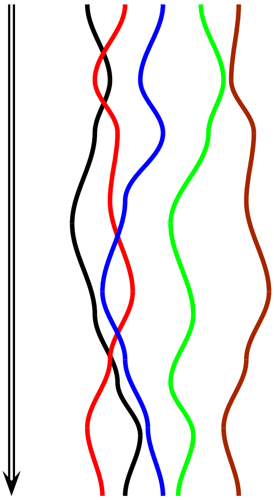


An Abstraction of Execution Paths

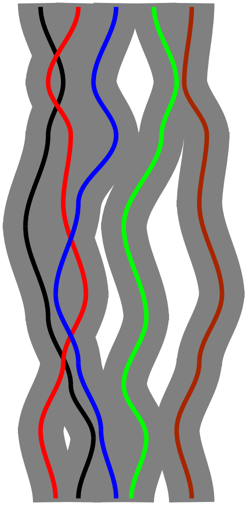


Static Analysis Computes Abstractions of Execution Paths

Execution Paths

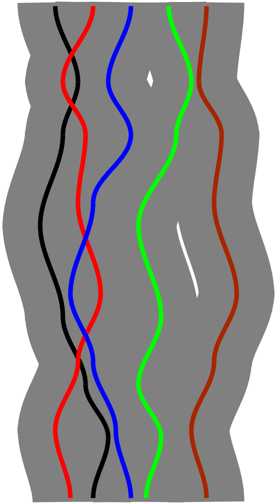


An Abstraction of Execution Paths



Soundness of Abstractions of Execution Paths

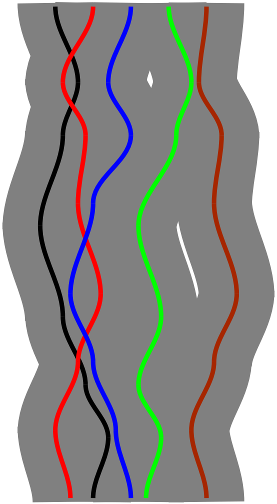
Sound



An over-approximation of execution paths is sound

Soundness of Abstractions of Execution Paths

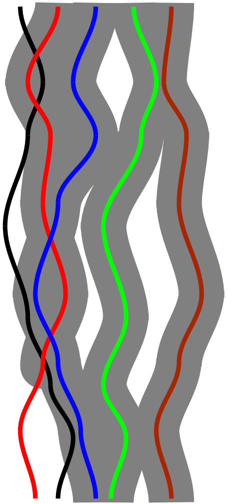
Sound



An over-approximation of execution paths is sound

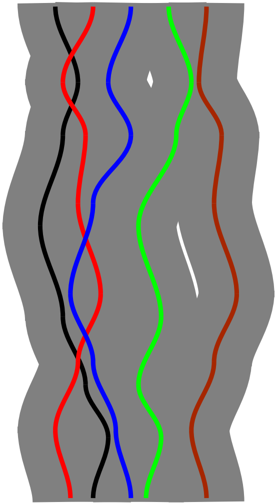
Missing any path (or a sub-path) causes unsoundness

Unsound



Soundness of Abstractions of Execution Paths

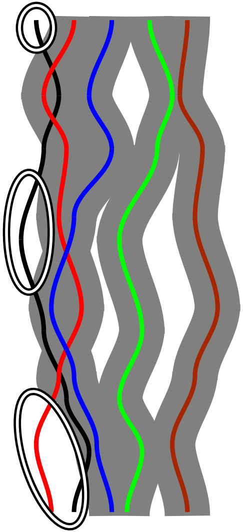
Sound



An over-approximation of execution paths is sound

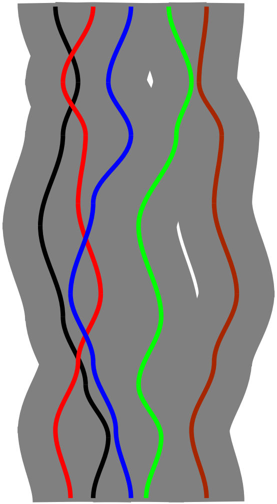
Missing any path (or a sub-path) causes unsoundness

Unsound



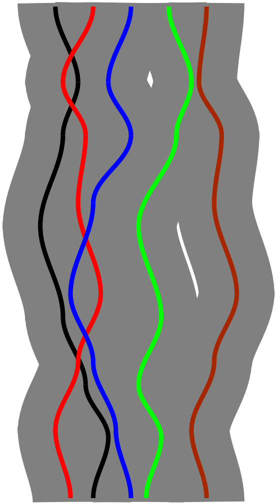
Precision of Sound Abstractions of Execution Paths

Sound but imprecise

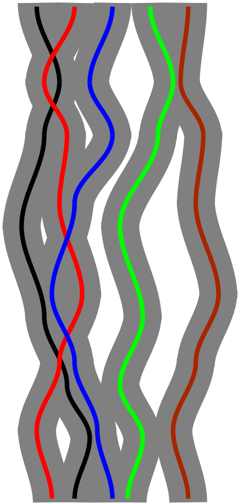


Precision of Sound Abstractions of Execution Paths

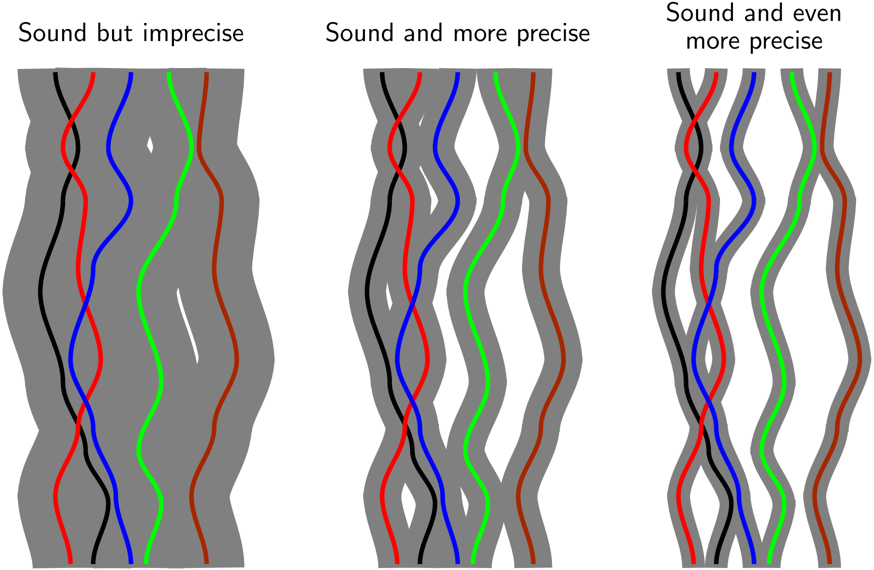
Sound but imprecise



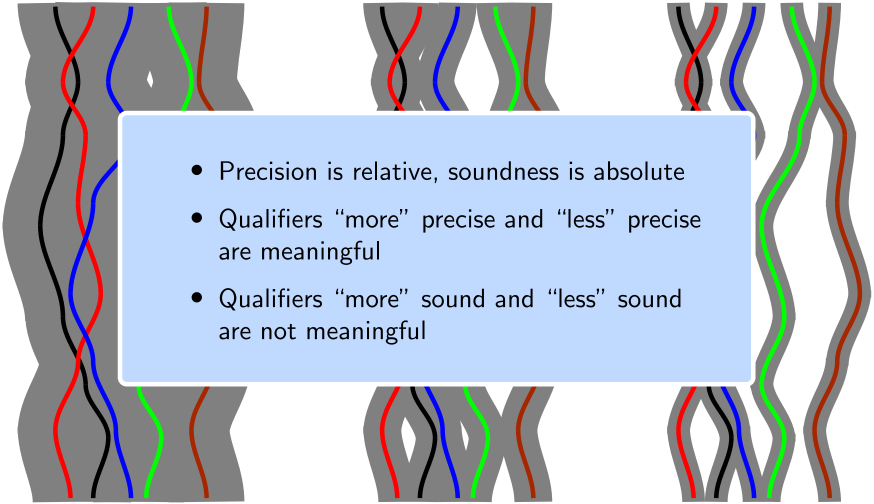
Sound and more precise



Precision of Sound Abstractions of Execution Paths



Precision of Sound Abstractions of Execution Paths



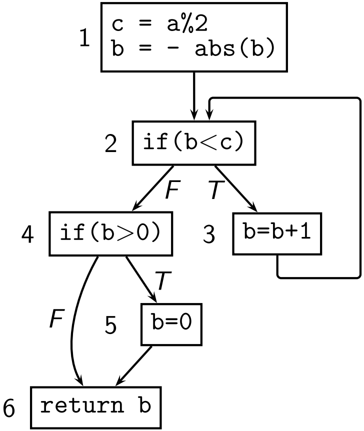
- Precision is relative, soundness is absolute
- Qualifiers “more” precise and “less” precise are meaningful
- Qualifiers “more” sound and “less” sound are not meaningful

Collecting Semantics for Representing Dynamic Behaviour at a Program Point

- A set $S \subseteq \Sigma$ of states σ reaching a program point along all execution traces
- Each state σ consists of two parts
  - Standard Semantics. Concrete values of variables
  - Instrumented Semantics. Other relevant properties that cannot be derived from values of variables
    Liveness of variables, availability of expressions, typestate etc.

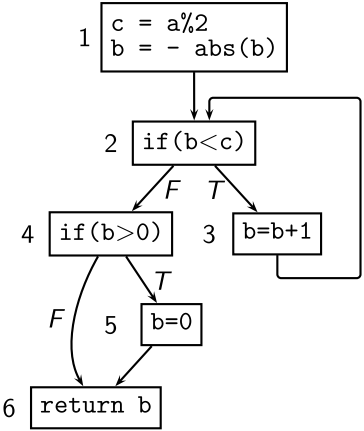
Example of States along a Trace

Consider a trace $(1, 2, 3, 2, 3, 2, 3, 2, 4, 5, 6)$ with $a = 5$, $b = 2$, $c = 7$ at the start



Example of States along a Trace

Consider a trace $(1, 2, 3, 2, 3, 2, 3, 2, 4, 5, 6)$ with $a = 5$, $b = 2$, $c = 7$ at the start

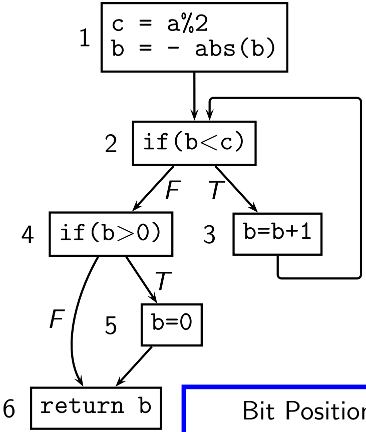


State σ reaching the entry of the node

	Standard Semantics	Instrumented semantics (for available expressions)
1	$\{a \mapsto 5, b \mapsto 2, c \mapsto 7\}$	0000
2	$\{a \mapsto 5, b \mapsto -2, c \mapsto 1\}$	1000
3	$\{a \mapsto 5, b \mapsto -2, c \mapsto 1\}$	1100
2	$\{a \mapsto 5, b \mapsto -1, c \mapsto 1\}$	1000
3	$\{a \mapsto 5, b \mapsto -1, c \mapsto 1\}$	1100
2	$\{a \mapsto 5, b \mapsto 0, c \mapsto 1\}$	1000
3	$\{a \mapsto 5, b \mapsto 0, c \mapsto 1\}$	1100
2	$\{a \mapsto 5, b \mapsto 1, c \mapsto 1\}$	1000
4	$\{a \mapsto 5, b \mapsto 1, c \mapsto 1\}$	1100
5	$\{a \mapsto 5, b \mapsto 1, c \mapsto 1\}$	1110
6	$\{a \mapsto 5, b \mapsto 0, c \mapsto 1\}$	1000

Example of States along a Trace

Consider a trace $(1, 2, 3, 2, 3, 2, 3, 2, 4, 5, 6)$ with $a = 5$, $b = 2$, $c = 7$ at the start



State σ reaching the entry of the node

	Standard Semantics	Instrumented semantics (for available expressions)
1	$\{a \mapsto 5, b \mapsto 2, c \mapsto 7\}$	0000
2	$\{a \mapsto 5, b \mapsto -2, c \mapsto 1\}$	1000
3	$\{a \mapsto 5, b \mapsto -2, c \mapsto 1\}$	1100
2	$\{a \mapsto 5, b \mapsto -1, c \mapsto 1\}$	1000
3	$\{a \mapsto 5, b \mapsto -1, c \mapsto 1\}$	1100
2	$\{a \mapsto 5, b \mapsto 0, c \mapsto 1\}$	1000
3	$\{a \mapsto 5, b \mapsto 0, c \mapsto 1\}$	1100
2	$\{a \mapsto 5, b \mapsto 1, c \mapsto 1\}$	1000
4	$\{a \mapsto 5, b \mapsto 1, c \mapsto 1\}$	1100
5	$\{a \mapsto 5, b \mapsto 1, c \mapsto 1\}$	1110
6	$\{a \mapsto 5, b \mapsto 0, c \mapsto 1\}$	1000

Bit Positions

0	1	2	3
$a\%2$	$b<c$	$b>0$	$b+1$

Concrete and Abstract Worlds

Collecting Semantics

$$\mathbb{C} = (2^{\Sigma}, \supseteq)$$

$\top = \emptyset$

$\bot = \Sigma$

Abstract Semantics

$$\mathbb{A} = (L, \sqsubseteq)$$

$\top$

$\bot$

Concrete and Abstract Worlds

Collecting Semantics

$\mathbb{C} = (2^{\Sigma}, \supseteq)$

$\top = \emptyset$

$\bot = \Sigma$

Abstract Semantics

$\mathbb{A} = (L, \sqsubseteq)$

$\top$

$\bot$

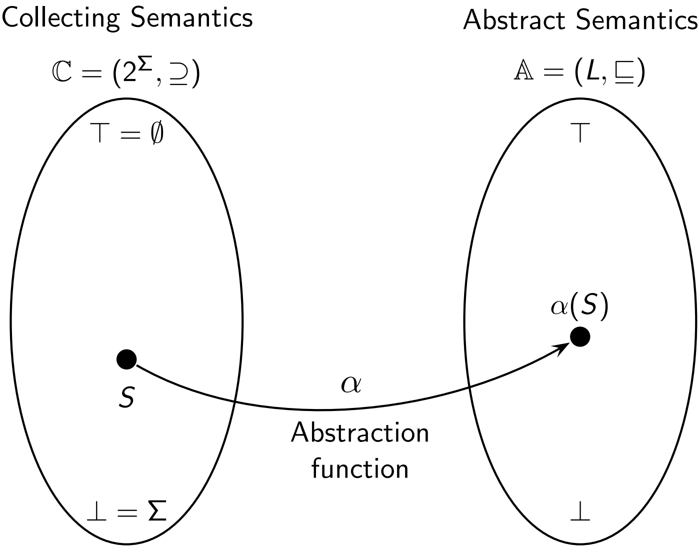
Collecting Semantics at a program point u

- Set $S \subseteq \Sigma$ of states σ
  (reaching u along all traces)

Abstract Semantics at a program point u

- Data flow value $D \in L$
  (computed by data flow analysis)

Concrete and Abstract Worlds



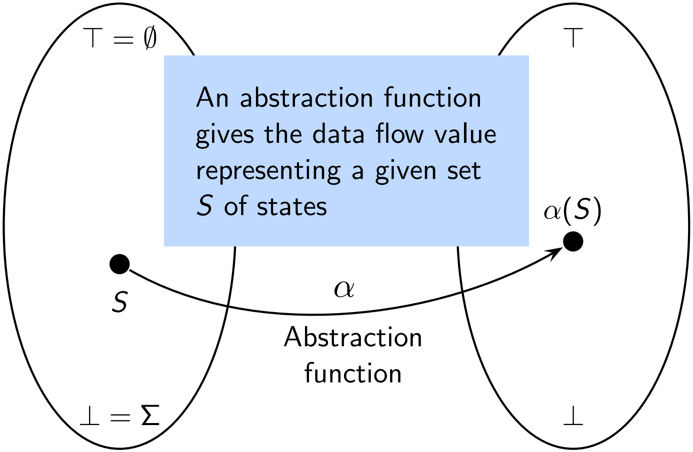
Concrete and Abstract Worlds

Collecting Semantics

$\mathbb{C} = (2^{\Sigma}, \supseteq)$

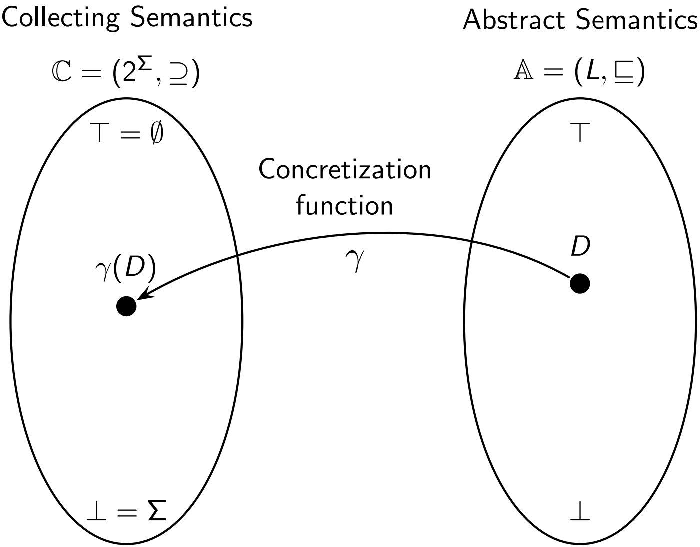
Abstract Semantics

$\mathbb{A} = (L, \sqsubseteq)$

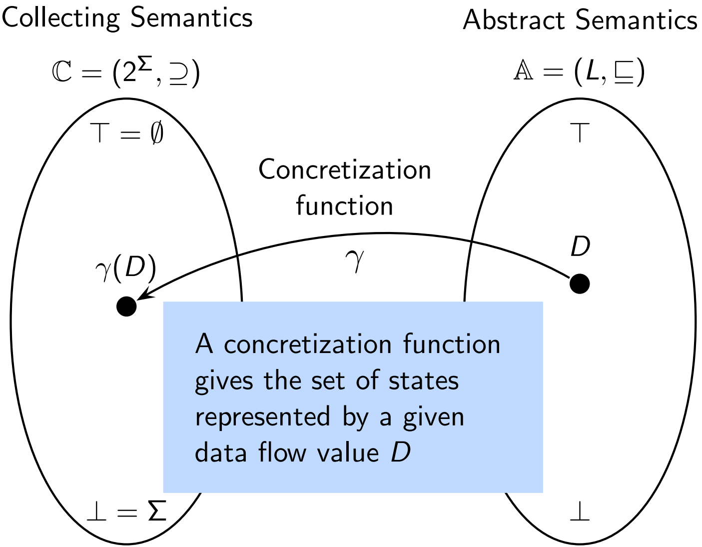


An abstraction function gives the data flow value representing a given set S of states

Concrete and Abstract Worlds



Concrete and Abstract Worlds



A concretization function gives the set of states represented by a given data flow value D

Representation Function for States

- Each state σ consists of two parts
  - Standard Semantics. Concrete values of variables
  - Instrumented Semantics. Other relevant properties that cannot be derived from values of variables
    Liveness of variables, availability of expressions etc.
- Representation function β for an analysis extracts the part of a state that is relevant to the analysis
  - For available expressions analysis, $\beta(\sigma)$ returns the instrumentation semantics in σ representing the expressions that are available
  - For live variables analysis, $\beta(\sigma)$ returns the instrumentation semantics in σ representing the variables that are live
  - For constant propagation, $\beta(\sigma)$ returns the standard semantics in σ representing the values of variables

Illustrating Concrete and Abstract Worlds for Available Expressions Analysis

- Assume that our program has two expressions $\mathbb{E}\text{xpr} = \{a*b, b*c\}$
- Then the instrumented semantics in a state σ is expressed by $\varepsilon \subseteq \mathbb{E}\text{xpr}$
- For simplicity of illustration
  - We ignore the standard semantics in σ and hence $\beta(\sigma) = \sigma = \varepsilon \subseteq \mathbb{E}\text{xpr}$ and $\Sigma = 2^{\mathbb{E}\text{xpr}}$
  - We represent $\varepsilon \subseteq \mathbb{E}\text{xpr}$ using a bit vector of two bits

  Then, our

  - concrete world is $\mathbb{C} = (2^{2^{\mathbb{E}\text{xpr}}}, \supseteq)$
  - abstract world is $\mathbb{A} = (2^{\mathbb{E}\text{xpr}}, \sqsubseteq)$ where $\sqsubseteq$ is $\subseteq$

Illustrating Concrete and Abstract Worlds for Available Expressions Analysis

- Assume that our program has two expressions $\mathbb{E}\text{xpr} = \{a * b, b * c\}$
- Then th… $\subseteq \mathbb{E}\text{xpr}$
- For simp…
  - We …
    $\beta($…
  - We …

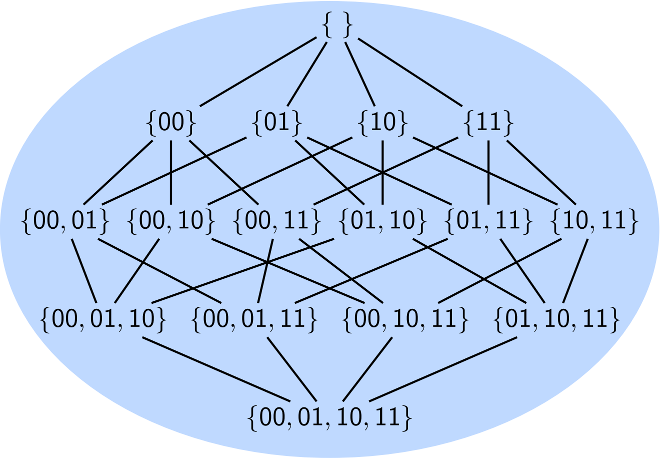
Then, ou…

- A single state σ is subset of $\mathbb{E}\text{xpr}$
- A set of states S is a subset of $2^{\mathbb{E}\text{xpr}}$
- The concrete world is a set of S, i.e., $2^{2^{\mathbb{E}\text{xpr}}}$ (i.e., a set of set of states)

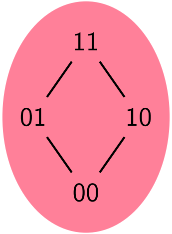
  - concrete world is $\mathbb{C} = (2^{2^{\mathbb{E}\text{xpr}}}, \supseteq)$
  - abstract world is $\mathbb{A} = (2^{\mathbb{E}\text{xpr}}, \sqsubseteq)$ where $\sqsubseteq$ is $\subseteq$

Illustrating Concrete and Abstract Worlds for Available Expressions Analysis

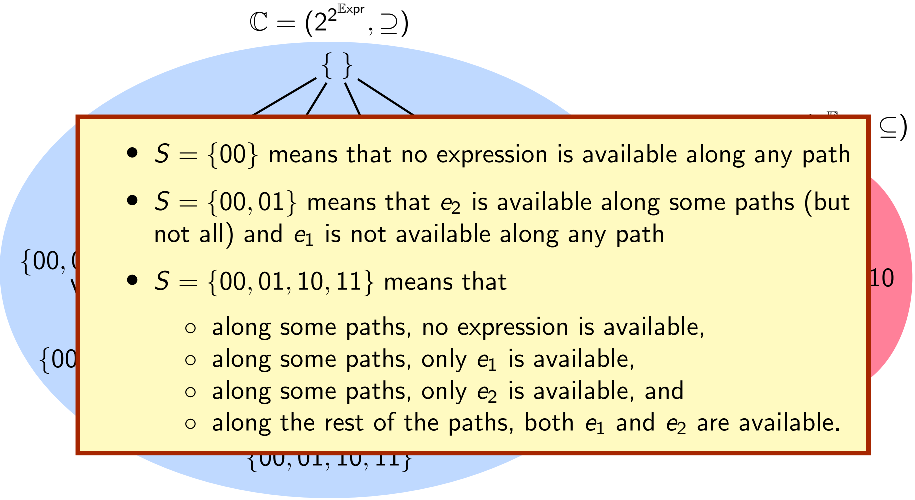
$\mathbb{C} = (2^{2^{\mathbb{E}\mathsf{xpr}}}, \supseteq)$



$\mathbb{A} = (2^{\mathbb{E}\mathsf{xpr}}, \subseteq)$



Illustrating Concrete and Abstract Worlds for Available Expressions Analysis



- $S = \{00\}$ means that no expression is available along any path
- $S = \{00, 01\}$ means that e_2 is available along some paths (but not all) and e_1 is not available along any path
- $S = \{00, 01, 10, 11\}$ means that
  - along some paths, no expression is available,
  - along some paths, only e_1 is available,
  - along some paths, only e_2 is available, and
  - along the rest of the paths, both e_1 and e_2 are available.

Illustrating Concrete and Abstract Worlds for Available Expressions Analysis

$\mathbb{C} = (2^{2^{\mathbb{E}\mathsf{xpr}}}, \supseteq)$

$\{\ \}$

$\{00, 01\}$

$\{00, 01$

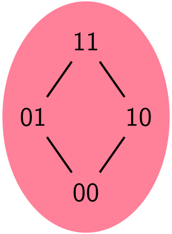
In the concrete world, $S_1 \supseteq S_2$ means that S_1 captures all behaviours represented by S_2 and may contain additional behaviours

- S_1 is a sound over-approximations of S_2
- S_1 may be more imprecise than S_2

What can we say about the abstract world?

We need to understand the properties of α and γ to answer the question

$\mathbb{A} = (2^{\mathbb{E}\mathsf{xpr}}, \subseteq)$



Abstraction and Concretization Functions for Available Expressions Analysis

$\beta(\sigma)$ gives the set of expressions available in a state

- An expression is available at a program point provided it is contained in $\beta(\sigma)$ of every state σ reaching the program point

$$\alpha(S) = \bigcap_{\sigma \in S} \beta(\sigma)$$

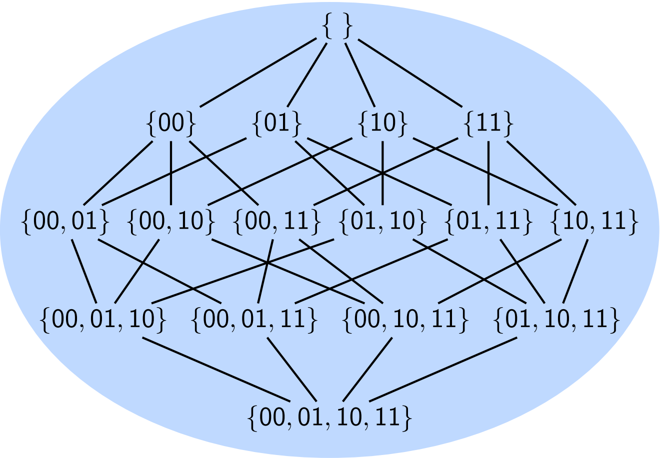
- A data flow value D of expressions available at a program point represents all states σ reaching the program point such that $\beta(\sigma)$ contains all expressions in D (and may contain some expressions not contained in D)

$$\gamma(D) = \{\sigma \mid \beta(\sigma) \supseteq D\}$$

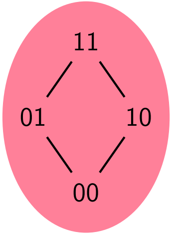
If we take $\beta(\sigma) \subseteq D$, then $\alpha(\gamma(D)) \subseteq D$ which is not acceptable

Illustrating Abstraction and Concretization Functions for Available Expressions Analysis

$\mathbb{C} = (2^{2^{\mathbb{E}\mathsf{xpr}}}, \supseteq)$



$\mathbb{A} = (2^{\mathbb{E}\mathsf{xpr}}, \subseteq)$



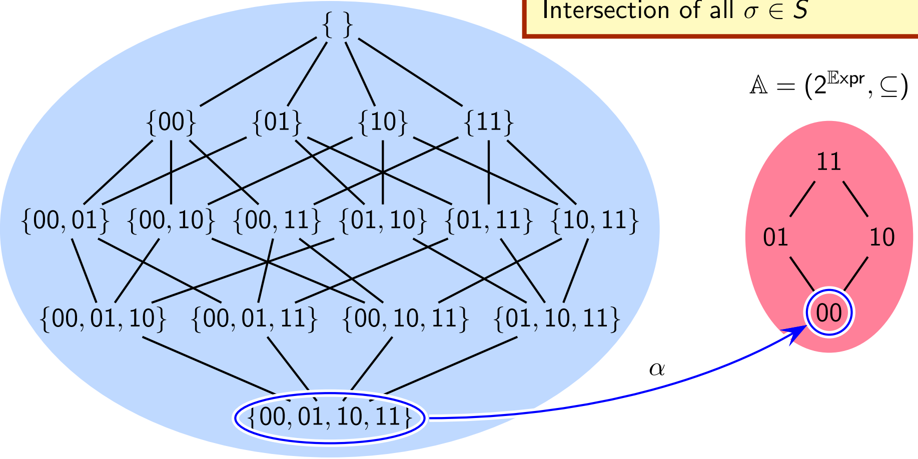
Illustrating Abstraction and Concretization Functions for Available Expressions Analysis

Abstraction Function $\alpha(S) = \bigcap_{\sigma \in S} \sigma$

Intersection of all $\sigma \in S$

$\mathbb{C} = (2^{2^{\mathbb{E}\mathrm{xpr}}}, \supseteq)$

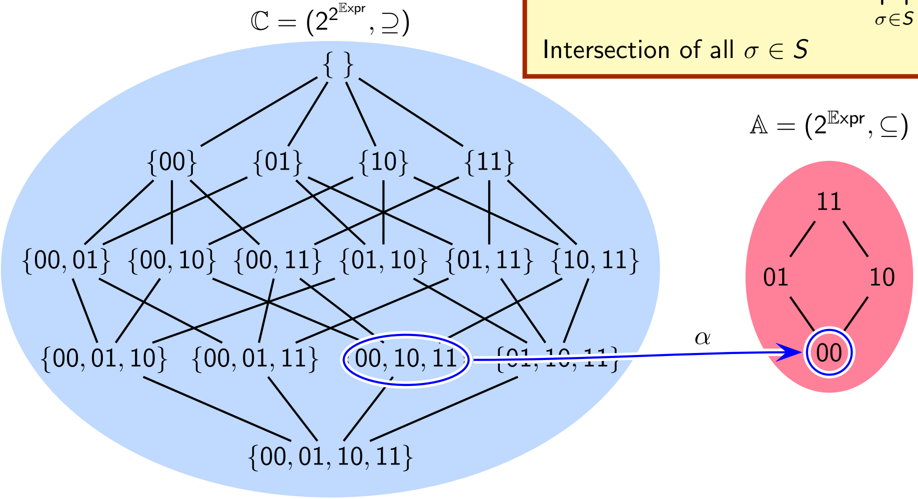
$\mathbb{A} = (2^{\mathbb{E}\mathrm{xpr}}, \subseteq)$



Illustrating Abstraction and Concretization Functions for Available Expressions Analysis

Abstraction Function $\alpha(S) = \bigcap_{\sigma \in S} \sigma$

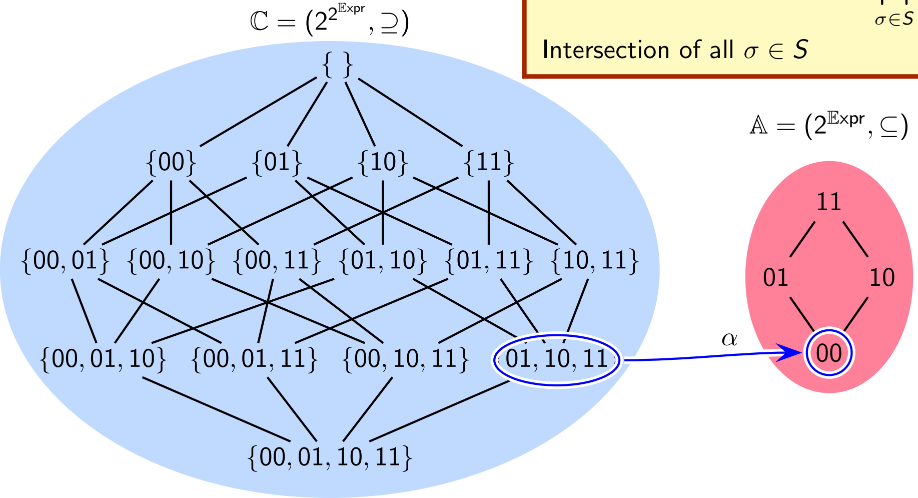
Intersection of all $\sigma \in S$



Illustrating Abstraction and Concretization Functions for Available Expressions Analysis

Abstraction Function $\alpha(S) = \bigcap_{\sigma \in S} \sigma$

Intersection of all $\sigma \in S$

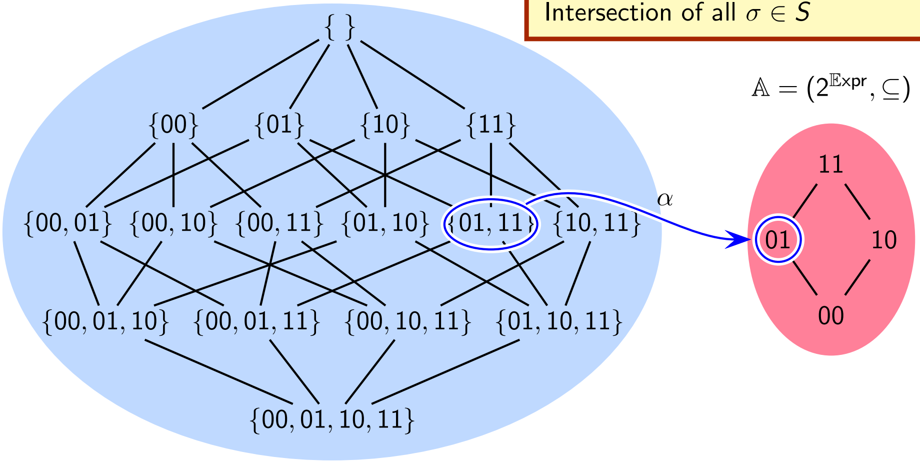


Illustrating Abstraction and Concretization Functions for Available Expressions Analysis

Abstraction Function $\alpha(S) = \bigcap\limits_{\sigma \in S} \sigma$

Intersection of all $\sigma \in S$

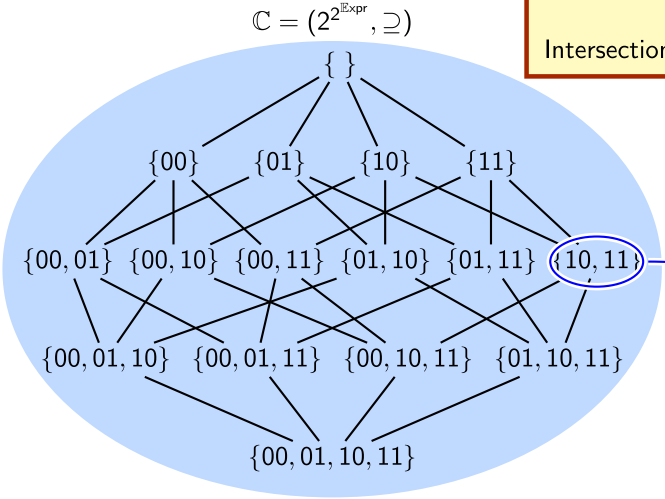
$\mathbb{C} = (2^{2^{\mathbb{E}\mathsf{xpr}}}, \supseteq)$

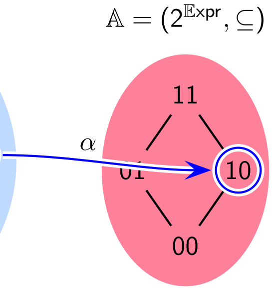


Illustrating Abstraction and Concretization Functions for Available Expressions Analysis

Abstraction Function $\alpha(S) = \bigcap\limits_{\sigma \in S} \sigma$

Intersection of all $\sigma \in S$

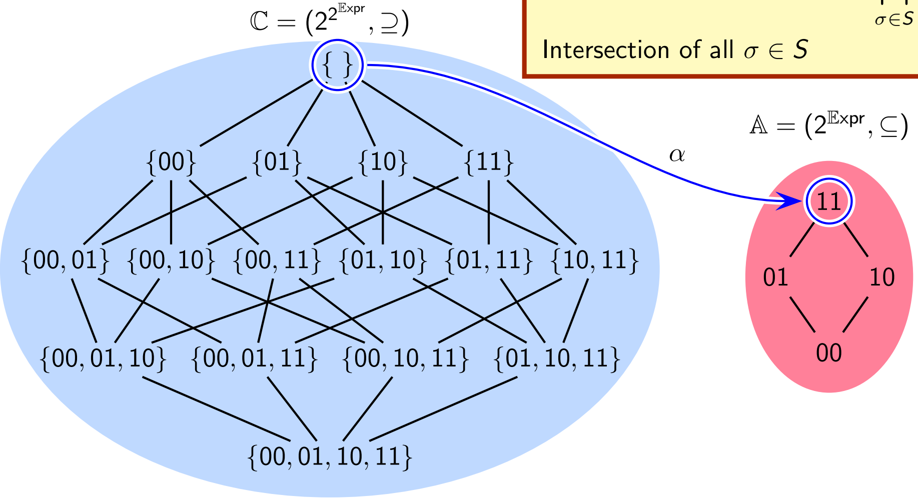




Illustrating Abstraction and Concretization Functions for Available Expressions Analysis

Abstraction Function $\alpha(S) = \bigcap_{\sigma \in S} \sigma$

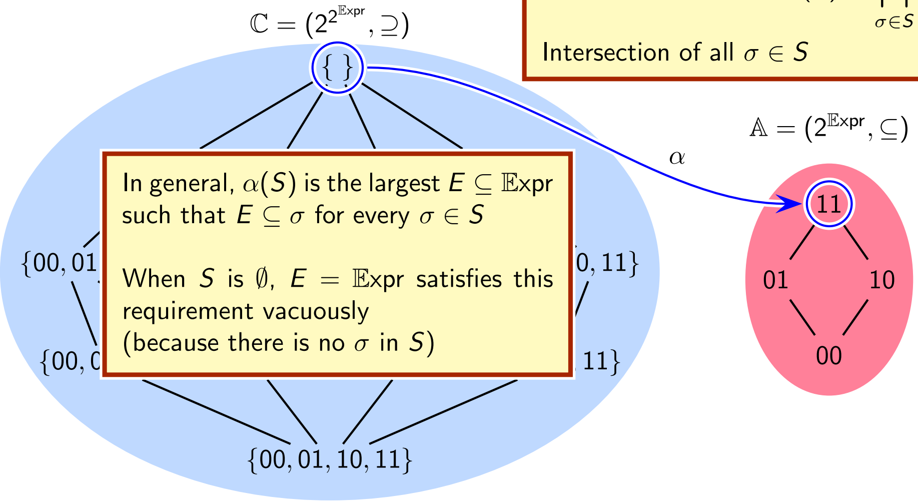
Intersection of all $\sigma \in S$



Illustrating Abstraction and Concretization Functions for Available Expressions Analysis

Abstraction Function $\alpha(S) = \bigcap_{\sigma \in S} \sigma$

Intersection of all $\sigma \in S$



In general, $\alpha(S)$ is the largest $E \subseteq \mathbb{E}\text{xpr}$ such that $E \subseteq \sigma$ for every $\sigma \in S$

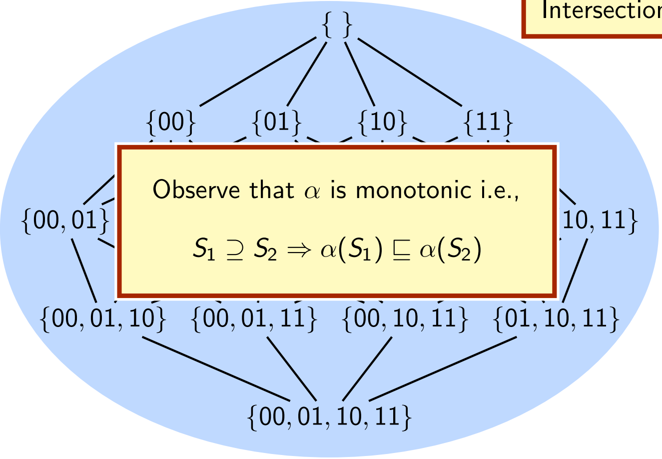
When S is $\emptyset$, $E = \mathbb{E}\text{xpr}$ satisfies this requirement vacuously (because there is no σ in S)

Illustrating Abstraction and Concretization Functions for Available Expressions Analysis

Abstraction Function $\alpha(S) = \bigcap_{\sigma \in S} \sigma$

Intersection of all $\sigma \in S$

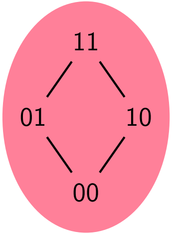
$\mathbb{C} = (2^{2^{\mathbb{E}\text{xpr}}}, \supseteq)$



Observe that α is monotonic i.e.,

$$S_1 \supseteq S_2 \Rightarrow \alpha(S_1) \sqsubseteq \alpha(S_2)$$

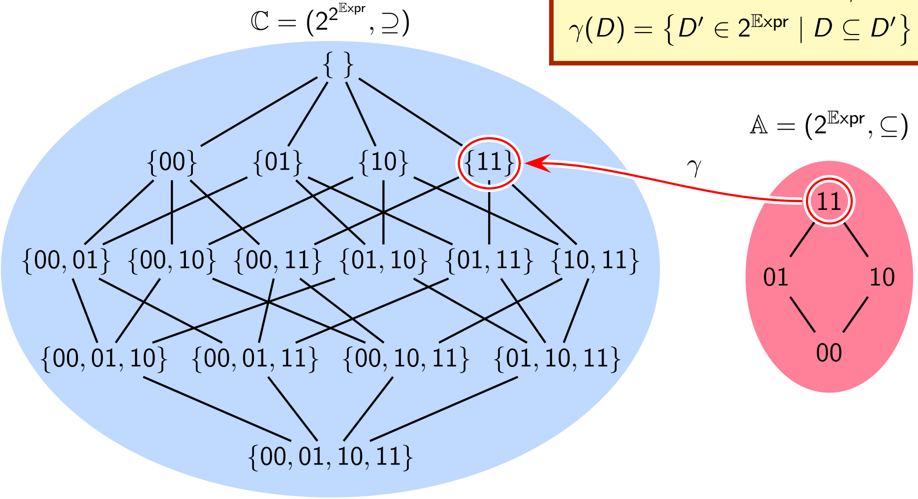
$\mathbb{A} = (2^{\mathbb{E}\text{xpr}}, \subseteq)$



Illustrating Abstraction and Concretization Functions for Available Expressions Analysis

Concretization Function γ

$\gamma(D) = \left\{ D' \in 2^{\mathbb{E}\text{xpr}} \mid D \subseteq D' \right\}$



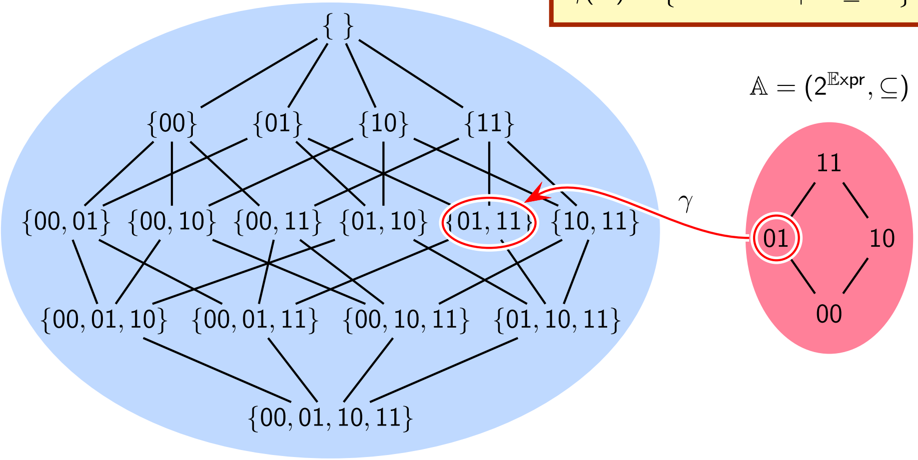
Illustrating Abstraction and Concretization Functions for Available Expressions Analysis

Concretization Function γ

$\gamma(D) = \left\{ D' \in 2^{\mathbb{E}\mathsf{xpr}} \mid D \subseteq D' \right\}$

$\mathbb{C} = (2^{2^{\mathbb{E}\mathsf{xpr}}}, \supseteq)$

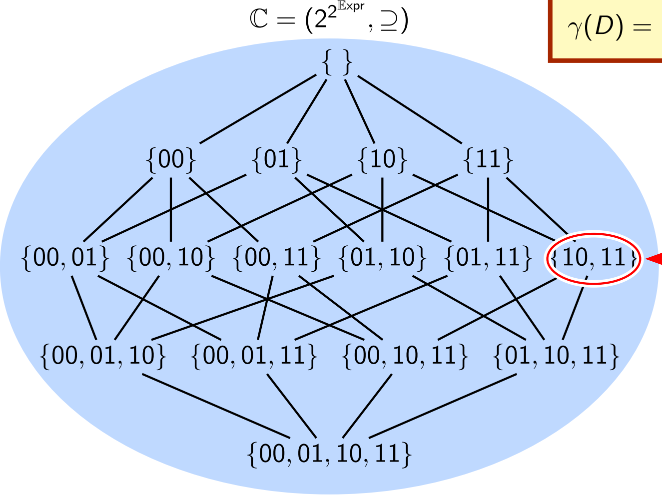
$\mathbb{A} = (2^{\mathbb{E}\mathsf{xpr}}, \subseteq)$

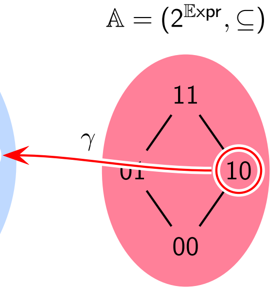


Illustrating Abstraction and Concretization Functions for Available Expressions Analysis

Concretization Function γ

$$\gamma(D) = \left\{ D' \in 2^{\mathbb{E}\text{xpr}} \mid D \subseteq D' \right\}$$





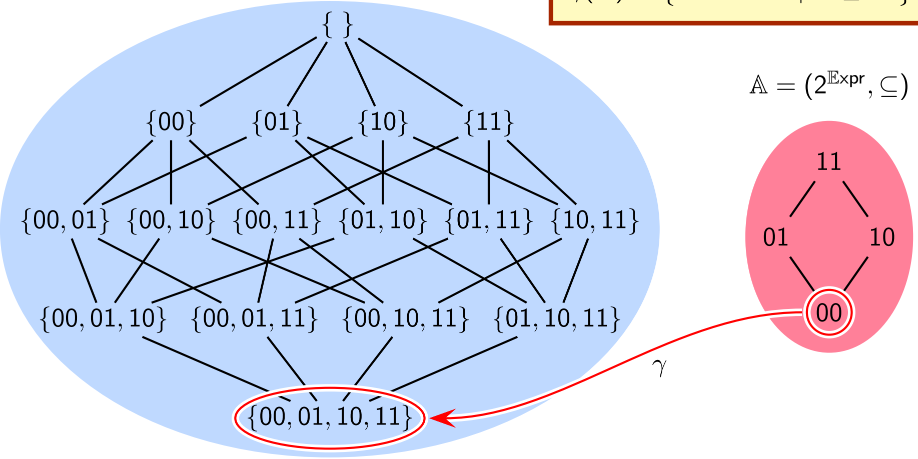
Illustrating Abstraction and Concretization Functions for Available Expressions Analysis

Concretization Function γ

$\gamma(D) = \{D' \in 2^{\mathbb{E}\text{xpr}} \mid D \subseteq D'\}$

$\mathbb{C} = (2^{2^{\mathbb{E}\text{xpr}}}, \supseteq)$

$\mathbb{A} = (2^{\mathbb{E}\text{xpr}}, \subseteq)$

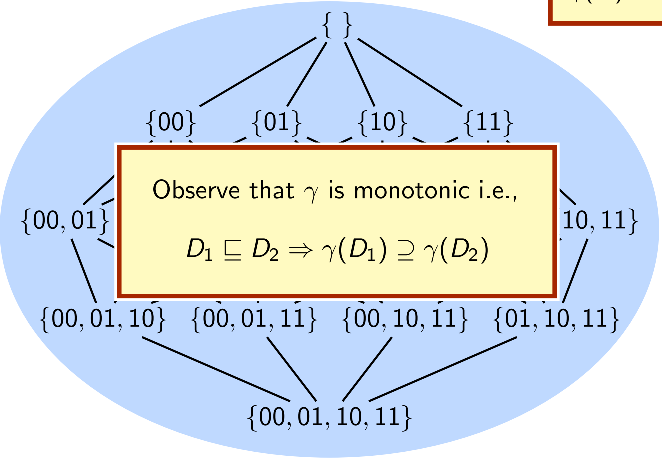


Illustrating Abstraction and Concretization Functions for Available Expressions Analysis

Concretization Function γ

$\gamma(D) = \left\{ D' \in 2^{\mathbb{E}\text{xpr}} \mid D \subseteq D' \right\}$

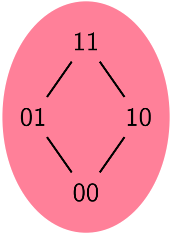
$\mathbb{C} = (2^{2^{\mathbb{E}\text{xpr}}}, \supseteq)$



Observe that γ is monotonic i.e.,

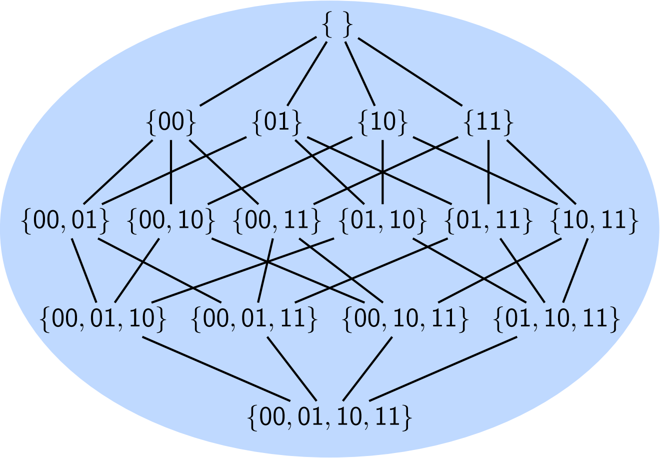
$$D_1 \sqsubseteq D_2 \Rightarrow \gamma(D_1) \supseteq \gamma(D_2)$$

$\mathbb{A} = (2^{\mathbb{E}\text{xpr}}, \subseteq)$

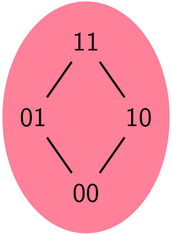


The Role of $\top$ Value in Available Expressions Analysis

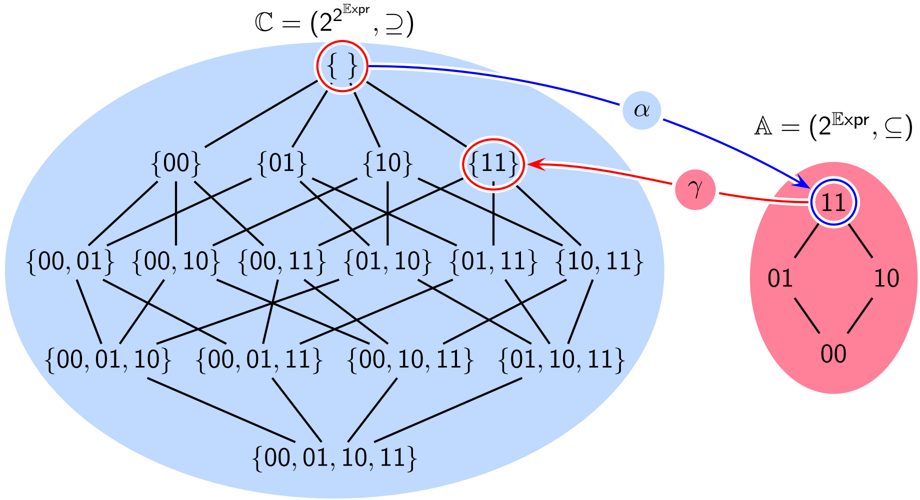
$\mathbb{C} = (2^{2^{\mathbb{E}\mathsf{xpr}}}, \supseteq)$



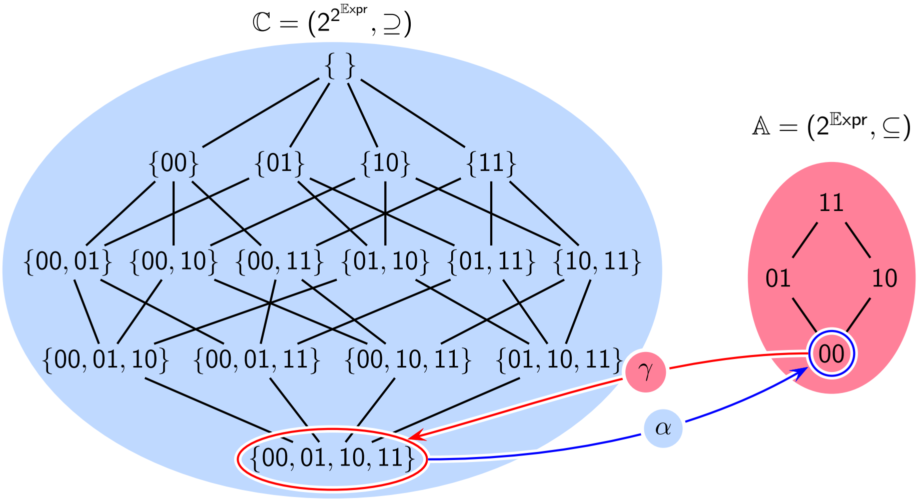
$\mathbb{A} = (2^{\mathbb{E}\mathsf{xpr}}, \subseteq)$



The Role of $\top$ Value in Available Expressions Analysis

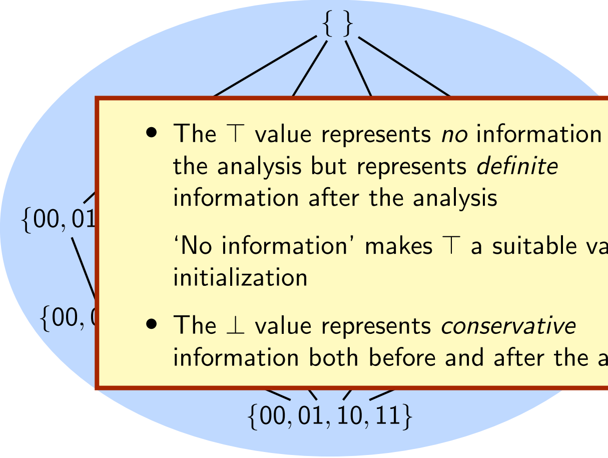


The Role of $\top$ Value in Available Expressions Analysis



The Role of $\top$ Value in Available Expressions Analysis

$\mathbb{C} = (2^{2^{\mathbb{E}\mathsf{xpr}}}, \supseteq)$

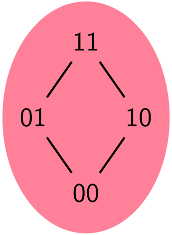


- The $\top$ value represents *no* information before the analysis but represents *definite* information after the analysis

  'No information' makes $\top$ a suitable value for initialization

- The $\bot$ value represents *conservative* information both before and after the analysis

$\mathbb{A} = (2^{\mathbb{E}\mathsf{xpr}}, \subseteq)$



Abstraction and Concretization Functions for Live Variables Analysis

$\beta(\sigma)$ gives the set of live variables in a state

- A variable is live at a program point provided it is contained in $\beta(\sigma)$ of some state σ reaching the program point

$$\forall S \in \mathbb{C}.\ \ \alpha(S) = \bigcup_{\sigma \in S} \beta(\sigma)$$

- A data flow value D of variables live at a program point represents all states σ reaching the program point such that $\beta(\sigma)$ does not contain any variables not contained in D (and may not contain some variable contained in D)

$$\forall D \in \mathbb{A}.\ \ \gamma(D) = \{\sigma \mid \beta(\sigma) \subseteq D\}$$

If we take $\beta(\sigma) \supseteq D$, then $\alpha(\gamma(D)) \supseteq D$ which is not acceptable

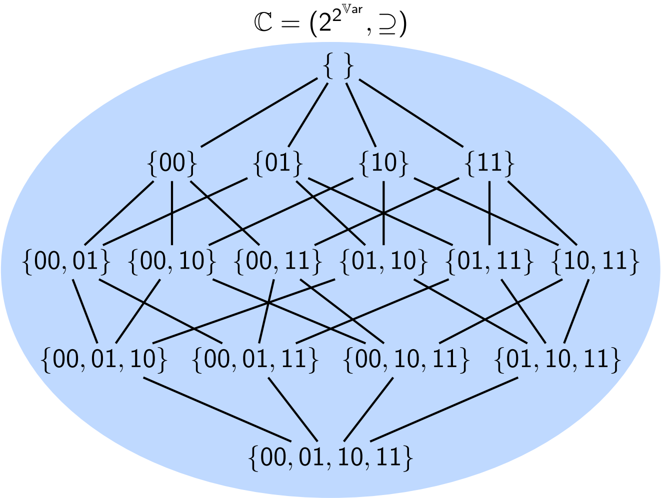
Illustrating Concrete and Abstract Worlds for Live Variables Analysis

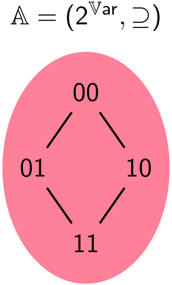
- Assume that our program has two variables $\mathbb{V}\text{ar} = \{a, b\}$
- Then the instrumented semantics in a state σ is expressed by $V \subseteq \mathbb{V}\text{ar}$
- For simplicity of illustration
  - We ignore the standard semantics in σ and hence $\beta(\sigma) = \sigma = V \subseteq \mathbb{V}\text{ar}$ and $\Sigma = 2^{\mathbb{V}\text{ar}}$
  - We represent $V \subseteq \mathbb{V}\text{ar}$ using a bit vector of two bits

Then, our

- concrete world is $\mathbb{C} = (2^{2^{\mathbb{V}\text{ar}}}, \supseteq)$
- abstract world is $\mathbb{A} = (2^{\mathbb{V}\text{ar}}, \sqsubseteq)$ where $\sqsubseteq$ is $\supseteq$

Illustrating Abstraction and Concretization Functions for Live Variables Analysis

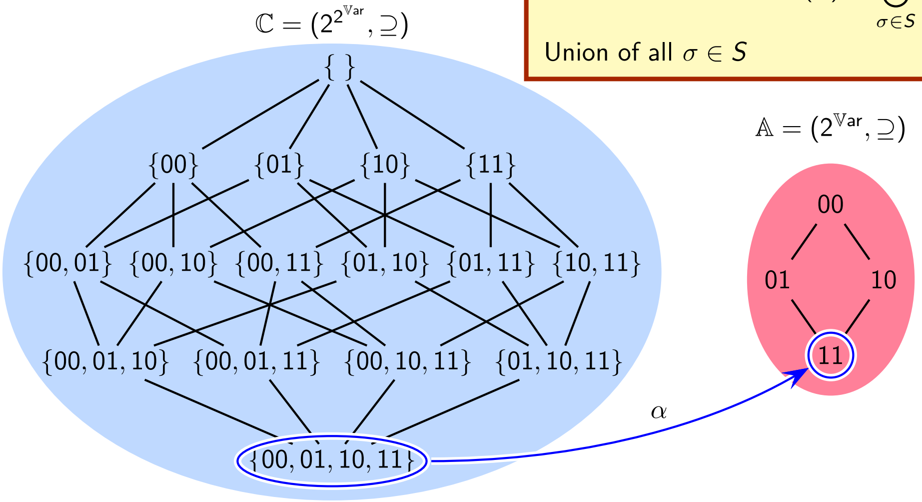




Illustrating Abstraction and Concretization Functions for Live Variables Analysis

Abstraction Function $\alpha(S) = \bigcup_{\sigma \in S} \sigma$

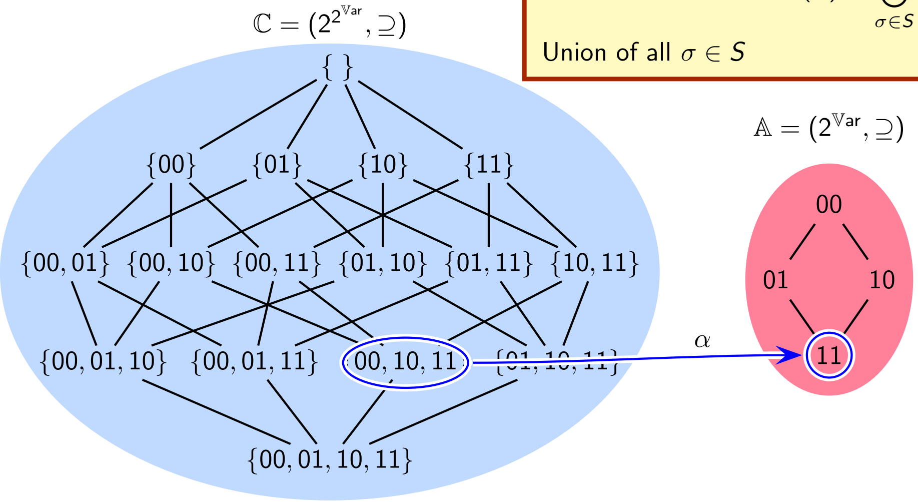
Union of all $\sigma \in S$



Illustrating Abstraction and Concretization Functions for Live Variables Analysis

Abstraction Function $\alpha(S) = \bigcup_{\sigma \in S} \sigma$

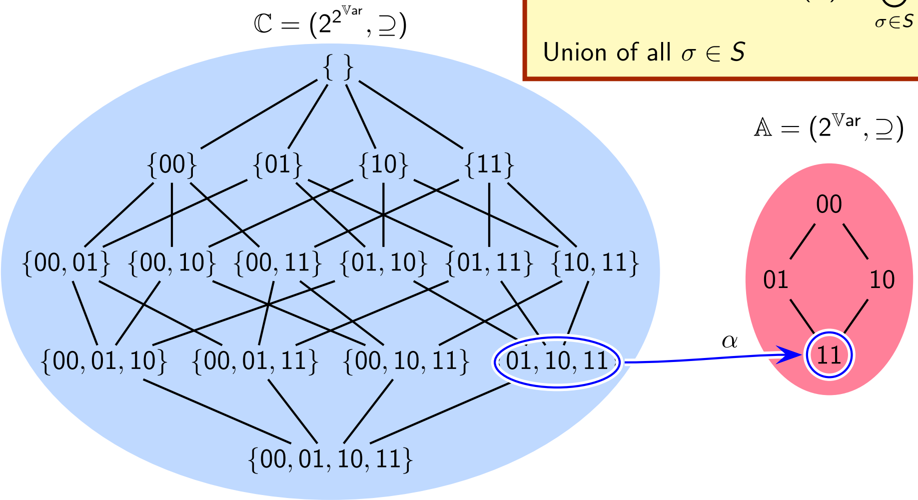
Union of all $\sigma \in S$



Illustrating Abstraction and Concretization Functions for Live Variables Analysis

Abstraction Function $\alpha(S) = \bigcup\limits_{\sigma \in S} \sigma$

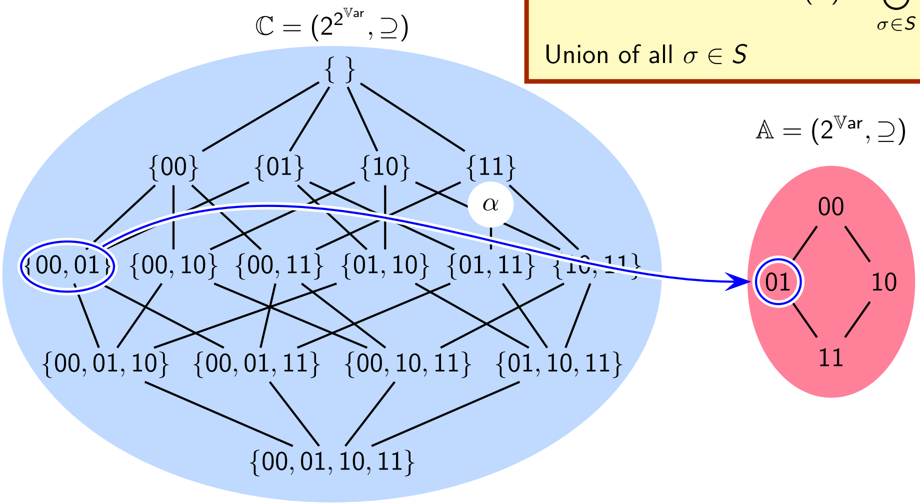
Union of all $\sigma \in S$



Illustrating Abstraction and Concretization Functions for Live Variables Analysis

Abstraction Function $\alpha(S) = \bigcup_{\sigma \in S} \sigma$

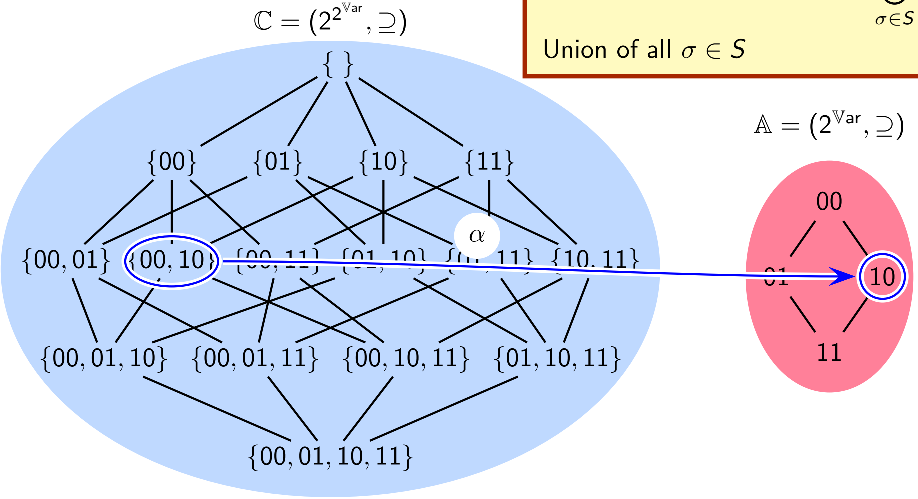
Union of all $\sigma \in S$



Illustrating Abstraction and Concretization Functions for Live Variables Analysis

Abstraction Function $\alpha(S) = \bigcup_{\sigma \in S} \sigma$

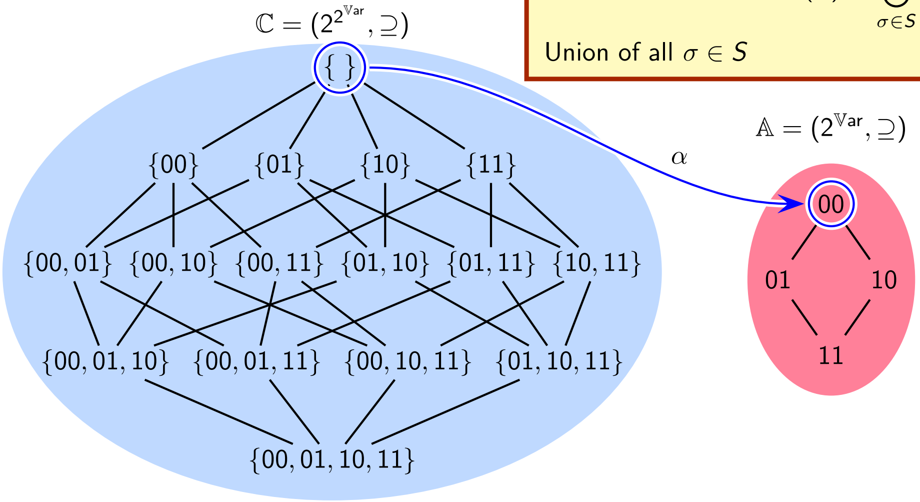
Union of all $\sigma \in S$



Illustrating Abstraction and Concretization Functions for Live Variables Analysis

Abstraction Function $\alpha(S) = \bigcup_{\sigma \in S} \sigma$

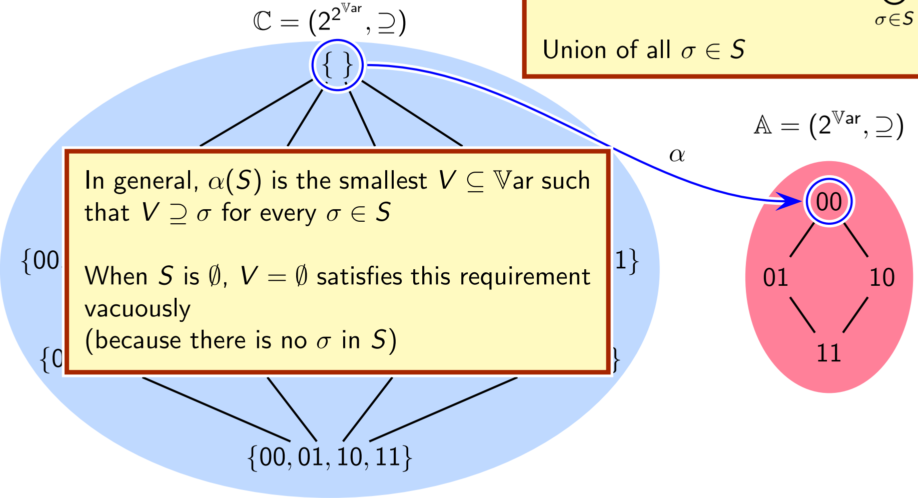
Union of all $\sigma \in S$



Illustrating Abstraction and Concretization Functions for Live Variables Analysis

Abstraction Function $\alpha(S) = \bigcup\limits_{\sigma \in S} \sigma$

Union of all $\sigma \in S$



In general, $\alpha(S)$ is the smallest $V \subseteq \mathbb{V}\mathsf{ar}$ such that $V \supseteq \sigma$ for every $\sigma \in S$

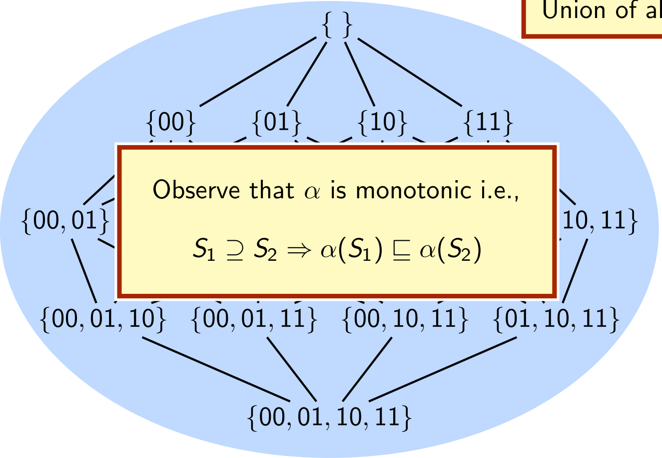
When S is $\emptyset$, $V = \emptyset$ satisfies this requirement vacuously
(because there is no σ in S)

Illustrating Abstraction and Concretization Functions for Live Variables Analysis

Abstraction Function $\alpha(S) = \bigcup_{\sigma \in S} \sigma$

Union of all $\sigma \in S$

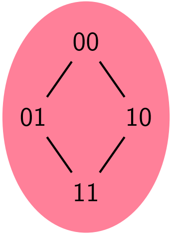
$\mathbb{C} = (2^{2^{\mathbb{V}\mathrm{ar}}}, \supseteq)$



Observe that α is monotonic i.e.,

$S_1 \supseteq S_2 \Rightarrow \alpha(S_1) \sqsubseteq \alpha(S_2)$

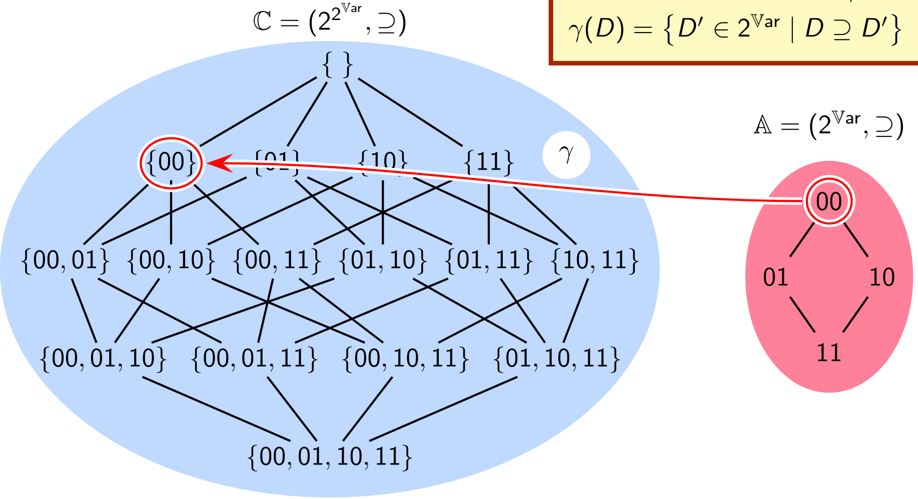
$\mathbb{A} = (2^{\mathbb{V}\mathrm{ar}}, \supseteq)$



Illustrating Abstraction and Concretization Functions for Live Variables Analysis

Concretization Function γ

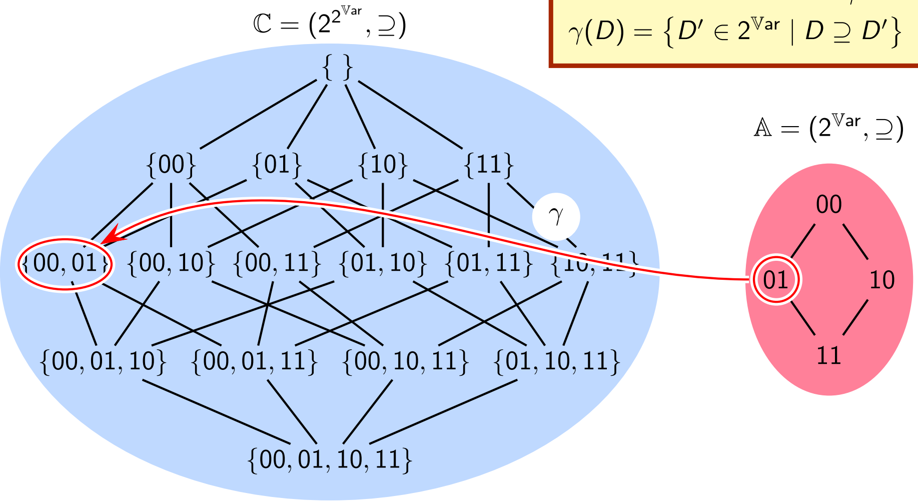
$\gamma(D) = \left\{ D' \in 2^{\mathbb{V}\mathsf{ar}} \mid D \supseteq D' \right\}$



Illustrating Abstraction and Concretization Functions for Live Variables Analysis

Concretization Function γ

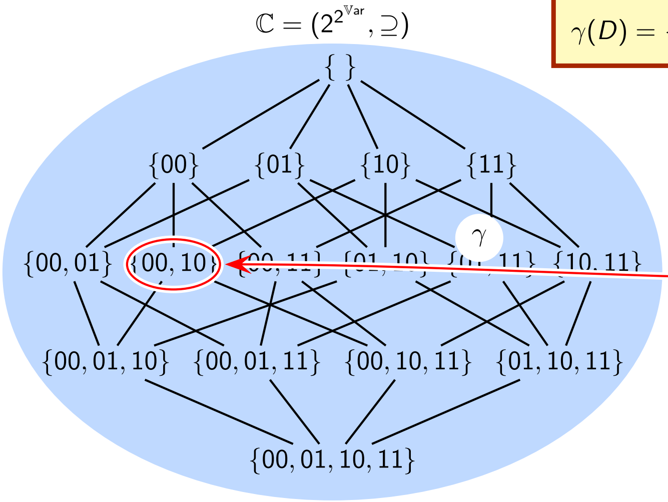
$\gamma(D) = \left\{D' \in 2^{\mathbb{V}\text{ar}} \mid D \supseteq D'\right\}$

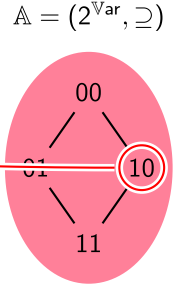


Illustrating Abstraction and Concretization Functions for Live Variables Analysis

Concretization Function γ

$\gamma(D) = \left\{ D' \in 2^{\mathbb{V}\text{ar}} \mid D \supseteq D' \right\}$

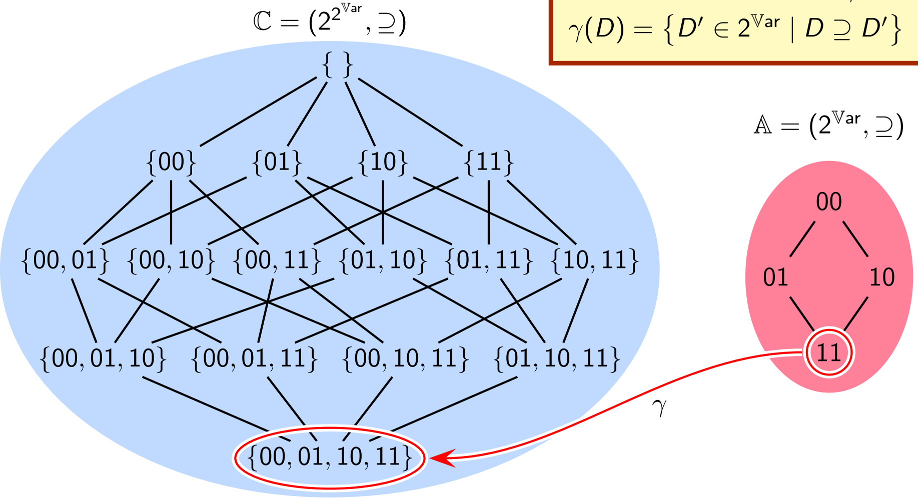




Illustrating Abstraction and Concretization Functions for Live Variables Analysis

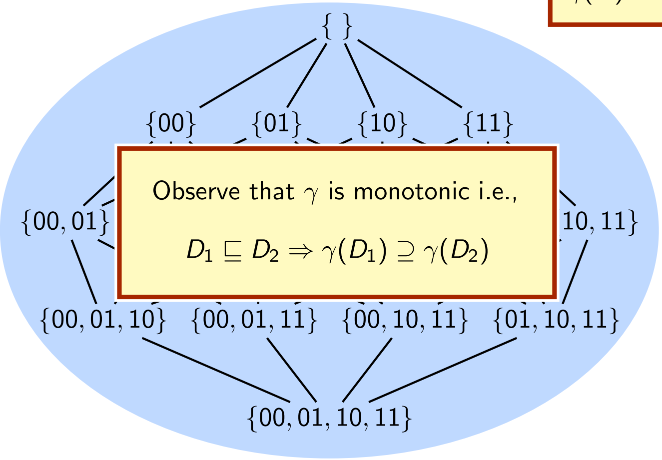
Concretization Function γ

$\gamma(D) = \left\{D' \in 2^{\mathbb{V}\mathsf{ar}} \mid D \supseteq D'\right\}$



Illustrating Abstraction and Concretization Functions for Live Variables Analysis

$\mathbb{C} = (2^{2^{\mathbb{V}\text{ar}}}, \supseteq)$



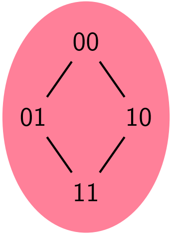
Observe that γ is monotonic i.e.,

$$D_1 \sqsubseteq D_2 \Rightarrow \gamma(D_1) \supseteq \gamma(D_2)$$

Concretization Function γ

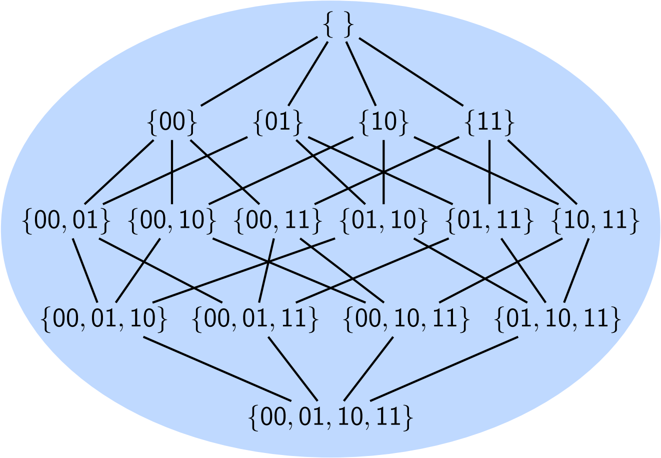
$$\gamma(D) = \{D' \in 2^{\mathbb{V}\text{ar}} \mid D \supseteq D'\}$$

$\mathbb{A} = (2^{\mathbb{V}\text{ar}}, \supseteq)$

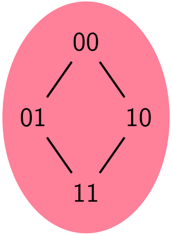


The Role of $\top$ Value in Live Variables Analysis

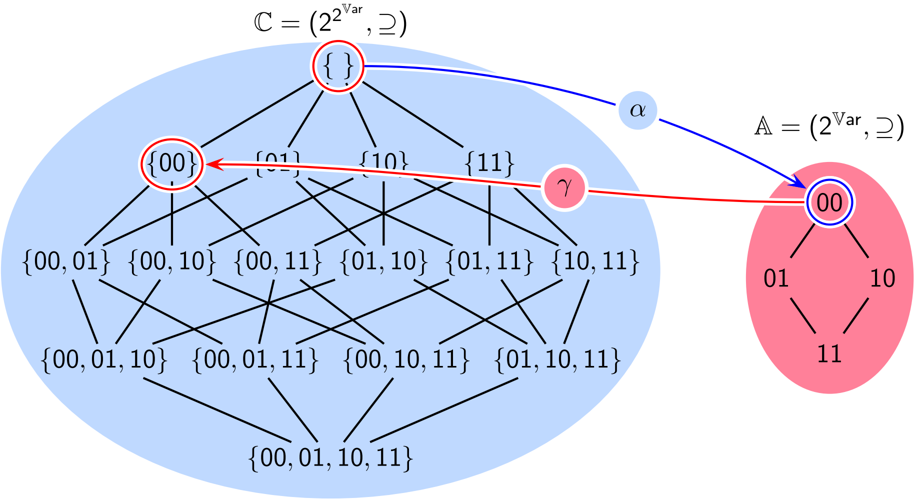
$\mathbb{C} = (2^{2^{\mathbb{V}\mathsf{ar}}}, \supseteq)$



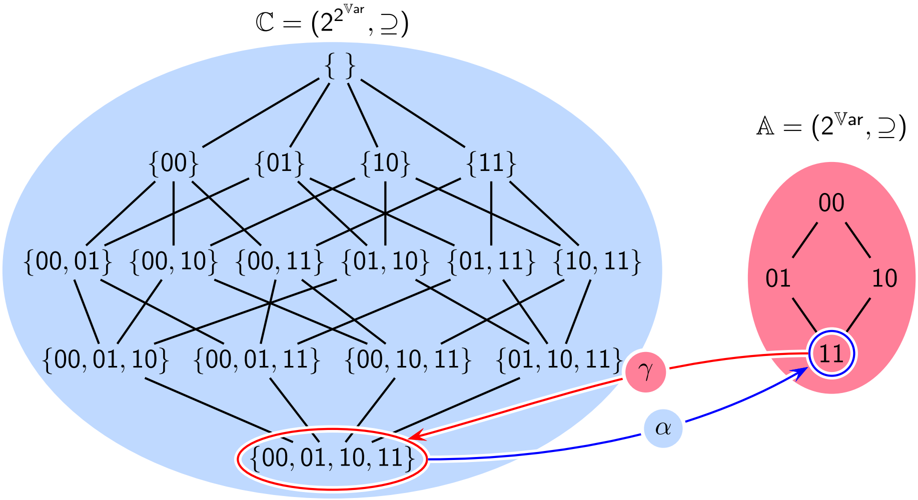
$\mathbb{A} = (2^{\mathbb{V}\mathsf{ar}}, \supseteq)$



The Role of $\top$ Value in Live Variables Analysis

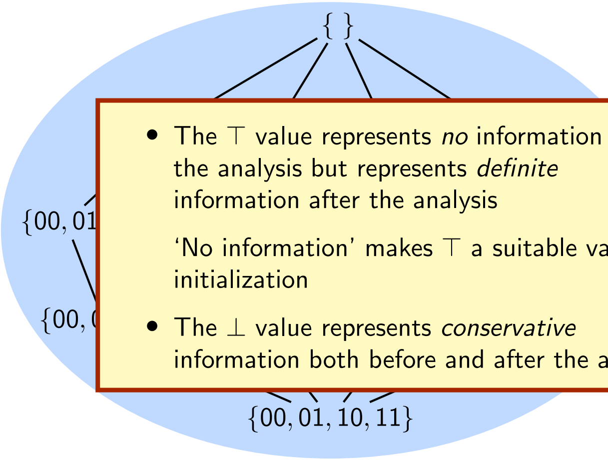


The Role of $\top$ Value in Live Variables Analysis

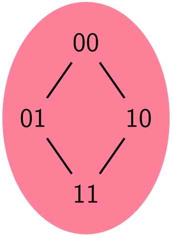


The Role of $\top$ Value in Live Variables Analysis

$\mathbb{C} = (2^{2^{\mathbb{V}\mathsf{ar}}}, \supseteq)$



$\mathbb{A} = (2^{\mathbb{V}\mathsf{ar}}, \supseteq)$



- The $\top$ value represents *no* information before the analysis but represents *definite* information after the analysis

  'No information' makes $\top$ a suitable value for initialization

- The $\perp$ value represents *conservative* information both before and after the analysis

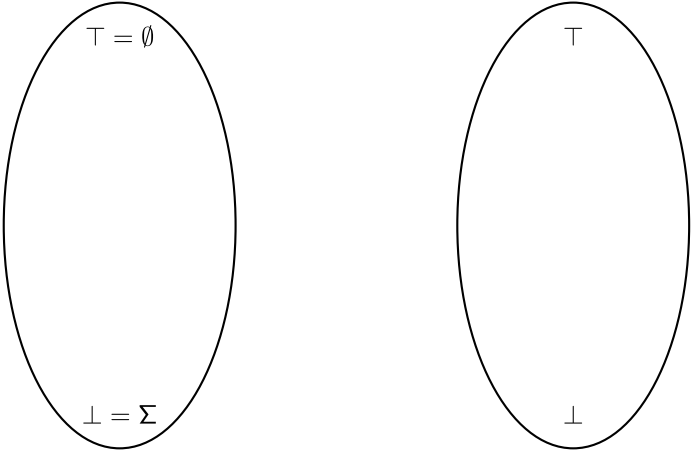
Galois Connection for Soundness of Static Analysis

Collecting Semantics

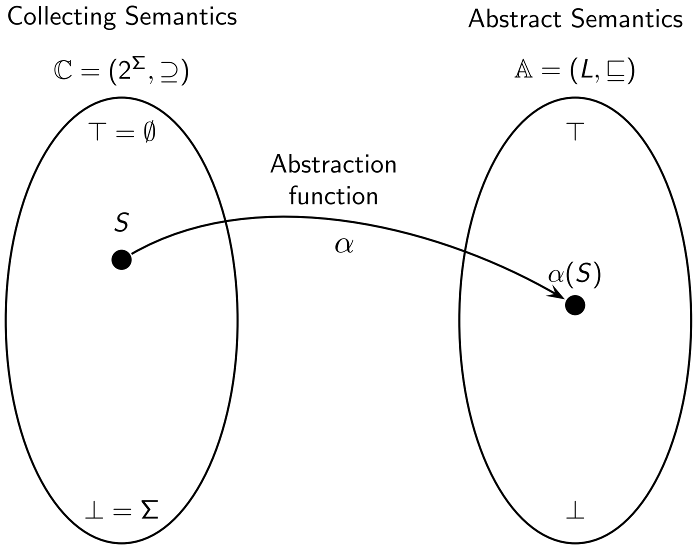
$\mathbb{C} = (2^{\Sigma}, \supseteq)$

Abstract Semantics

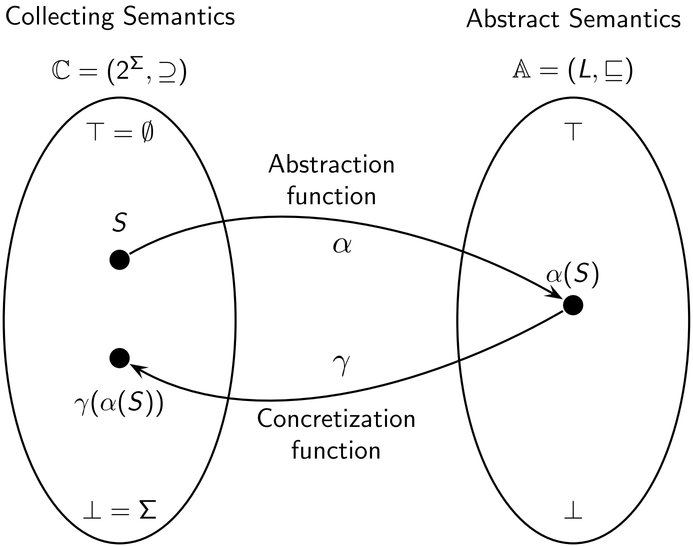
$\mathbb{A} = (L, \sqsubseteq)$



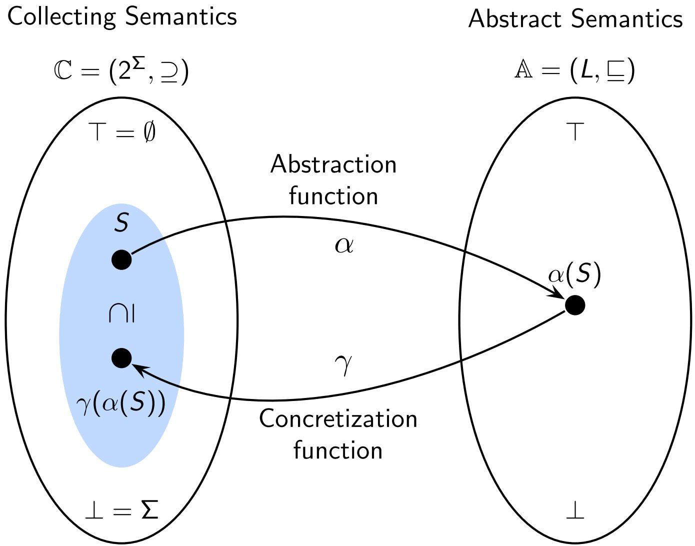
Galois Connection for Soundness of Static Analysis



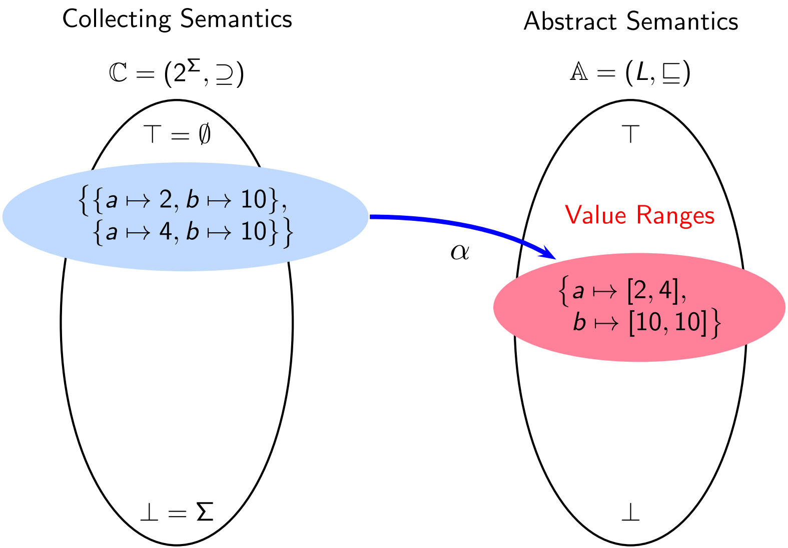
Galois Connection for Soundness of Static Analysis



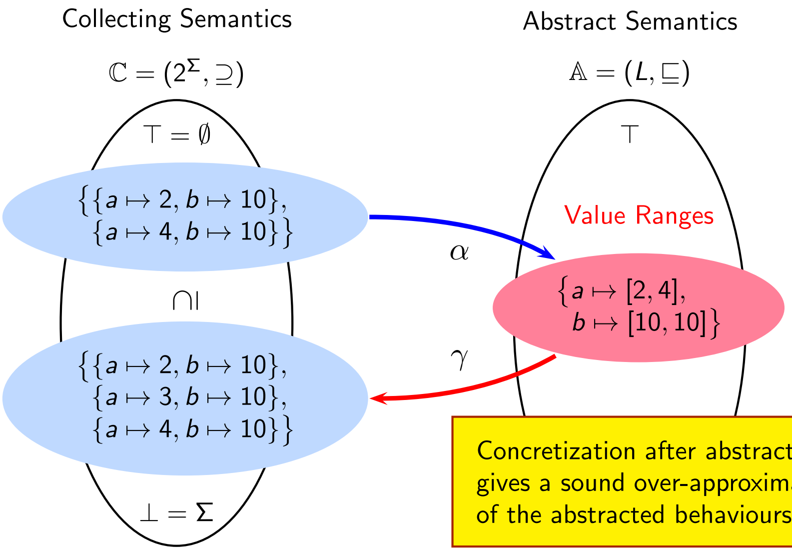
Galois Connection for Soundness of Static Analysis



Galois Connection for Soundness of Static Analysis

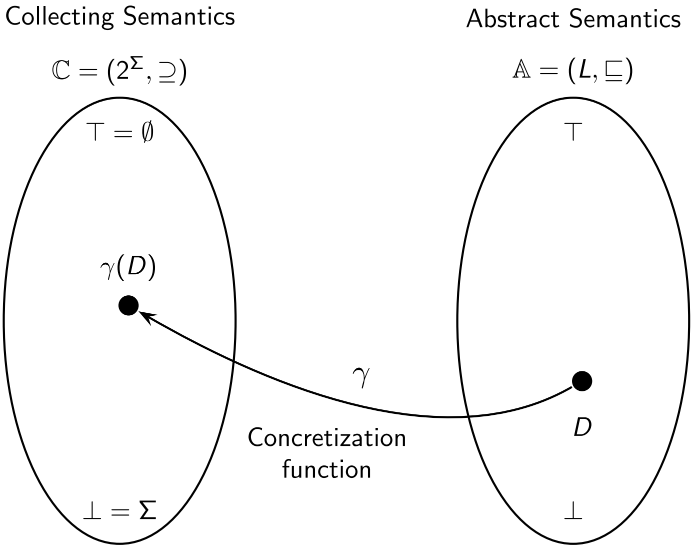


Galois Connection for Soundness of Static Analysis

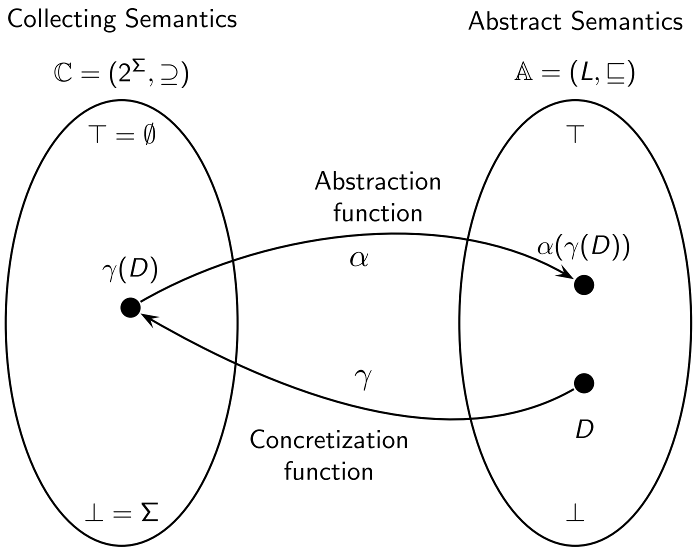


Concretization after abstraction gives a sound over-approximation of the abstracted behaviours

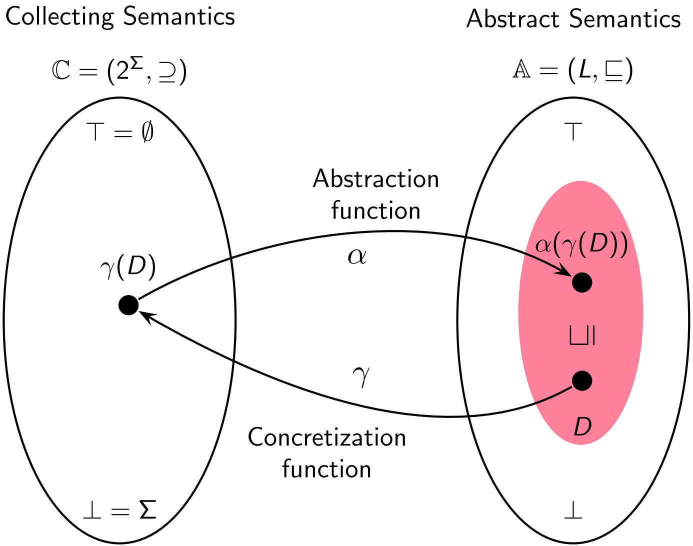
Galois Connection for Soundness of Static Analysis



Galois Connection for Soundness of Static Analysis



Galois Connection for Soundness of Static Analysis



IIT Bombay
cs618: Program Analysis

Topic:

Theoretical Abstractions

Section:

More General Setting

Data Flow Values

Flow Functions

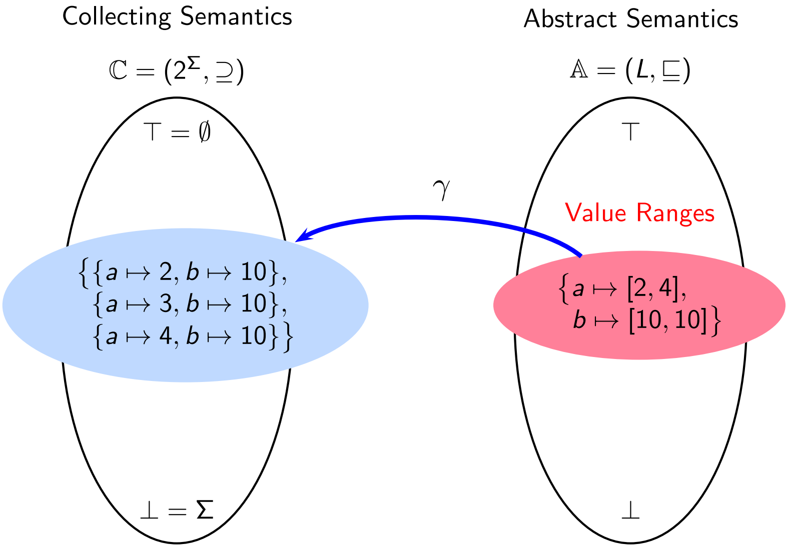
Solutions

Algorithms

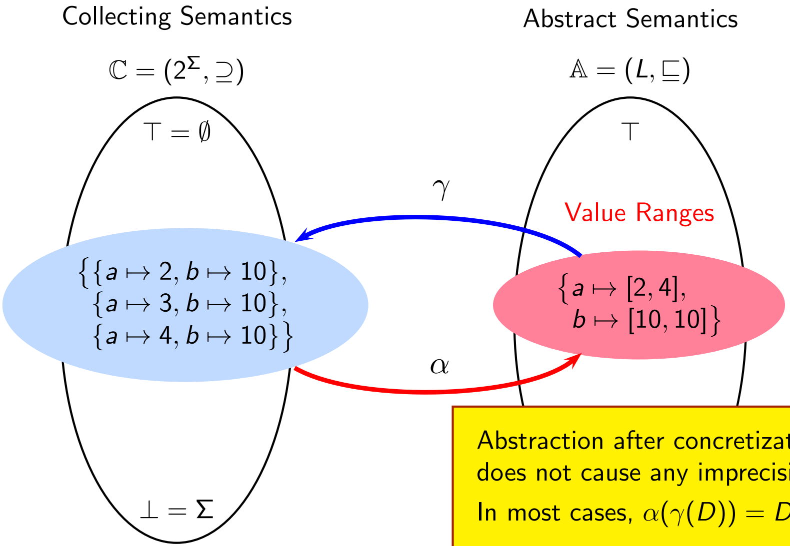
Modelling General Flows

Extra Material

Galois Connection for Soundness of Static Analysis



Galois Connection for Soundness of Static Analysis



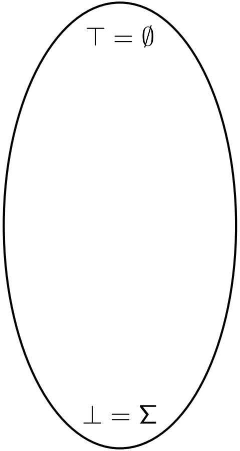
Abstraction after concretization does not cause any imprecision

In most cases, $\alpha(\gamma(D)) = D$

Galois Connection for Soundness of Static Analysis

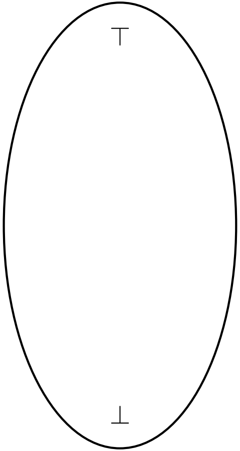
Collecting Semantics

$\mathbb{C} = (2^{\Sigma}, \supseteq)$



Abstract Semantics

$\mathbb{A} = (L, \sqsubseteq)$



Galois Connection for Soundness of Static Analysis

Collecting Semantics

$$\mathbb{C} = (2^{\Sigma}, \supseteq)$$

Abstract Semantics

$$\mathbb{A} = (L, \sqsubseteq)$$

For ensuring soundness, design monotone functions α and γ satisfying the Galois Connection

$$\gamma(\alpha(S)) \supseteq S \qquad \text{and} \qquad \alpha(\gamma(D)) \sqsupseteq D$$

In most cases, Galois Insertion suffices

$$\gamma(\alpha(S)) \supseteq S \qquad \text{and} \qquad \alpha(\gamma(D)) = D$$

Besides, it is sufficient to define one of α or γ and the other function can be derived from the defined function

We will define a formal soundness criterion when we talk about solutions of data flow analysis

Galois Connection for Soundness of Static Analysis

Collecting Semantics

$\mathbb{C} = (2^{\Sigma}, \supseteq)$

Abstract Semantics

$\mathbb{A} = (L, \sqsubseteq)$

The following properties must hold if we want a Galois connection:

$$\gamma(\bot) = \Sigma$$
$$\alpha(\emptyset) = \top$$

Intuitions:

- $\top$ indicates absence of information (nothing is known)
  Definite information is assumed as a hypothesis
- $\bot$ is the most conservative information (everything is possible)
  The hypothesis of definite information is falsified

Suitability of a Lattice for Representing Data Flow Values (1)

$\sqsubseteq$ captures valid approximations for soundness

- When $x \sqsubseteq y$, we say that x is *weaker than* y
- The data flow information represented by x captures all run time behaviours represented by the data flow information represented by y because

$$x \sqsubseteq y \;\Rightarrow\; \gamma(x) \supseteq \gamma(y)$$

- When $x \sqsubseteq y$, x is more conservative than y for soundness

  $\Rightarrow$ x can be safely used in place y

- However, since it may capture more behaviours, it may be imprecise

Suitability of a Lattice for Representing Data Flow Values (2)

$x \sqcap y$ 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 represented by x and y

Suitability of a Lattice for Representing Data Flow Values (2)

$x \sqcap y$ 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 represented by x and y

The 'g' of glb facilitates precision and 'lb' of glb ensures soundness

Properties of the Partial Order Relation

Reflexive $x \sqsubseteq x$

Transitive $x \sqsubseteq y, y \sqsubseteq z \Rightarrow x \sqsubseteq z$

Antisymmetric $x \sqsubseteq y, y \sqsubseteq x \Leftrightarrow x = y$

Properties of the 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

Properties of the Meet Operation

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

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

  Idempotent $x \sqcap x = x$

Properties of the Meet Operation

- 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

Properties of the Meet Operation

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 for cyclic dependences facilitates computation of the largest safe approximation

The Final Recipe for the Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

The Final Recipe for the Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

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

The Final Recipe for the Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

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

  What guarantees the presence of $\perp$?

The Final Recipe for the Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

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

  What guarantees the presence of $\perp$?

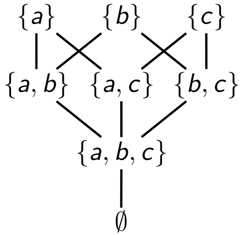
- $\top$ may not exist. Can be added artificially

The Final Recipe for the Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

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

> - In this lattice, glb takes the union of elements except when one of them is an empty set
>
> $$X \sqcap Y = \begin{cases} X \cup Y & X \neq \emptyset \wedge Y \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$
>
> - There is no natural $\top$ that can be expressed as a subset of $\{a, b, c\}$



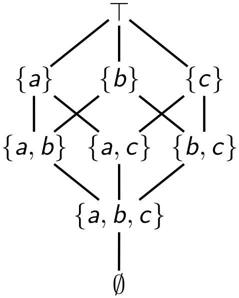
- $\top$ may not exist. Can be added artificially

The Final Recipe for the Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

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

> - In this lattice, glb takes the union of elements except when one of them is an empty set
>
> $$X \sqcap Y = \begin{cases} X \cup Y & X \neq \emptyset \wedge Y \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$
>
> - There is no natural $\top$ that can be expressed as a subset of $\{a, b, c\}$



- $\top$ may not exist. Can be added artificially
  - This makes the lattice complete

The Final Recipe for the Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

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

  What guarantees the presence of $\perp$?

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

- $\top$ may not exist. Can be added artificially
  - This makes the lattice complete

The Final Recipe for the Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

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

  What guarantees the presence of $\bot$?

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

- $\top$ may not exist. Can be added artificially
  - This makes the lattice complete

The Final Recipe for the Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

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

  What guarantees the presence of $\perp$?

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

- $\top$ may not exist. Can be added artificially
  - This makes the lattice complete

The Final Recipe for the Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

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

  What guarantees the presence of $\perp$?

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

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 $f : L \rightarrow L$ 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

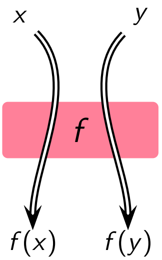
$$f(x) = \text{Gen} \cup (x - \text{Kill})$$

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

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

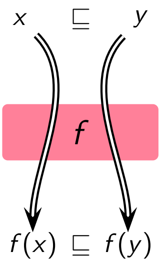
Monotonicity of Flow Functions

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



Monotonicity of Flow Functions

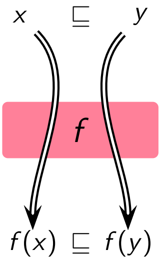
- Partial order is preserved: If x can be safely used in place of y then $f(x)$ can be safely used in place of $f(y)$



Monotonicity of Flow Functions

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

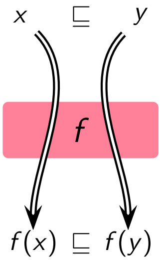
$$\forall x, y \in L, x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$$



Monotonicity of Flow Functions

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

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



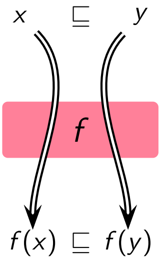
- Alternative definition

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

Monotonicity of Flow Functions

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

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



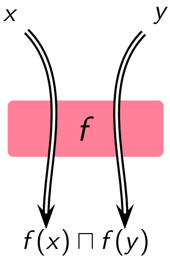
- Alternative definition

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

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

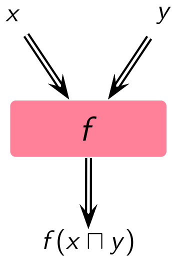
Distributivity of Flow Functions

- Merging distributes over function application



Distributivity of Flow Functions

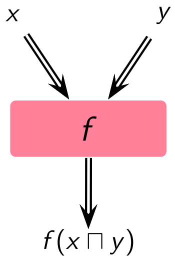
- Merging distributes over function application



Distributivity of Flow Functions

- Merging distributes over function application

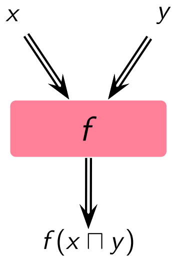
$$\forall x, y \in L, f(x \sqcap y) = f(x) \sqcap f(y)$$



Distributivity of Flow Functions

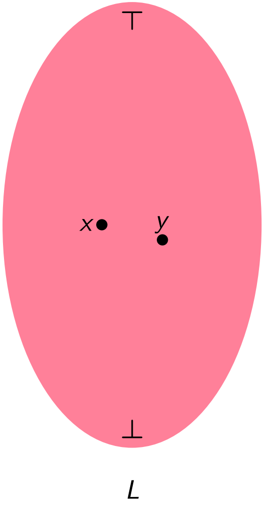
- Merging distributes over function application

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

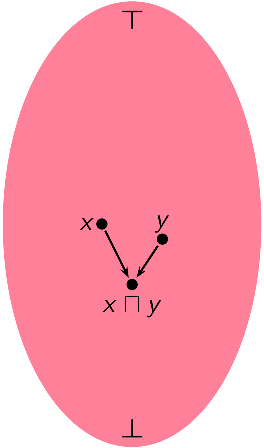


- Merging at intermediate points in shared segments of paths does not lead to imprecision

Monotonicity and Distributivity

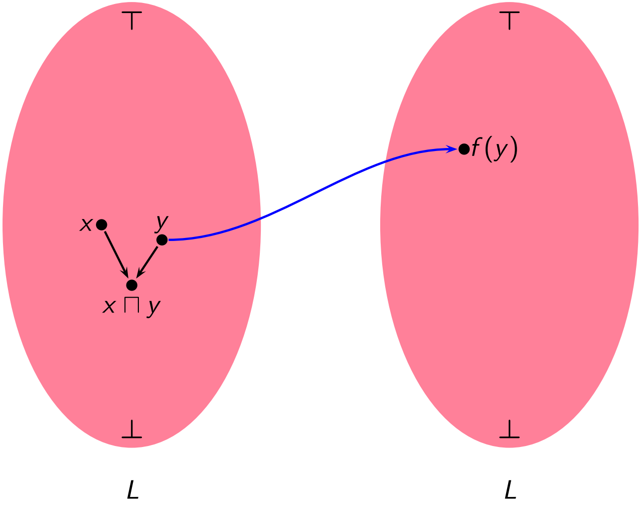


Monotonicity and Distributivity

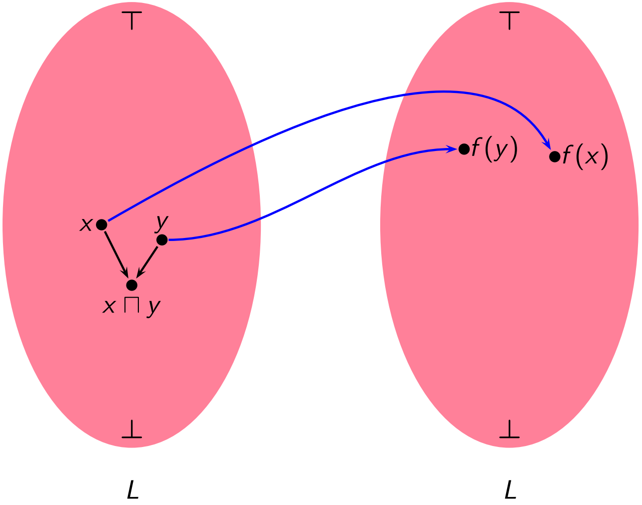


L

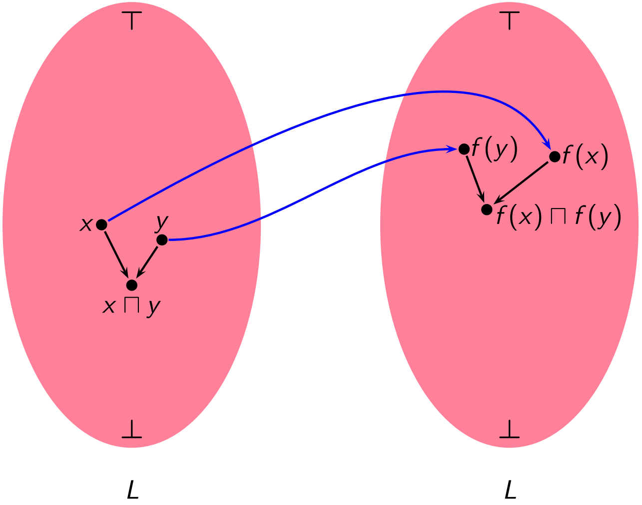
Monotonicity and Distributivity



Monotonicity and Distributivity

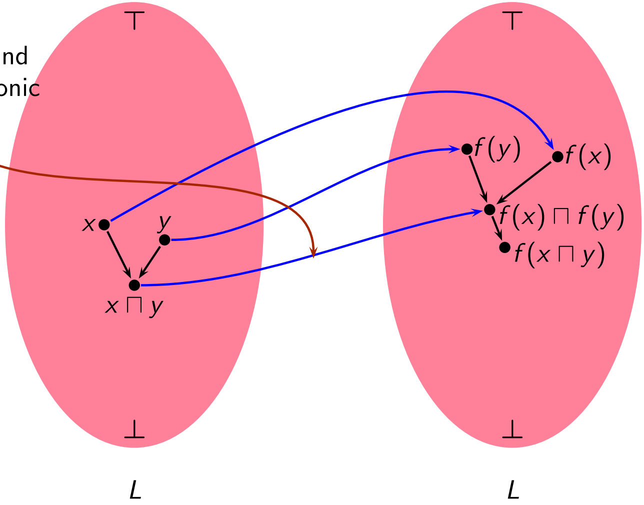


Monotonicity and Distributivity

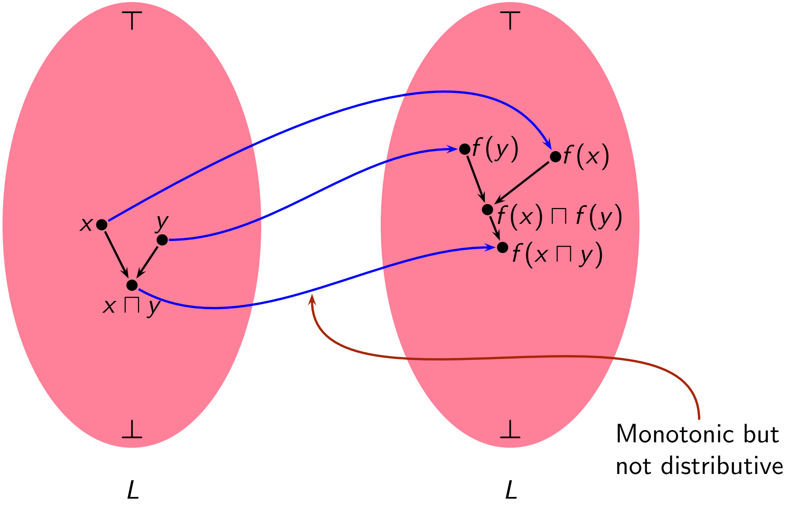


Monotonicity and Distributivity

Distributive and hence monotonic



Monotonicity and Distributivity



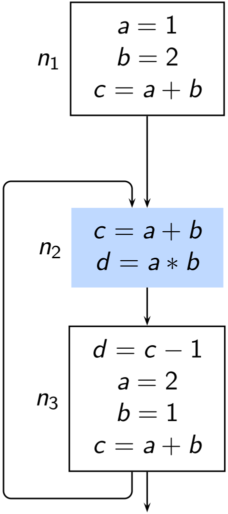
Distributivity of Bit Vector Frameworks

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

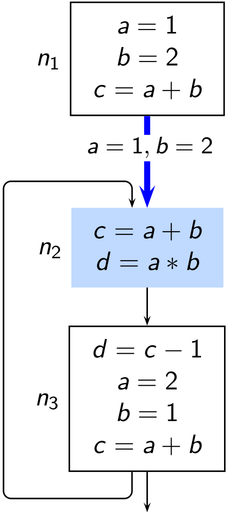
$$
\begin{aligned}
f(x \cup y) &= \text{Gen} \cup ((x \cup y) - \text{Kill}) \\
&= \text{Gen} \cup ((x - \text{Kill}) \cup (y - \text{Kill})) \\
&= (\text{Gen} \cup (x - \text{Kill}) \cup \text{Gen} \cup (y - \text{Kill})) \\
&= f(x) \cup f(y)
\end{aligned}
$$

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

Non-Distributivity of Constant Propagation

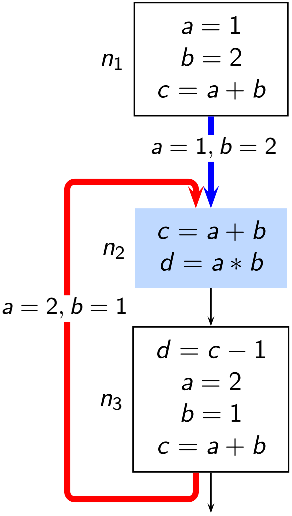


Non-Distributivity of Constant Propagation



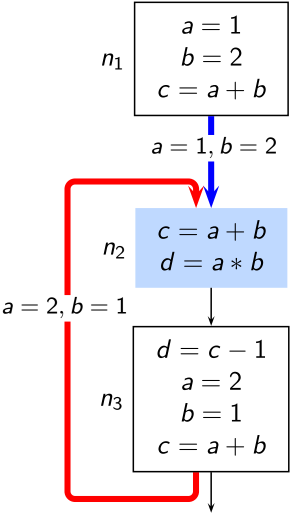
- $x = \langle 1, 2, 3, \widehat{\top} \rangle$ (Along $Out_{n_1} \rightarrow In_{n_2}$)

Non-Distributivity of Constant Propagation



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

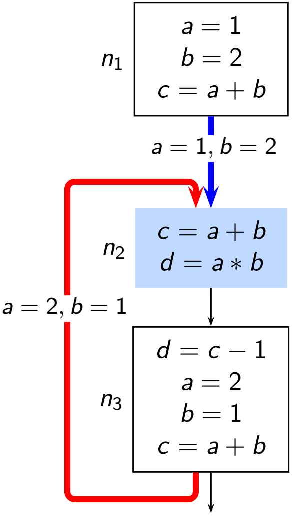
Non-Distributivity of Constant Propagation



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

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

Non-Distributivity of Constant Propagation



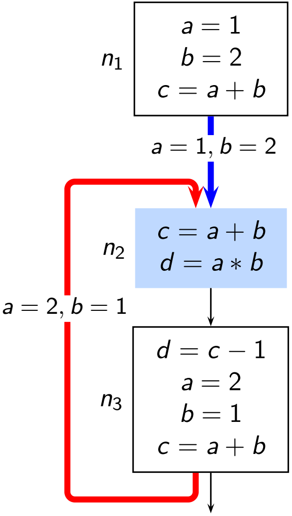
- $x = \langle 1, 2, 3, \widehat{\top} \rangle$ (Along $Out_{n_1} \rightarrow In_{n_2}$)
- $y = \langle 2, 1, 3, 2 \rangle$ (Along $Out_{n_3} \rightarrow In_{n_2}$)
- Function application for block n_2 before merging

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

- Function application for block n_2 after merging

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

Non-Distributivity of Constant Propagation



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

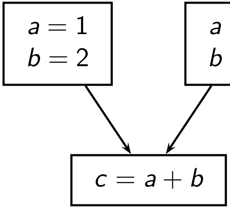
$$\begin{aligned} f(x) \sqcap f(y) &= f(\langle 1, 2, 3, \widehat{\top} \rangle) \sqcap f(\langle 2, 1, 3, 2 \rangle) \\ &= \langle 1, 2, 3, 2 \rangle \sqcap \langle 2, 1, 3, 2 \rangle \\ &= \langle \widehat{\bot}, \widehat{\bot}, 3, 2 \rangle \end{aligned}$$

- Function application for block n_2 after merging

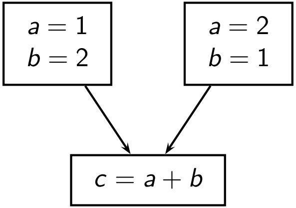
$$\begin{aligned} f(x \sqcap y) &= f(\langle 1, 2, 3, \widehat{\top} \rangle \sqcap \langle 2, 1, 3, 2 \rangle) \\ &= f(\langle \widehat{\bot}, \widehat{\bot}, 3, 2 \rangle) \\ &= \langle \widehat{\bot}, \widehat{\bot}, \widehat{\bot}, \widehat{\bot} \rangle \end{aligned}$$

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

Why is Constant Propagation Non-Distributive?



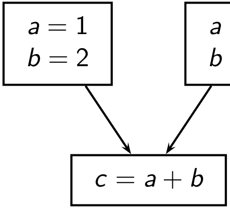
Why is Constant Propagation Non-Distributive?



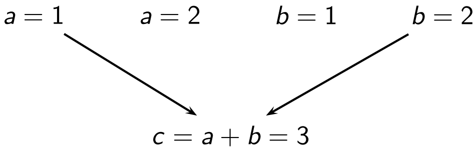
Possible combinations due to merging

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

Why is Constant Propagation Non-Distributive?

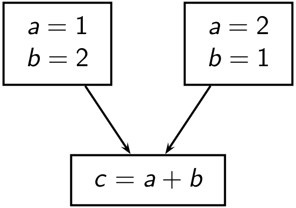


Possible combinations due to merging



- Correct combination

Why is Constant Propagation Non-Distributive?



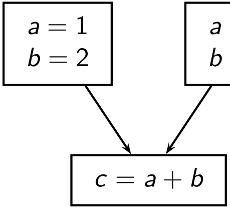
Possible combinations due to merging

$a = 1 \qquad a = 2 \qquad b = 1 \qquad b = 2$

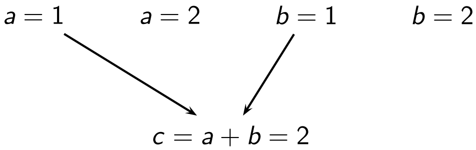
$c = a + b = 3$

- Correct combination

Why is Constant Propagation Non-Distributive?

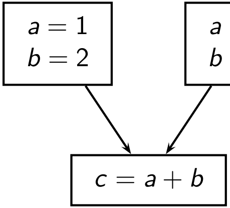


Possible combinations due to merging

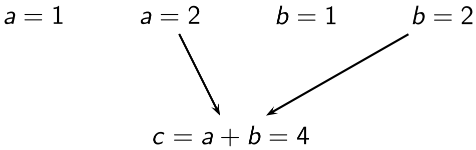


- Wrong combination
- Mutually exclusive information
- No execution path along which this information holds

Why is Constant Propagation Non-Distributive?



Possible combinations due to merging



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

Solutions of Data Flow Analysis

Solutions of Data Flow Analysis: An Outline of Our Discussion

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

Solutions of Data Flow Analysis

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

  Given

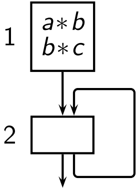
  - A set of flow functions, a lattice, and merge operation
  - A program flow graph with a mapping from nodes to flow functions

  Find out

  - An assignment A which is as “large” as possible and is sound

An Example For Available Expressions Analysis

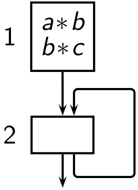
Program



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

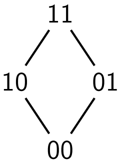
An Example For Available Expressions Analysis

Program



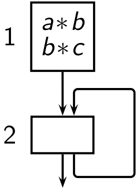
Some Assignments							
	A_0	A_1	A_2	A_3	A_4	A_5	A_6
In_1	11	00	00	00	00	00	00
Out_1	11	11	00	11	11	11	11
In_2	11	11	00	00	10	01	01
Out_2	11	11	00	00	10	01	10

Lattice L of data flow values at a node



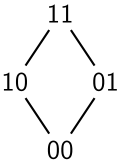
An Example For Available Expressions Analysis

Program

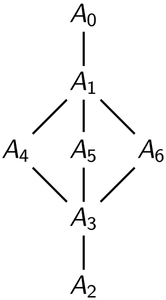


Some Assignments							
	A_0	A_1	A_2	A_3	A_4	A_5	A_6
In_1	11	00	00	00	00	00	00
Out_1	11	11	00	11	11	11	11
In_2	11	11	00	00	10	01	01
Out_2	11	11	00	00	10	01	10

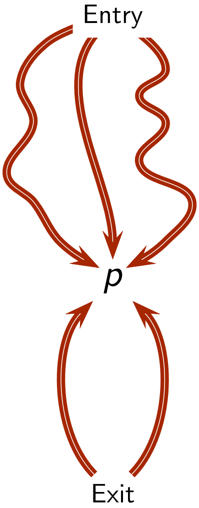
Lattice L of data flow values at a node



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



Meet Over Paths (MoP) Assignment

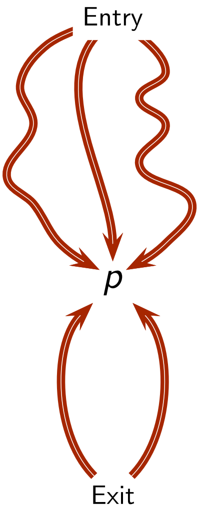


- The largest safe approximation of the abstract information reaching a program point along all control flow paths

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

  - f_ρ represents the compositions of flow functions along ρ
  - BI is the boundary information
  - All control flow paths are considered potentially executable by ignoring the results of conditionals

Meet Over Paths (MoP) Assignment



- Alternatively,

$$MoP(p) \quad \sqsubseteq \quad \alpha(S(p))$$

where $S(p)$ is the set of states reaching p along all traces

Since all control flow paths are considered potentially executable, they over-aproximate all execution traces

- Any $Info(p) \sqsubseteq MoP(p)$ is safe because it over-approximates all behaviours in $S(p)$

$$Info(p) \sqsubseteq MoP(p) \;\; \Rightarrow \;\; \gamma(Info(p)) \supseteq \gamma(MoP(p))$$

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

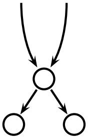
    If all paths need to be traversed $\Rightarrow$ Undecidability

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 $\Rightarrow$ 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



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 $\Rightarrow$ 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

Path based specification

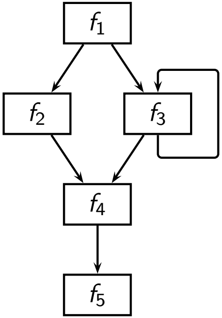
- Why not merge information at intermediate points?
  - Merging is safe but may lead to imprecision
  - Computes fixed point solutions of data flow equations

Edge based specifications

Computing MFP Vs. Computing MoP

Expression Tree for MFP

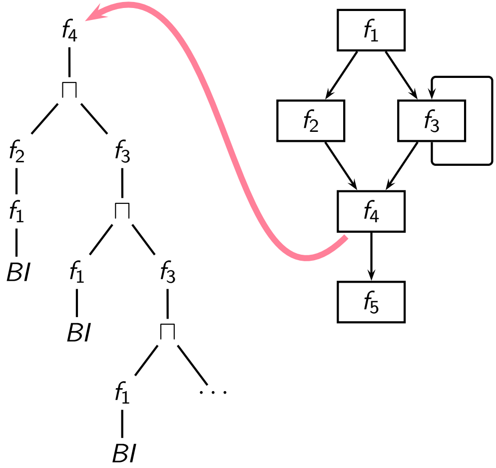
Program



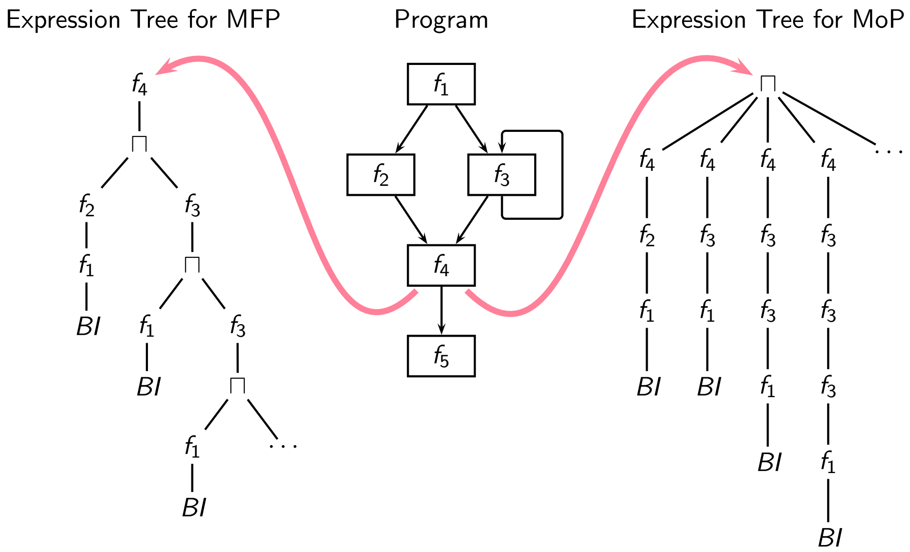
Expression Tree for MoP

Computing MFP Vs. Computing MoP

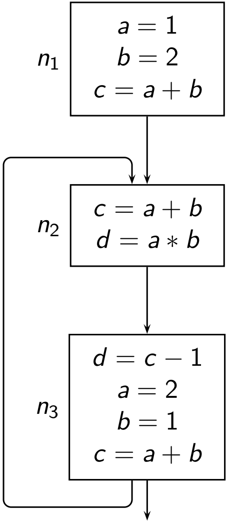
Expression Tree for MFP | Program | Expression Tree for MoP



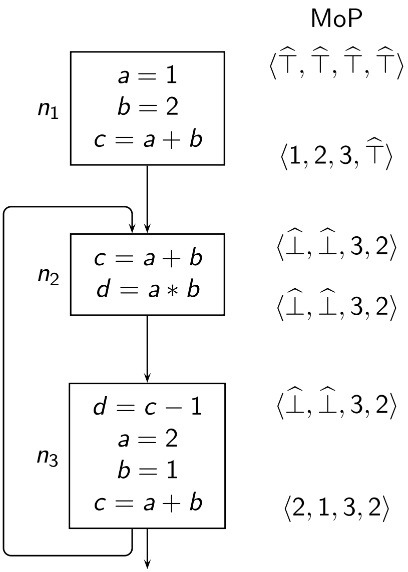
Computing MFP Vs. Computing MoP



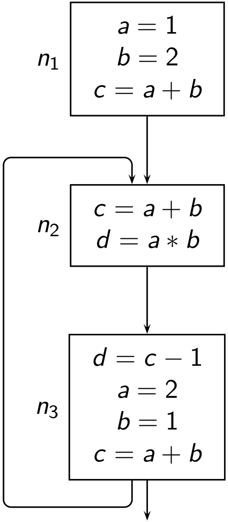
Assignments for Constant Propagation Example



Assignments for Constant Propagation Example



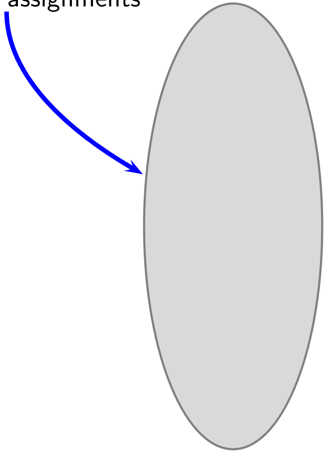
Assignments for Constant Propagation Example



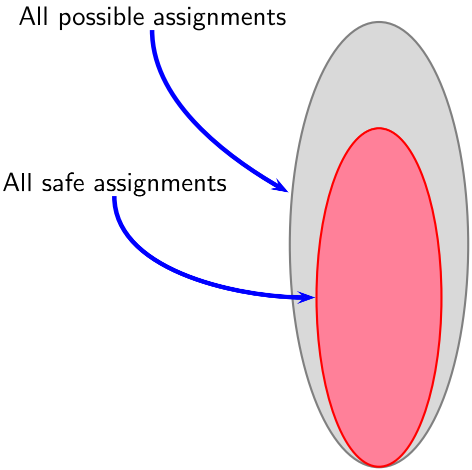
MoP	MFP
$\langle \widehat{\top}, \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle$	$\langle \widehat{\top}, \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle$
$\langle 1, 2, 3, \widehat{\top} \rangle$	$\langle 1, 2, 3, \widehat{\top} \rangle$
$\langle \widehat{\bot}, \widehat{\bot}, 3, 2 \rangle$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

Possible Assignments as Solutions of Data Flow Analyses

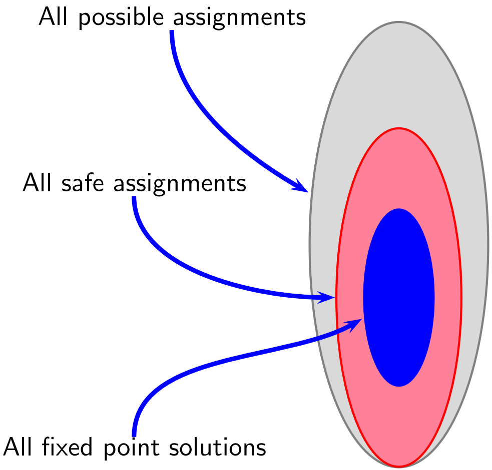
All possible assignments



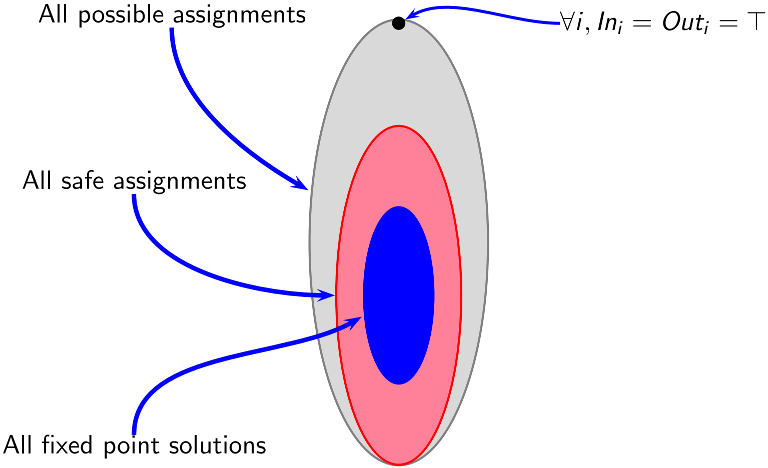
Possible Assignments as Solutions of Data Flow Analyses



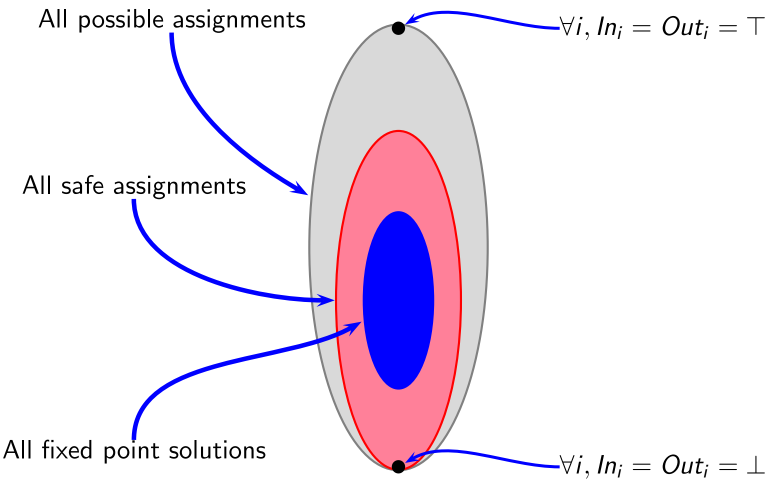
Possible Assignments as Solutions of Data Flow Analyses



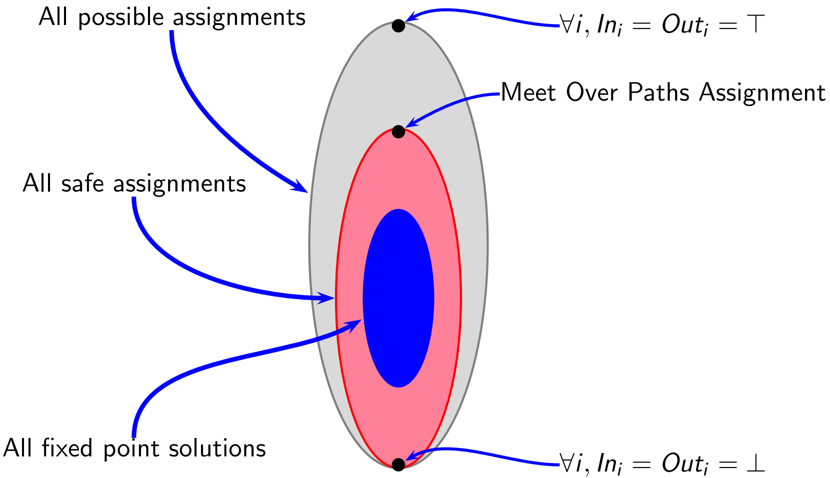
Possible Assignments as Solutions of Data Flow Analyses



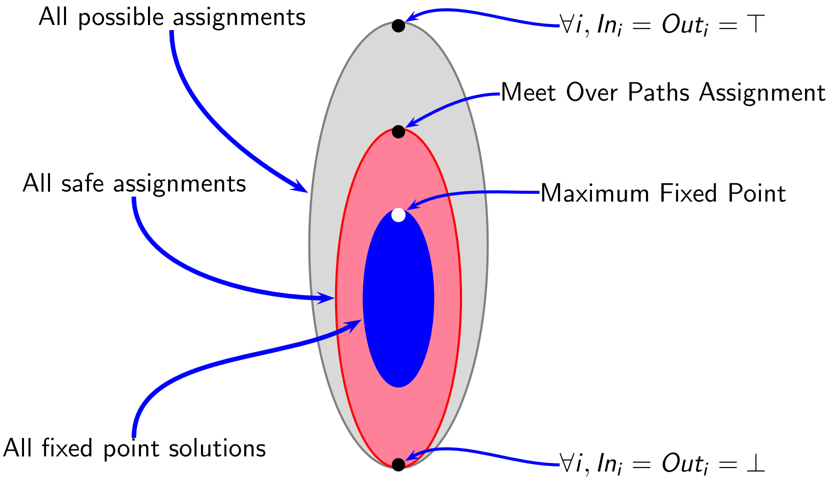
Possible Assignments as Solutions of Data Flow Analyses



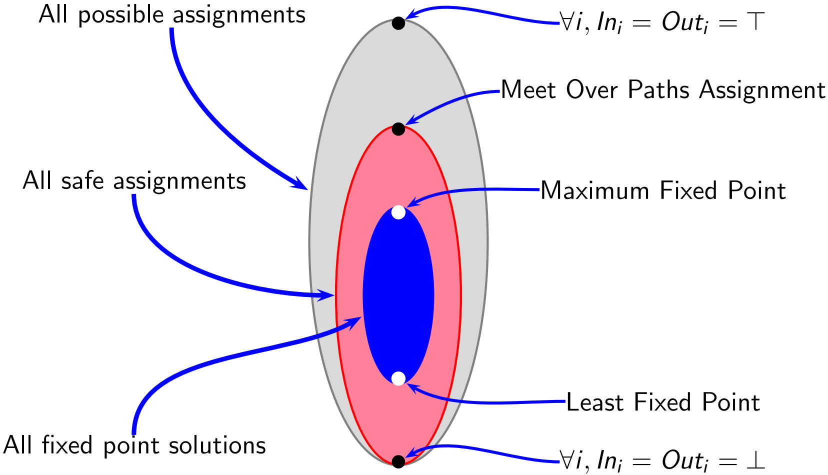
Possible Assignments as Solutions of Data Flow Analyses



Possible Assignments as Solutions of Data Flow Analyses



Possible Assignments as Solutions of Data Flow Analyses



An Instance of Available Expressions Analysis

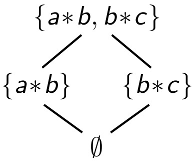
Lattice

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

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

An Instance of Available Expressions Analysis

Lattice

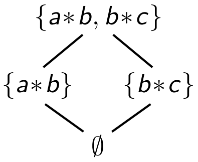


Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
$f_\top$	$\{a*b, b*c\}$	f_{id}	x
$f_\bot$	$\emptyset$	f_c	$x \cup \{a*b\}$
f_a	$\{a*b\}$	f_d	$x \cup \{b*c\}$
f_b	$\{b*c\}$	f_e	$x - \{a*b\}$
		f_f	$x - \{b*c\}$

- Is the lattice a meet semilattice?

An Instance of Available Expressions Analysis

Lattice

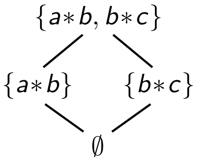


Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
$f_\top$	$\{a*b, b*c\}$	f_{id}	x
$f_\perp$	$\emptyset$	f_c	$x \cup \{a*b\}$
f_a	$\{a*b\}$	f_d	$x \cup \{b*c\}$
f_b	$\{b*c\}$	f_e	$x - \{a*b\}$
		f_f	$x - \{b*c\}$

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

An Instance of Available Expressions Analysis

Lattice

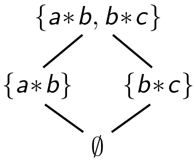


Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
$f_\top$	$\{a*b, b*c\}$	f_{id}	x
$f_\bot$	$\emptyset$	f_c	$x \cup \{a*b\}$
f_a	$\{a*b\}$	f_d	$x \cup \{b*c\}$
f_b	$\{b*c\}$	f_e	$x - \{a*b\}$
		f_f	$x - \{b*c\}$

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

An Instance of Available Expressions Analysis

Lattice

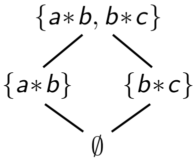


Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
$f_\top$	$\{a*b, b*c\}$	f_{id}	x
$f_\bot$	$\emptyset$	f_c	$x \cup \{a*b\}$
f_a	$\{a*b\}$	f_d	$x \cup \{b*c\}$
f_b	$\{b*c\}$	f_e	$x - \{a*b\}$
		f_f	$x - \{b*c\}$

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

An Instance of Available Expressions Analysis

Lattice

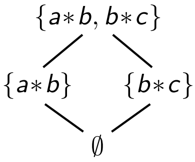


Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
$f_\top$	$\{a*b, b*c\}$	f_{id}	x
$f_\bot$	$\emptyset$	f_c	$x \cup \{a*b\}$
f_a	$\{a*b\}$	f_d	$x \cup \{b*c\}$
f_b	$\{b*c\}$	f_e	$x - \{a*b\}$
		f_f	$x - \{b*c\}$

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

An Instance of Available Expressions Analysis

Lattice

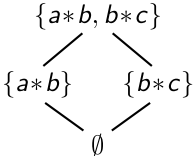


Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
$f_\top$	$\{a*b, b*c\}$	f_{id}	x
$f_\bot$	$\emptyset$	f_c	$x \cup \{a*b\}$
f_a	$\{a*b\}$	f_d	$x \cup \{b*c\}$
f_b	$\{b*c\}$	f_e	$x - \{a*b\}$
		f_f	$x - \{b*c\}$

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

An Instance of Available Expressions Analysis

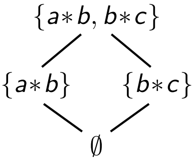
Lattice



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

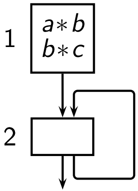
An Instance of Available Expressions Analysis

Lattice



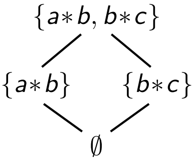
Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
$f_\top$	$\{a*b, b*c\}$	f_{id}	x
$f_\perp$	$\emptyset$	f_c	$x \cup \{a*b\}$
f_a	$\{a*b\}$	f_d	$x \cup \{b*c\}$
f_b	$\{b*c\}$	f_e	$x - \{a*b\}$
		f_f	$x - \{b*c\}$

Program



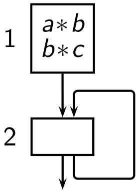
An Instance of Available Expressions Analysis

Lattice



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

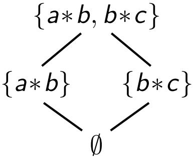
Program



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

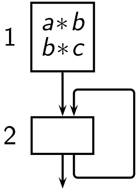
An Instance of Available Expressions Analysis

Lattice



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

Program



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

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

An Instance of Available Expressions Analysis

Lattice

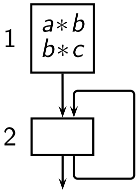
$\{a*b, b*c\}$

$\{a*b\}$

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

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

Program

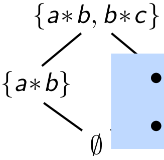


Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

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

An Instance of Available Expressions Analysis

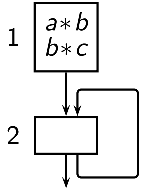
Lattice



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

- Not a fixed point assignment
- Sound assignment

Program



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

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

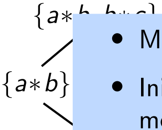
An Instance of Available Expressions Analysis

IIT Bombay
cs618: Program Analysis

Topic:
Theoretical Abstractions

Section:

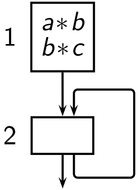
Lattice



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

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

Program

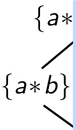


Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

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

An Instance of Available Expressions Analysis

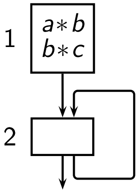
Lattice



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

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

Program

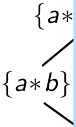


Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

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

An Instance of Available Expressions Analysis

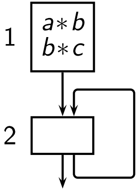
Lattice



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

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

Program



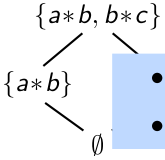
Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

Some Possible Assignments

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

An Instance of Available Expressions Analysis

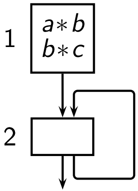
Lattice



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

Program

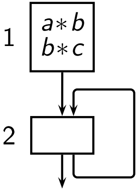


Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

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

Lattice of Assignments for Available Expressions Analysis

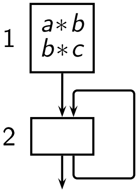
Program



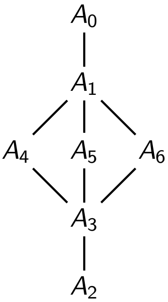
Some Assignments							
	A_0	A_1	A_2	A_3	A_4	A_5	A_6
In_1	11	00	00	00	00	00	00
Out_1	11	11	00	11	11	11	11
In_2	11	11	00	00	10	01	01
Out_2	11	11	00	00	10	01	10

Lattice of Assignments for Available Expressions Analysis

Program



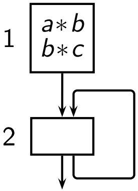
Some Assignments							
	A_0	A_1	A_2	A_3	A_4	A_5	A_6
In_1	11	00	00	00	00	00	00
Out_1	11	11	00	11	11	11	11
In_2	11	11	00	00	10	01	01
Out_2	11	11	00	00	10	01	10



- Lattice $L \times L \times L \times L$ for all assignments
- Four possible values at each of the four program points $\Rightarrow 4 \times 4 \times 4 \times 4 = 256$ possible assignments
- We have shown only a few of them

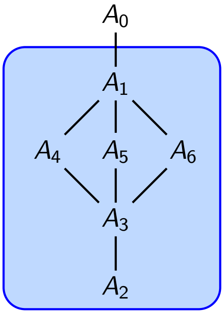
Lattice of Assignments for Available Expressions Analysis

Program



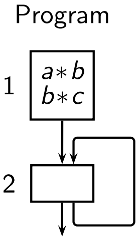
Some Assignments							
	A_0	A_1	A_2	A_3	A_4	A_5	A_6
In_1	11	00	00	00	00	00	00
Out_1	11	11	00	11	11	11	11
In_2	11	11	00	00	10	01	01
Out_2	11	11	00	00	10	01	10

- Lattice $L \times L \times L \times L$ for all assignments
- Four possible values at each of the four program points $\Rightarrow 4 \times 4 \times 4 \times 4 = 256$ possible assignments
- We have shown only a few of them



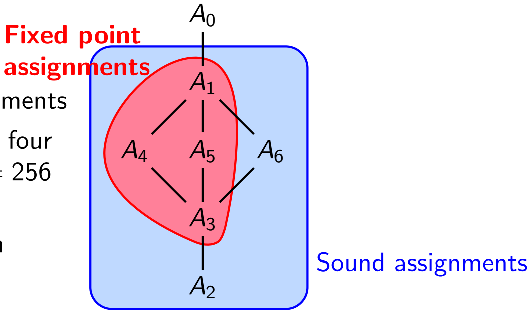
Sound assignments

Lattice of Assignments for Available Expressions Analysis



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

- Lattice $L \times L \times L \times L$ for all assignments
- Four possible values at each of the four program points $\Rightarrow 4 \times 4 \times 4 \times 4 = 256$ possible assignments
- We have shown only a few of them



Existence of an MoP Assignment (1)

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

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

Existence of an MoP Assignment (2)

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

- If an infinite number of paths reach p then,

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

Existence of an MoP Assignment (2)

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

- If an infinite number of paths reach p then,

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

Existence of an MoP Assignment (2)

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

- If an infinite number of paths reach p then,

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

- Every meet results in a weaker value

IIT Bombay
cs618: Program Analysis

Topic:

Theoretical Abstractions

Section:

More General Setting

Data Flow Values

Flow Functions

Solutions

Algorithms

Modelling General Flows

Extra Material

Existence of an MoP Assignment (2)

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

- If an infinite number of paths reach p then,

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

- Every meet results in a weaker value

Existence of an MoP Assignment (2)

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

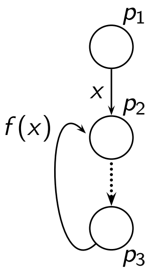
- If an infinite number of paths reach p then,

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

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

Computability of MoP

Does existence of MoP imply it is computable?



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

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

MoP Computation is Undecidable

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

- Reducing MPCP (Modified Post's Correspondence Problem) to constant propagation
- MPCP is known to be undecidable
- If an algorithm exists for detecting all constants
  $\Rightarrow$ MPCP would be decidable
- Since MPCP is undecidable
  $\Rightarrow$ There does not exist an algorithm for detecting all constants
  $\Rightarrow$ Static analysis is undecidable

Is MFP Always Computable?

MFP assignment may not be computable

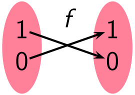
- if the flow functions are non-monotonic, or
- if some strictly descending chain is not finite

Computability of MFP

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

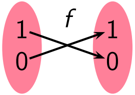
Computability of MFP

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



Computability of MFP

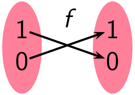
- If f is not monotonic, the computation may not converge



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

Computability of MFP

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

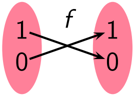


x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\ldots$
1	0	1	0	1	$\ldots$

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

Computability of MFP

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



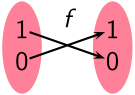
x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\ldots$
1	0	1	0	1	$\ldots$

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

- Computing MFP iteratively

Computability of MFP

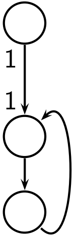
- If f is not monotonic, the computation may not converge



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

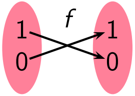
$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots \;=\; 0$$

- Computing MFP iteratively



Computability of MFP

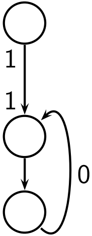
- If f is not monotonic, the computation may not converge



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

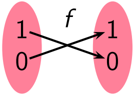
$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots \; = \; 0$$

- Computing MFP iteratively



Computability of MFP

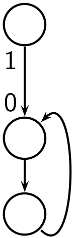
- If f is not monotonic, the computation may not converge



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

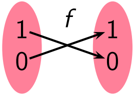
$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \dots \; = \; 0$$

- Computing MFP iteratively



Computability of MFP

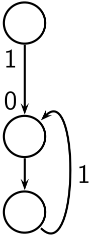
- If f is not monotonic, the computation may not converge



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

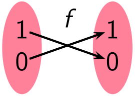
$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots \; = \; 0$$

- Computing MFP iteratively



Computability of MFP

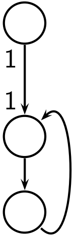
- If f is not monotonic, the computation may not converge



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

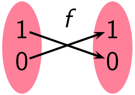
$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \dots = 0$$

- Computing MFP iteratively



Computability of MFP

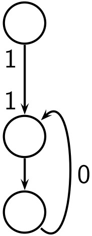
- If f is not monotonic, the computation may not converge



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

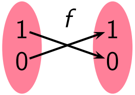
$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots \;=\; 0$$

- Computing MFP iteratively



Computability of MFP

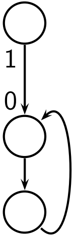
- If f is not monotonic, the computation may not converge



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

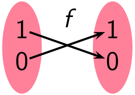
$$MoP = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots \; = \; 0$$

- Computing MFP iteratively



Computability of MFP

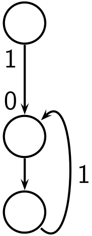
- If f is not monotonic, the computation may not converge



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

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

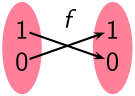
- Computing MFP iteratively



MFP does not exist and is not computable

Computability of MFP

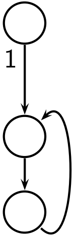
- If f is not monotonic, the computation may not converge



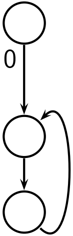
x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\ldots$
1	0	1	0	1	$\ldots$

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

- Computing MFP iteratively

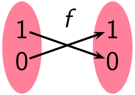


MFP does not exist and is not computable



Computability of MFP

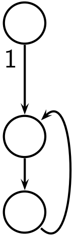
- If f is not monotonic, the computation may not converge



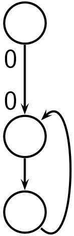
x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\ldots$
1	0	1	0	1	$\ldots$

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

- Computing MFP iteratively

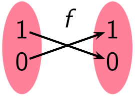


MFP does not exist and is not computable



Computability of MFP

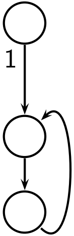
- If f is not monotonic, the computation may not converge



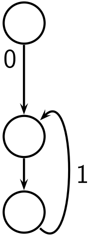
x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\dots$
1	0	1	0	1	$\dots$

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

- Computing MFP iteratively

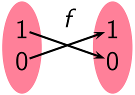


MFP does not exist and is not computable



Computability of MFP

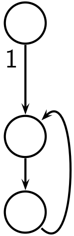
- If f is not monotonic, the computation may not converge



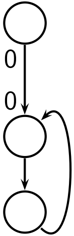
x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\ldots$
1	0	1	0	1	$\ldots$

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

- Computing MFP iteratively

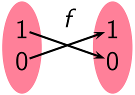


MFP does not exist and is not computable



Computability of MFP

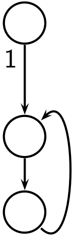
- If f is not monotonic, the computation may not converge



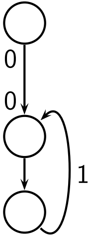
x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\ldots$
1	0	1	0	1	$\ldots$

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

- Computing MFP iteratively

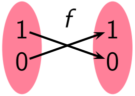


MFP does not exist and is not computable



Computability of MFP

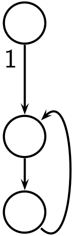
- If f is not monotonic, the computation may not converge



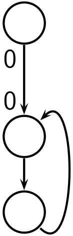
x	$f(x)$	$f^2(x)$	$f^3(x)$	$f^4(x)$	$\ldots$
1	0	1	0	1	$\ldots$

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

- Computing MFP iteratively

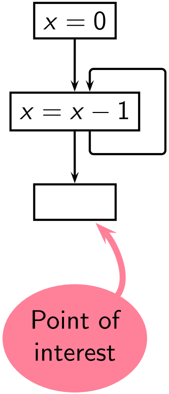


MFP does not exist and is not computable



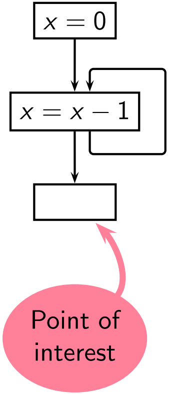
MFP exist and is computable

Computability of MFP



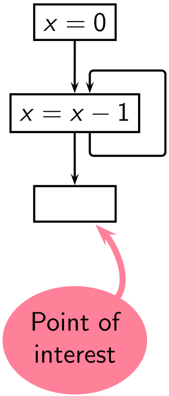
$\sqsubseteq$	$\leq$	$\leq$
$\sqcap$	min	min
Hasse diagram of the values of interest	0 | -1 | -2 | -3 | . . .	0 | -1 | -2 | -3 | . . . | $-\infty$
MFP exists?		
MFP computable?		
MoP exists?		

Computability of MFP



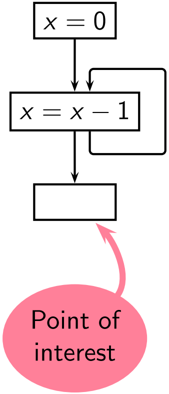
$\sqsubseteq$	$\leq$	$\leq$
$\sqcap$	min	min
Hasse diagram of the values of interest	0 | -1 | -2 | -3 | . . .	0 | -1 | -2 | -3 | . . . | $-\infty$
MFP exists?	No	
MFP computable?	No	
MoP exists?	No	

Computability of MFP



$\sqsubseteq$	$\leq$	$\leq$
$\sqcap$	min	min
Hasse diagram of the values of interest	0 — -1 — -2 — -3 — …	0 — -1 — -2 — -3 — … — $-\infty$
MFP exists?	No	Yes
MFP computable?	No	No
MoP exists?	No	Yes

Computability of MFP



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

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

Existence and Computation of the Maximum Fixed Point

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

Existence and Computation of the Maximum Fixed Point

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

Claims being made:

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

Existence and Computation of the Maximum Fixed Point

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

Claims being made:

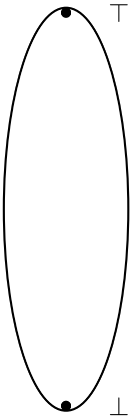
- $\exists k \ s.t. \ f^{k+1}(\top) = f^k(\top)$
- Since k is finite, $f^k(\top)$ exists and is computable
- $f^k(\top)$ is a fixed point
- $f^k(\top)$ is a the *maximum* fixed point

The proof depends on:

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

Existence and Computation of the Maximum Fixed Point

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



IIT Bombay
cs618: Program Analysis

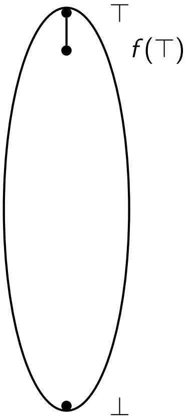
Topic:

Theoretical Abstractions

Section:

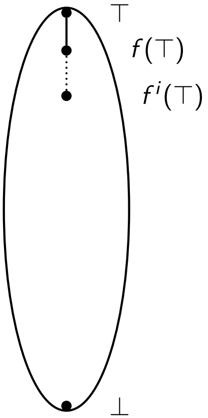
Existence and Computation of the Maximum Fixed Point

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



Existence and Computation of the Maximum Fixed Point

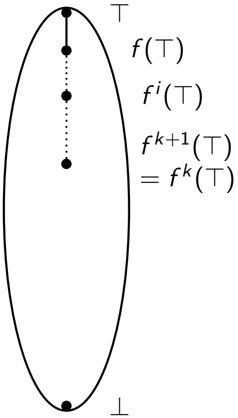
If L is a meet semilattice satisfying DCC, $f : L \rightarrow L$ is monotonic, then $MFP(f) = f^{k+1}(\top) = f^k(\top)$ such that $f^{j+1}(\top) \neq f^j(\top)$, $j < k$



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

Existence and Computation of the Maximum Fixed Point

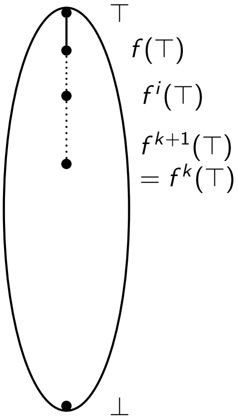
If L is a meet semilattice satisfying DCC, $f : L \rightarrow L$ is monotonic, then $MFP(f) = f^{k+1}(\top) = f^k(\top)$ such that $f^{j+1}(\top) \neq f^j(\top)$, $j < k$



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

Existence and Computation of the Maximum Fixed Point

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

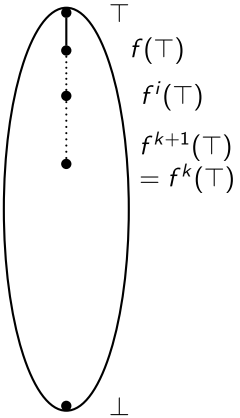


- $\top \sqsupseteq f(\top) \sqsupseteq f^{2}(\top) \sqsupseteq f^{3}(\top) \sqsupseteq f^{4}(\top) \sqsupseteq \ldots$
- Since strictly descending chains are finite, there must exist $f^{k}(\top)$ such that $f^{k+1}(\top) = f^{k}(\top)$ and $f^{j+1}(\top) \neq f^{j}(\top)$, $j < k$
- If p is a fixed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

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

Existence and Computation of the Maximum Fixed Point

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

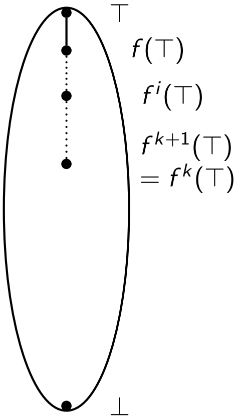


- $\top \sqsupseteq f(\top) \sqsupseteq f^2(\top) \sqsupseteq f^3(\top) \sqsupseteq f^4(\top) \sqsupseteq \ldots$
- Since strictly descending chains are finite, there must exist $f^k(\top)$ such that $f^{k+1}(\top) = f^k(\top)$ and $f^{j+1}(\top) \neq f^j(\top),\ j < k$
- If p is a fixed point of f then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

  Proof strategy: Induction on i for $f^i(\top)$
  - Basis ($i = 0$): $p \sqsubseteq f^0(\top) = \top$
  - Inductive Hypothesis: Assume that $p \sqsubseteq f^i(\top)$

Existence and Computation of the Maximum Fixed Point

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



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

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

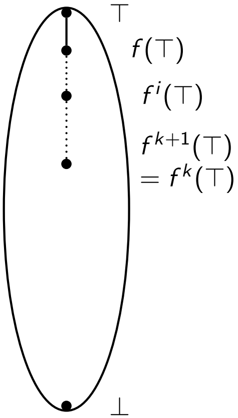
  - Basis ($i = 0$): $p \sqsubseteq f^0(\top) = \top$
  - Inductive Hypothesis: Assume that $p \sqsubseteq f^i(\top)$
  - Proof: $f(p) \sqsubseteq f(f^i(\top))$ (f is monotonic)

    $\Rightarrow \quad p \sqsubseteq f(f^i(\top))$ ($f(p) = p$)

    $\Rightarrow \quad p \sqsubseteq f^{i+1}(\top)$

Existence and Computation of the Maximum Fixed Point

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



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

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

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

Fixed Points Computation: Flow Functions Vs. Equations

- Recall that

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

Fixed Points Computation: Flow Functions Vs. Equations

- Recall that

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

  - What is f in the above?

Fixed Points Computation: Flow Functions Vs. Equations

- Recall that

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

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

Fixed Points Computation: Flow Functions Vs. Equations

- Recall that

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

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

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

Fixed Points Computation: Flow Functions Vs. Equations

- Recall that

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

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

- Our method computes the maximum fixed point of data flow equations!
- What is the relation between the maximum fixed point of data flow equations and the MFP defined above?

Fixed Points Computation: Flow Functions Vs. Equations

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

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

Fixed Points Computation: Flow Functions Vs. Equations

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

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

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

Fixed Points Computation: Flow Functions Vs. Equations

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

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

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

Fixed Points Computation: Flow Functions Vs. Equations

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

$$\mathcal{X} = \langle \; f_{In_1}(\mathcal{X}), \; f_{Out_1}(\mathcal{X}), \; \ldots \; f_{In_N}(\mathcal{X}), \; f_{Out_N}(\mathcal{X}), \; \rangle$$

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

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

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

Fixed Points Computation: Flow Functions Vs. Equations

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

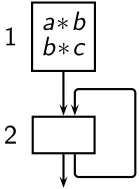
$$\mathcal{X} = \mathcal{F}(\mathcal{X})$$

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

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

An Instance of Available Expressions Analysis

Program

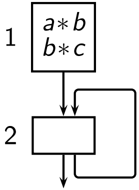


- Conventional data flow equations

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

An Instance of Available Expressions Analysis

Program

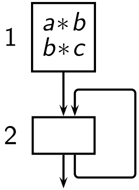


- Conventional data flow equations

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

An Instance of Available Expressions Analysis

Program



- Conventional data flow equations

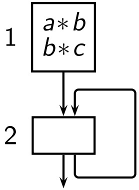
$$\begin{aligned} In_1 &= 00 & \qquad In_2 &= Out_1 \cap Out_2 \\ Out_1 &= 11 & \qquad Out_2 &= In_2 \end{aligned}$$

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

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

An Instance of Available Expressions Analysis

Program



- Conventional data flow equations

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

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

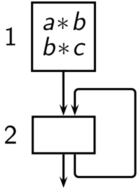
$$\mathcal{F}(\langle In_1, Out_1, In_2, Out_2 \rangle) = \langle 00, 11, Out_1 \cap Out_2, In_2 \rangle$$

- The maximum fixed point assignment is

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

An Instance of Available Expressions Analysis

Program



- Conventional data flow equations

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

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

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

- The maximum fixed point assignment is

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

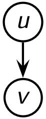
- The minimum fixed point assignment is

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

The Essential Difference Between MFP and MoP

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

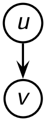
  because $FP(v) = \prod\limits_{u \in pred(v)} f_{u\rightarrow v}(FP(u))$



The Essential Difference Between MFP and MoP

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

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



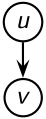
- Such a relationship does not exist for MoP

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

The Essential Difference Between MFP and MoP

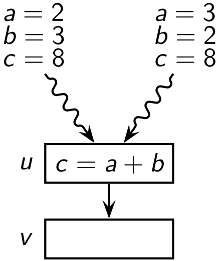
- For all edges $u \to v$, $FP(v) \sqsubseteq f_{u\to v}\,(FP(u))$

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



- Such a relationship does not exist for MoP

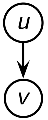
  because $MoP(v)$ is not computed from $MoP(u)$



The Essential Difference Between MFP and MoP

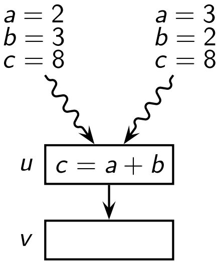
- For all edges $u \rightarrow v$, $FP(v) \sqsubseteq f_{u\rightarrow v}\,(FP(u))$

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



- Such a relationship does not exist for MoP

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



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

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

The Essential Difference Between MFP and MoP (1)

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

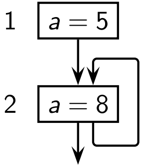
$$MoP(v) = \bigsqcap_{\rho_v \in Paths(v)} f_{\rho_v}(BI) = f_{\rho^0}(BI) \sqcap f_{\rho^1}(BI) \sqcap \ldots f_{\rho^i}(BI) \sqcap \ldots$$

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

- $MoP(v)$ is unrelated to $MoP(u)$; $MFP(v)$ may be influenced by $MFP(u)$ if there is a path from u to v
- $MoP(v)$ needs to iterate over all paths reaching v. For termination,
  - the meet across all paths upto some ρ_i should result in $\perp$ value, or
  - all paths reaching v should be exhausted
  - DCC does not guarantee anything unless we reach the bottom
- $MFP(v)$ needs to iterate over the entire CFG repeatedly
  - In each iteration over the graph, it needs to iterate over all predecessors
  - Termination depends on finding two successive value that are same Guaranteed by DCC and monotonicity

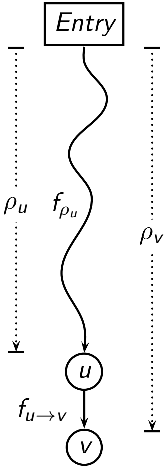
The Essential Difference Between MFP and MoP (2)

Consider a constant propagation example

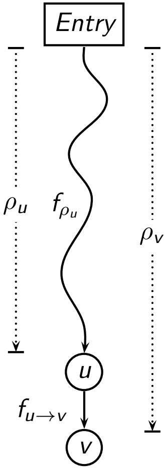


- An algorithm to compute $MoP(2)$ needs to consider the paths $(1),\ (1,2),\ (1,2,2),\ (1,2,2,2),\ \ldots$
- After how many paths should it terminate?
  Unless we get a $\perp$ value, we cannot ignore the remaining paths

Soundness of FP Assignment: FP $\sqsubseteq$ MoP

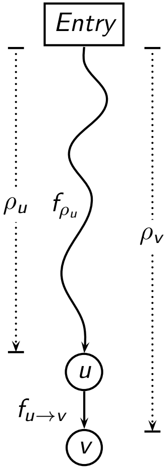


Soundness of FP Assignment: FP $\sqsubseteq$ MoP



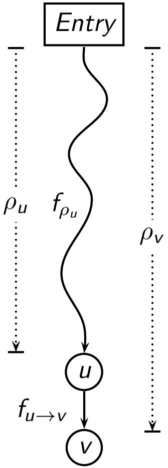
- $MoP(v) \quad = \displaystyle\prod_{\rho \in Paths(v)} f_\rho(BI)$

Soundness of FP Assignment: FP $\sqsubseteq$ MoP



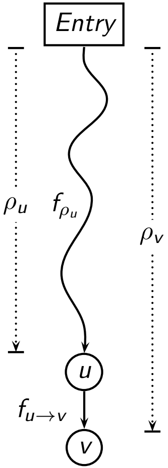
- $MoP(v) \quad = \quad \displaystyle\bigsqcap_{\rho \,\in\, Paths(v)} f_\rho(BI)$
- Proof Obligation: $\forall \rho_v \; FP(v) \sqsubseteq f_{\rho_v}(BI)$

Soundness of FP Assignment: FP $\sqsubseteq$ MoP



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

Soundness of FP Assignment: FP $\sqsubseteq$ MoP



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

  Base case: Path of length 0

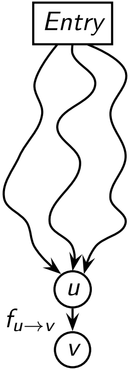
  $FP(Entry) = MoP(Entry) = BI$

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

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

This holds for every *FP* an hence for *MFP* also

MFP = MoP for Distributive Frameworks



$$
\begin{aligned}
\forall v,\quad MoP(v) &= \bigsqcap_{\rho \in Paths(v)} f_\rho(BI) \\
&= \bigsqcap_{u \in Pred(v)} \left(\bigsqcap_{\rho' \in Paths(u)} f_{u\rightarrow v}\left(f_{\rho'}(BI)\right) \right) \\
&= \bigsqcap_{u \in Pred(v)} f_{u\rightarrow v} \left(\bigsqcap_{\rho' \in Paths(u)} f_{\rho'}(BI) \right) && \text{(by distributivity)} \\
&= \bigsqcap_{u \in Pred(v)} f_{u\rightarrow v}\left(MoP(u)\right) && \text{(definition of MoP)}
\end{aligned}
$$

Thus MoP is a fixed point when flow functions are distributive. Hence,

$$
\begin{aligned}
& \forall v,\quad MoP(v) \sqsubseteq MFP(v) && \text{(every fixed point is weaker than MFP)} \\
& \forall v,\quad MoP(v) \sqsupseteq MFP(v) && \text{(by the soundness of MFP)} \\
\Rightarrow\quad & \forall v,\quad MFP(v) = MoP(v)
\end{aligned}
$$

Theoretical Abstractions: A Summary

Theoretical Abstractions: A Summary

Necessary and sufficient conditions for designing a data flow framework

Theoretical Abstractions: A Summary

Necessary and sufficient conditions for designing a data flow framework

- A meet semilattice satisfying dcc

Theoretical Abstractions: A Summary

Necessary and sufficient conditions for designing a data flow framework

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

Theoretical Abstractions: A Summary

Necessary and sufficient conditions for designing a data flow framework

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

Theoretical Abstractions: A Summary

Necessary and sufficient conditions for designing a data flow framework

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

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 stabilize

  + 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 stabilize

  + 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 stabilize

  + Simplest to understand and implement
  − May perform unnecessary computations

Our examples use this method

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

  Termination : When the list becomes empty

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

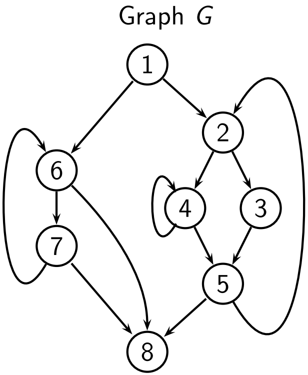
Elimination Methods of Performing Data Flow Analysis

Delayed computations of dependent data flow values of dependent nodes

Find suitable single-entry regions

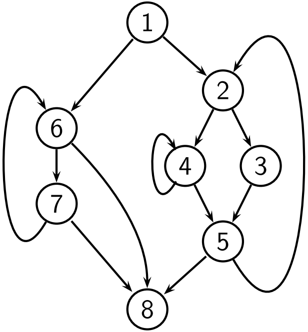
- *Interval Based Analysis*. Uses graph partitioning
- T_1, T_2 *Based Analysis*. Uses graph parsing

Classification of Edges in a Graph

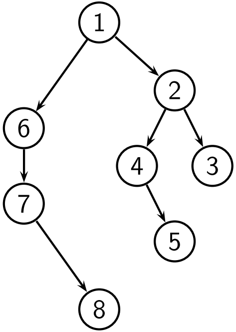


Classification of Edges in a Graph

Graph G

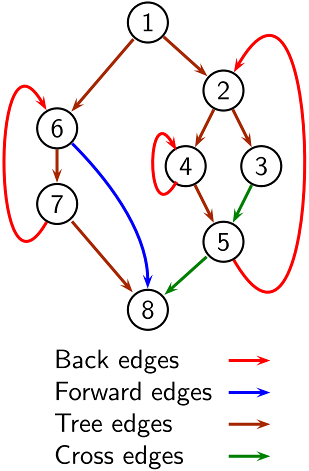


A depth first spanning tree of G

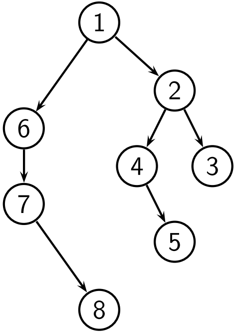


Classification of Edges in a Graph

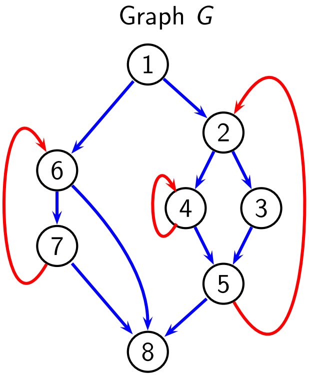
Graph G

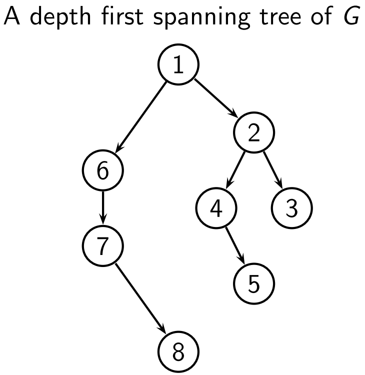


A depth first spanning tree of G



Classification of Edges in a Graph



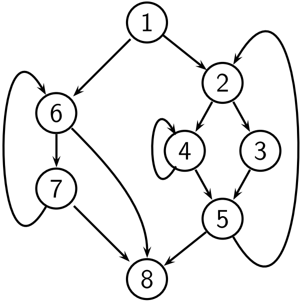


For data flow analysis, we club *tree*, *forward*, and *cross* edges into *forward* edges. Thus we have just forward or back edges in a control flow graph

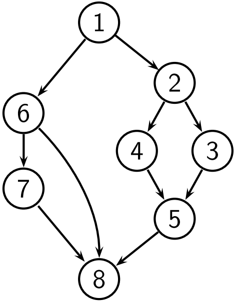
Reverse Post Order Traversal

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

Graph G



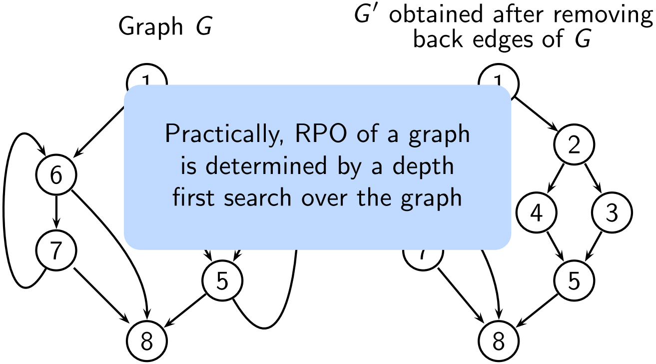
G' obtained after removing back edges of G



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

Reverse Post Order Traversal

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



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

Round Robin Iterative Algorithm

$In_0 = BI$
for all $j \neq 0$ **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**
        $\{$ $In_j = temp$
            $change = true$
        $\}$
    $\}$
$\}$

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 $j \neq 0$ **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)

Round Robin Iterative Algorithm

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

- Computation of Out_j has been left implicit

  Works fine for unidirectional frameworks
- $\top$ is the identity of $\sqcap$ (line 3)
- 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 $\quad \{ \quad temp = \bigsqcap_{p \in pred(j)} f_p(In_p)$
9 $\quad\quad$ **if** $temp \neq In_j$ **then**
10 $\quad\quad \{ \quad In_j = temp$
11 $\quad\quad\quad change = true$
12 $\quad\quad \}$
13 $\quad \}$
14 $\}$

- Computation of Out_j has been left implicit

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

  $\Rightarrow$ no need of initialization

Complexity of Round Robin Iterative Algorithm

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

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

Complexity of Round Robin Iterative Algorithm

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

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

- What about bidirectional bit vector frameworks?

Complexity of Round Robin Iterative Algorithm

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

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

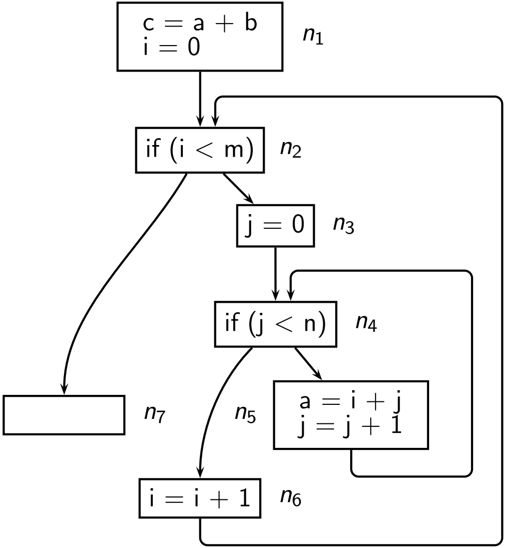
- What about bidirectional bit vector frameworks?
- What about other frameworks?

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

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

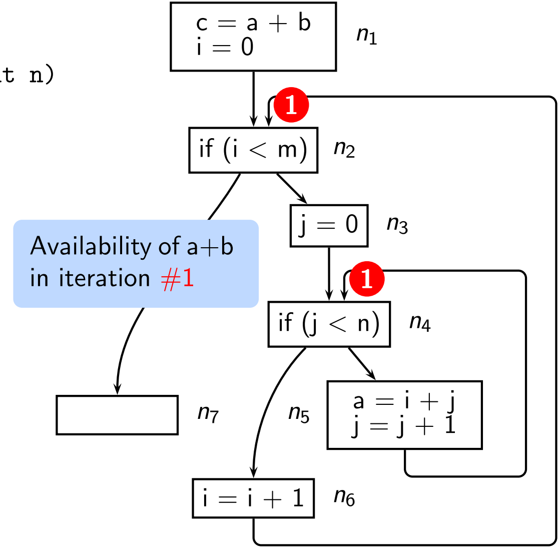
Example C Program with d(G,T) = 2

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



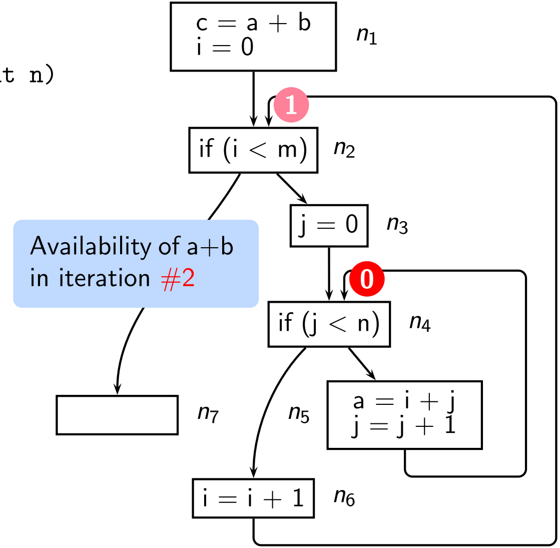
Example C Program with d(G,T) = 2

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



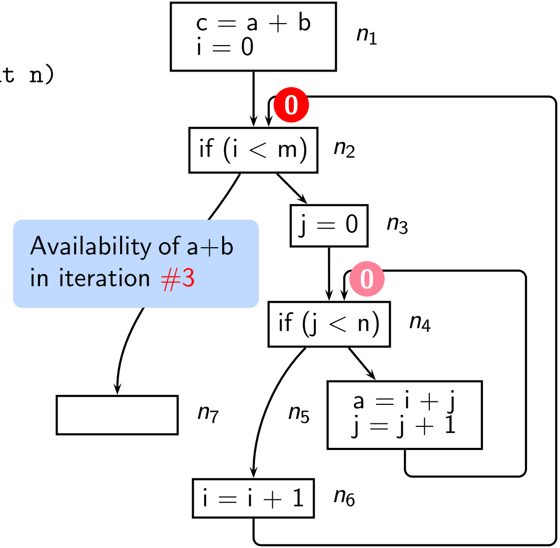
Example C Program with d(G,T) = 2

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



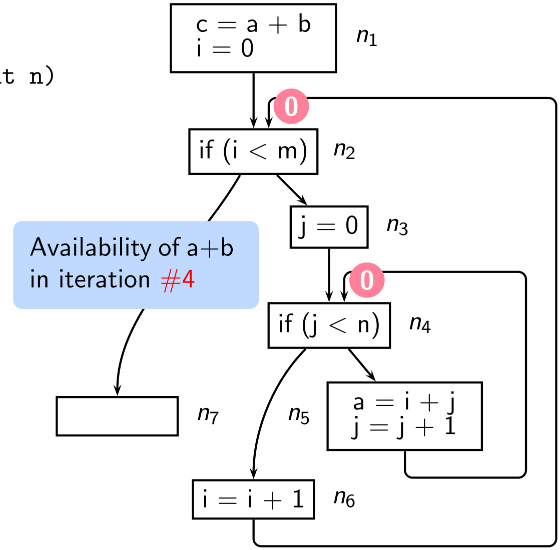
Example C Program with d(G,T) = 2

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



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

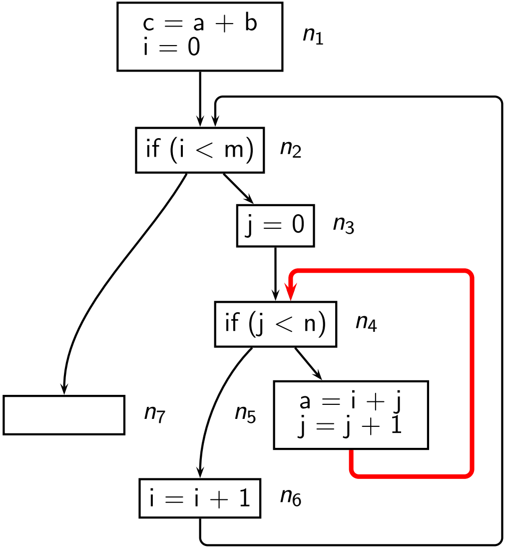
```
void fun(int m, int n)
{
   int i,j,a,b,c;
   c=a+b;
   i=0;
   while(i<m)
   {
         j=0;
         while(j<n)
         {
            a=i+j;
            j=j+1;
         }
         i=i+1;
      }
   }
```



3 + 1 iterations for available expressions analysis

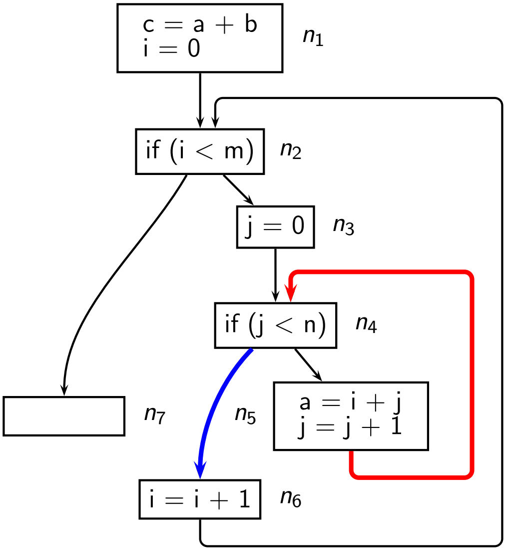
Example C Program with d(G,T) = 2

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



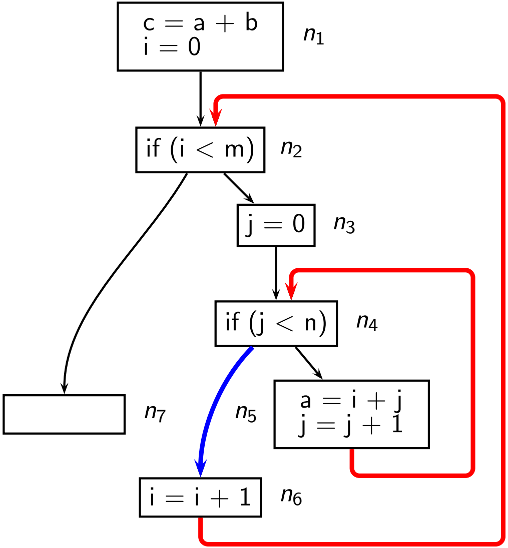
Example C Program with d(G,T) = 2

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



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

```
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;
      }
   }
```



Work List Based Iterative Algorithm

Directly traverses information flow paths

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

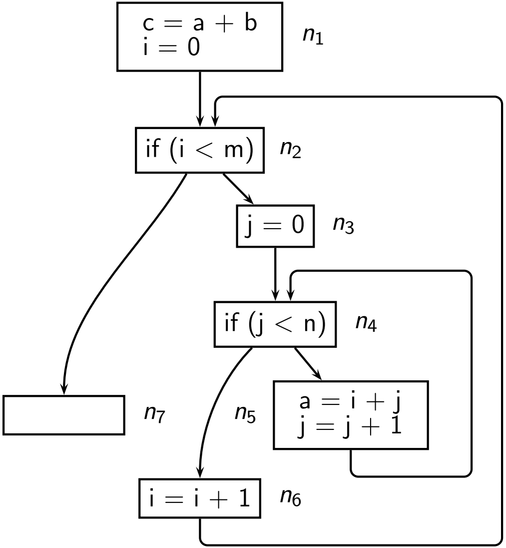
Tutorial Problem

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

Show the trace of the analysis in the following format:

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

Tutorial Problem for Work List Based Analysis



For available expressions analysis

- Round robin method needs 3+1 iterations

  Total number of nodes processed $= 7 \times 4 = 28$

- We illustrate work list method for expression $a + b$

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

Tutorial Problem for Work List Based Analysis

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

Comparing the Algorithms for Performing Data Flow Analysis

Round Robin Algorithm

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

Work List Algorithm

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

An Efficient Work List Algorithm (1)

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

    In the latter case, the work list must be examined from the beginning
- Notation
    - The work list is an array *WL* whose indices are RPO numbers
      $WL[\,i\,] = true \Rightarrow$ the node with RPO number i needs to be processed
    - $RPO[i]$ gives the RPO number of node i
    - $NODE[i]$ gives the node whose RPO number is i

An Efficient Work List Algorithm (2)

```
Efficient_Work_List_DFA()
    Initialize()
    i = Get_Node()
    while (i ≠ −1) do
        Process_Node(i)
        i = Get_Node()

Initialize()
    for (rpo = 0 to N − 1) do
        j = NODE[rpo]
        if (rpo = 0) then  IN_j = BI
        else  IN_j = ⊤
        Out_j = ⊤
        WL[rpo] = true

Get_Node()
    for (rpo = 0 to N − 1) do
        if (WL[rpo] = true) then
            WL[rpo] = false)
            return NODE[rpo]
    return −1

Process_Node(j)
    if (RPO[j] ≠ 0) then
        In_j = ⊓_{p ∈ pred(j)} (OUT_p)
    temp = f_j(In_j)
    if (temp ≠ OUT_j) then
        OUT_j = temp
        for (all s ∈ succ(j)) do
            srpo = RPO[s]
            WL[srpo] = true
```

Improving the Work List Algorithm Further

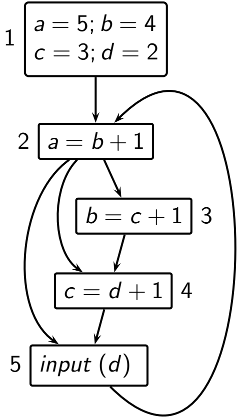
- During initialization, add only those nodes to the work list whose OUT value is non-$\top$
- For selecting a node, start seaching from the lowest *rpo* number added to the work list
  - Let *Process_Node* function return this number
  - Pass this number to *Get_Node* function

Improving the Work List Algorithm Further

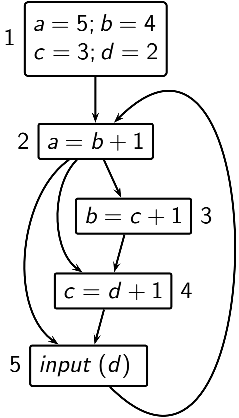
```
Efficient_Work_List_DFA()
    Initialize()
    i = Get_Node(0)
    while (i ≠ −1) do
        r = Process_Node(i)
        i = Get_Node(r)

Initialize()
    for (rpo = 0 to N − 1) do
        j = NODE[rpo]
        if (rpo = 0) then  IN_j = BI
        else  IN_j = ⊤
        Out_j = f_j(IN_j)
        if (OUT_j ≠ ⊤) then
                WL[rpo] = true
        else WL[rpo] = false

Get_Node(r)
    for (rpo = r to N − 1) do
        if (WL[rpo] = true) then
            WL[rpo] = false)
            return NODE[rpo]
    return −1

Process_Node(j)
    if (RPO[j] ≠ 0) then
        In_j = ⊓_{p ∈ pred(j)} (OUT_p)
    temp = f_j(In_j)
    r = j + 1
    if (temp ≠ OUT_j) then
        OUT_j = temp
        for (all s ∈ succ(j)) do
            srpo = RPO[s]
            WL[srpo] = true
            if (srpo < r) then  r = srpo
    return r
```

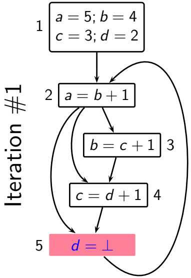
Precise Modelling of General Flows

Complexity of Constant Propagation?

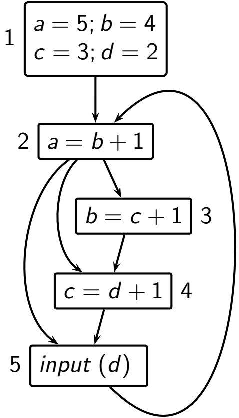


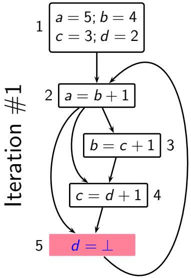
Complexity of Constant Propagation?

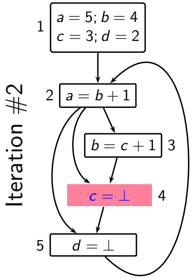




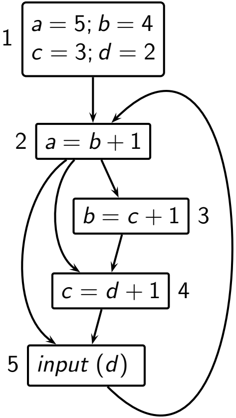
Complexity of Constant Propagation?

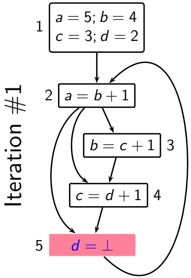


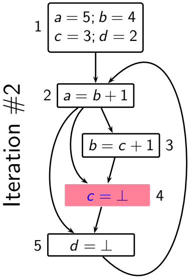


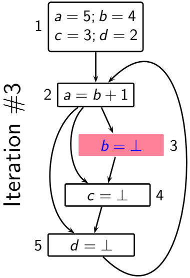


Complexity of Constant Propagation?

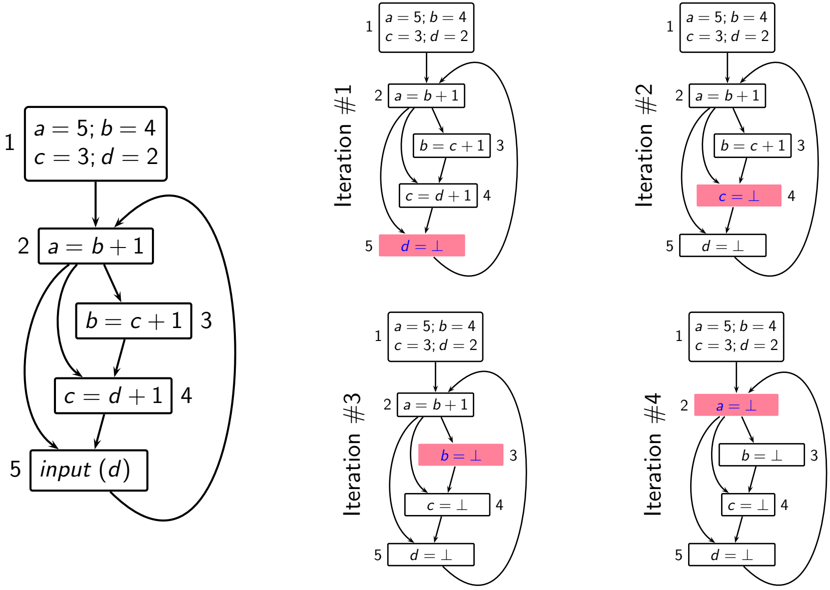








Complexity of Constant Propagation?



Extra Material

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

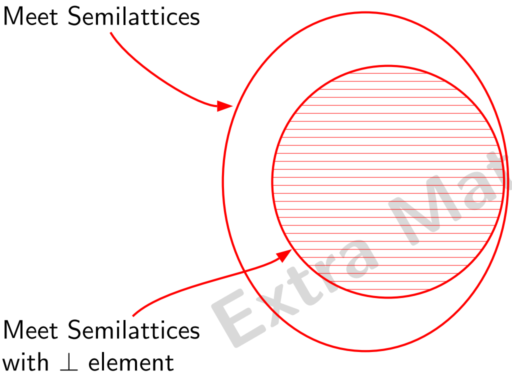
A Bounded Lattice Need Not be Complete

- Consider
$$A = \{1/n \mid n \in \mathbb{N}\} \cup \{-1/n \mid n \in \mathbb{N}\}$$
where $\mathbb{N}$ is the set of natural numbers
- Then, A contains all rational numbers from 1 to -1 *except* 0
- The poset $L = (A, \leq)$ is a
  - For all finite subsets of A we have the smallest and the largest number in A
    $\Rightarrow L$ is a lattice
  - 1 is the largest number in A and -1 is the smallest number in A
    $\Rightarrow L$ is a bounded lattice with $\top = 1$ and $\bot = -1$
  - there is no number that is greatest for the infinite set of all negative numbers in A
    $\Rightarrow L$ is not a complete lattice

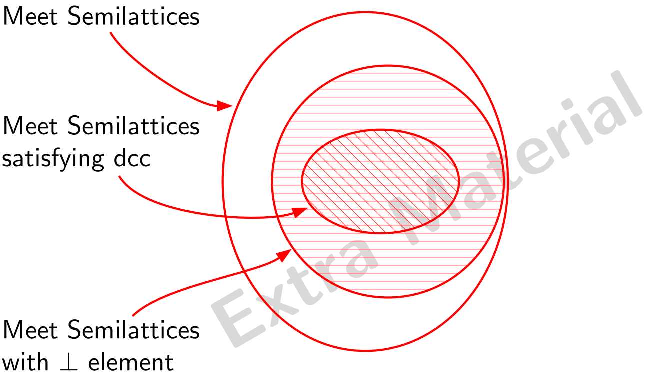
Variants of Lattices

Meet Semilattices

Variants of Lattices

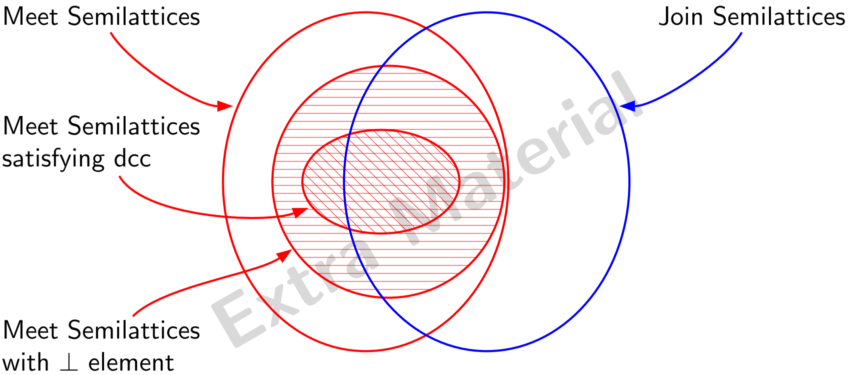


Variants of Lattices



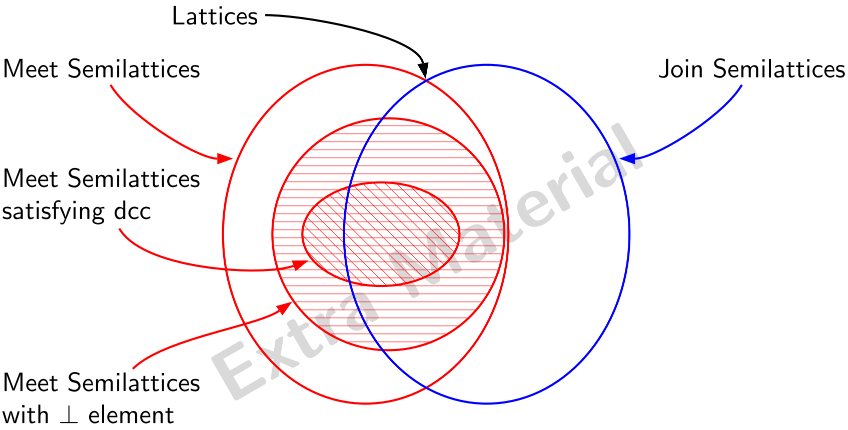
- dcc: descending chain condition

Variants of Lattices



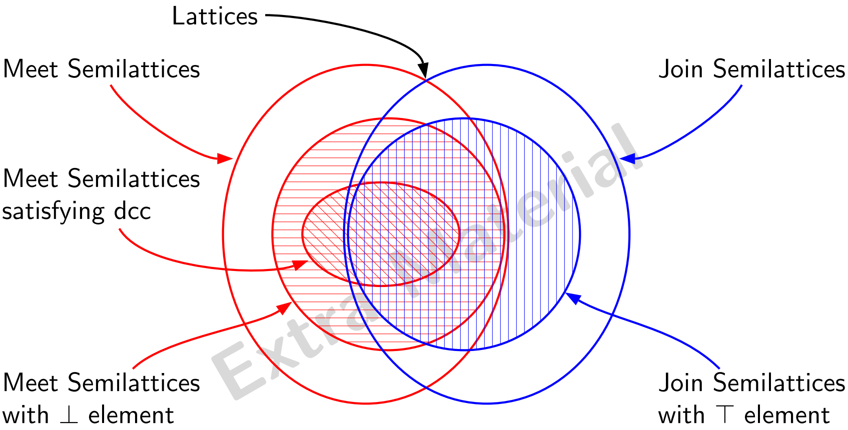
- dcc: descending chain condition

Variants of Lattices



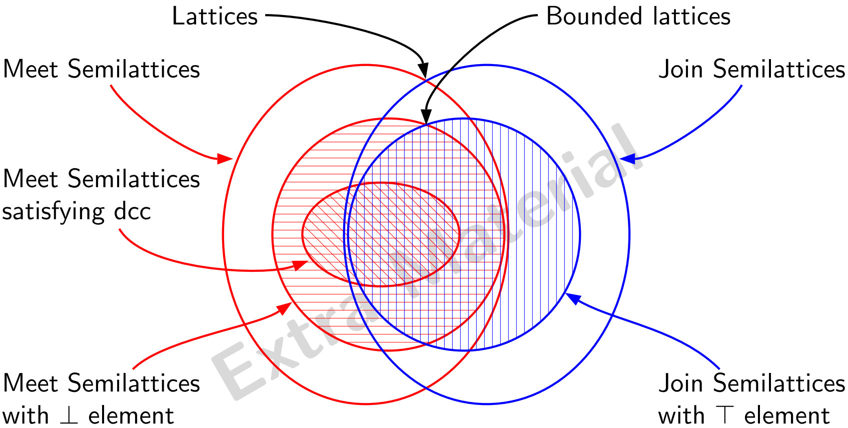
- dcc: descending chain condition

Variants of Lattices



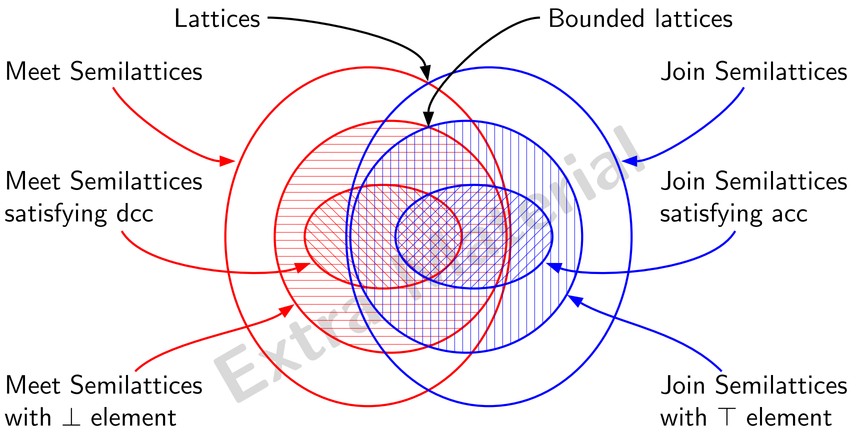
- dcc: descending chain condition

Variants of Lattices



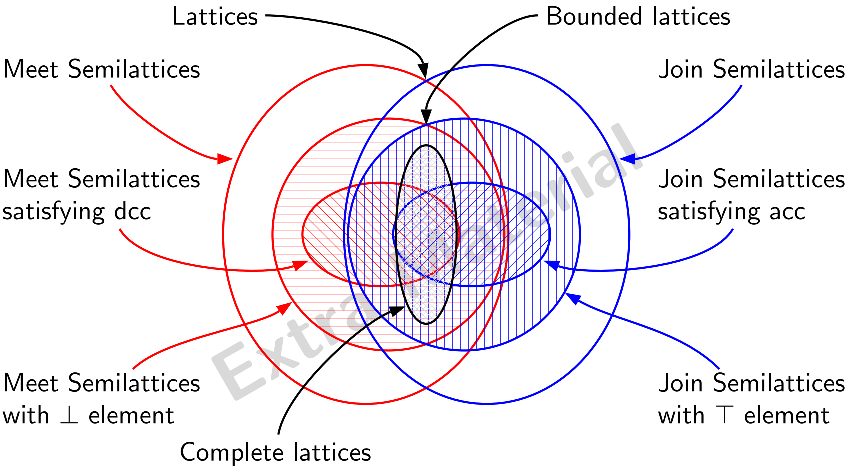
- dcc: descending chain condition

Variants of Lattices



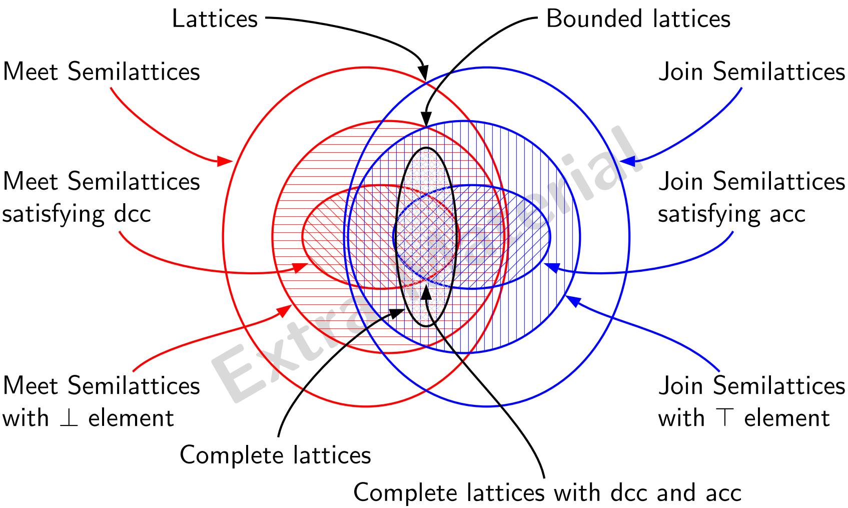
- dcc: descending chain condition
- acc: ascending chain condition

Variants of Lattices



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

Variants of Lattices



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

Example of Cartesian Product: Concept Lattices

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

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

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

Example of Concept Lattice (1)

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

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

Example of Concept Lattice (2)

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

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

Example of Concept Lattice (2)

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

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

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

Example of Concept Lattice (2)

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

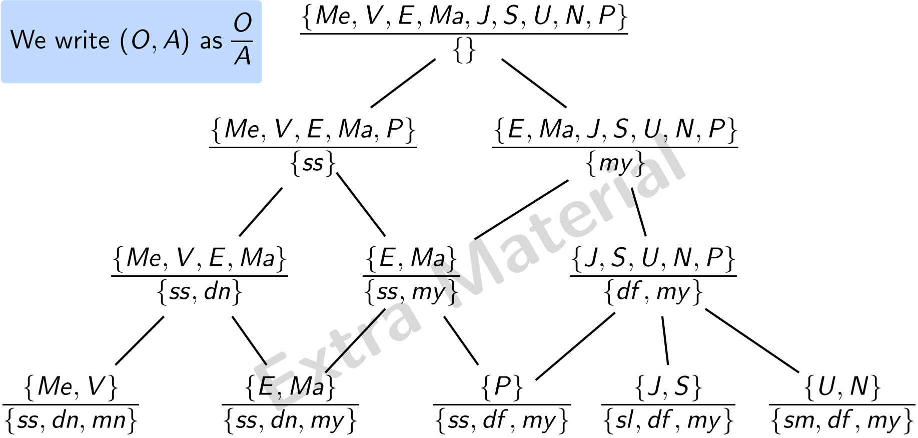
$$\frac{\{Me, V, E, Ma, J, S, U, N, P\}}{\{\}}$$

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

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

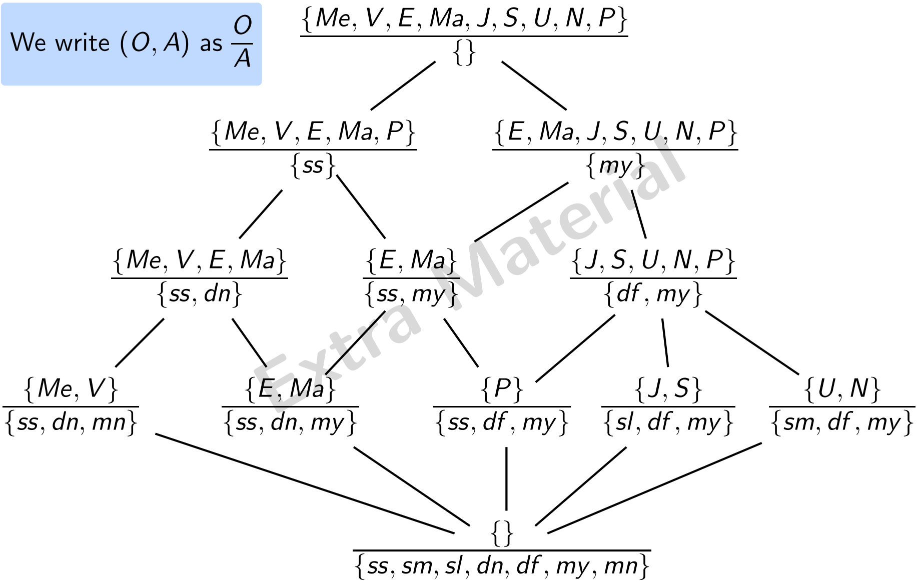
Example of Concept Lattice (2)

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



Example of Concept Lattice (2)

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



IIT Bombay
cs618: Program Analysis

Topic:
Theoretical Abstractions

Section:
More General Setting
Data Flow Values
Flow Functions
Solutions
Algorithms
Modelling General Flows
Extra Material

Post's Correspondence Problem (PCP)

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

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

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 $2, 3, 2$

$$\begin{aligned} u_2 u_3 u_2 &= \boxed{11}\ \boxed{100}\ \boxed{11} \\ v_2 v_3 v_2 &= \boxed{1}\ \boxed{11001}\ \boxed{1} \end{aligned}$$

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 $2, 3, 2$

$$\begin{aligned} u_2 u_3 u_2 &= \boxed{11}\,\boxed{100}\,\boxed{11} \\ v_2 v_3 v_2 &= \boxed{1}\,\boxed{11001}\,\boxed{1} \end{aligned}$$

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

Post's Correspondence Problem (PCP)

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

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

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

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

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

$$\begin{aligned} u_2 u_3 u_2 &= \boxed{11}\ \boxed{100}\ \boxed{11} \\ v_2 v_3 v_2 &= \boxed{1}\ \boxed{11001}\ \boxed{1} \end{aligned}$$

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

Post's Correspondence Problem (PCP)

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

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

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

Indices could repeat without any bound

Modified Post's Correspondence Problem (MPCP)

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

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

Modified Post's Correspondence Problem (MPCP)

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

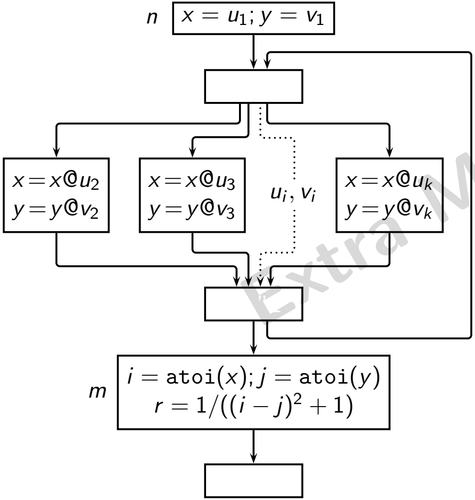
$$u_1 u_{i_1} u_{i_2} \ldots u_{i_m} = v_1 v_{i_1} v_{i_2} \ldots v_{i_m}$$

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

$$\begin{array}{rcl} u_1 u_3 u_2 u_2 u_4 & = & \boxed{11}\ \boxed{0111}\ \boxed{1}\ \boxed{1}\ \boxed{10} \\ v_1 v_3 v_2 v_2 v_4 & = & \boxed{1}\ \boxed{10}\ \boxed{111}\ \boxed{111}\ \boxed{0} \end{array}$$

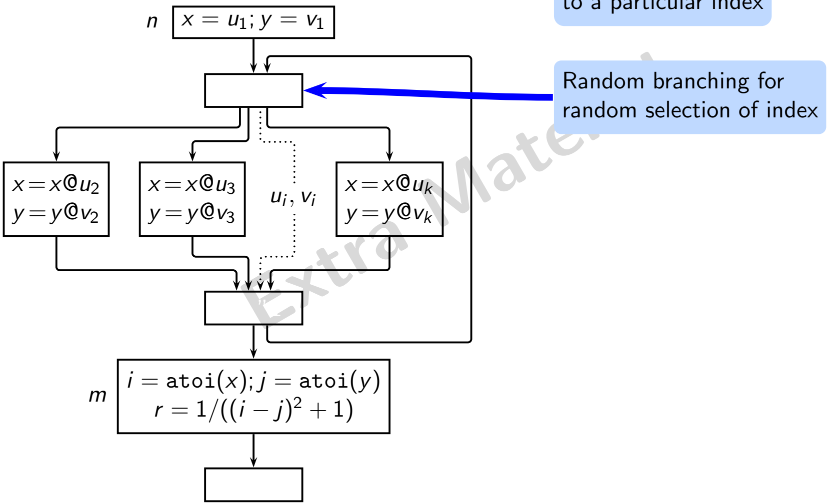
Hecht's Reduction of MPCP to Constant Propagation

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



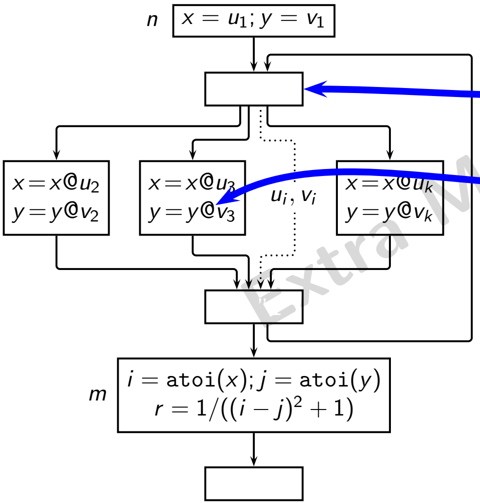
Hecht's Reduction of MPCP to Constant Propagation

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



Hecht's Reduction of MPCP to Constant Propagation

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



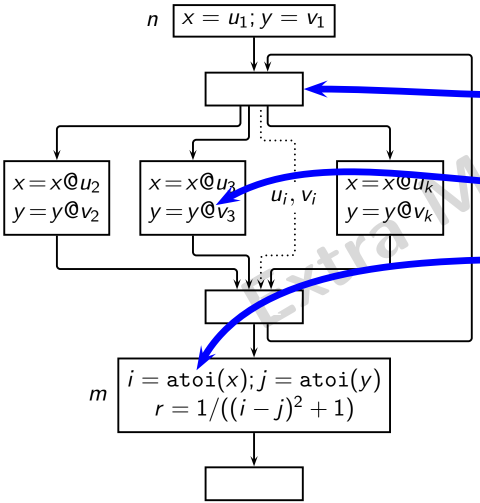
Each block in the loop corresponds to a particular index

Random branching for random selection of index

String append

Hecht's Reduction of MPCP to Constant Propagation

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



Each block in the loop corresponds to a particular index

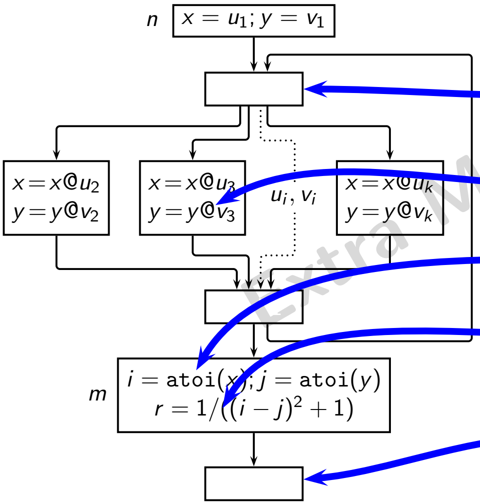
Random branching for random selection of index

String append

String to integer conversion

Hecht's Reduction of MPCP to Constant Propagation

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



Each block in the loop corresponds to a particular index

Random branching for random selection of index

String append

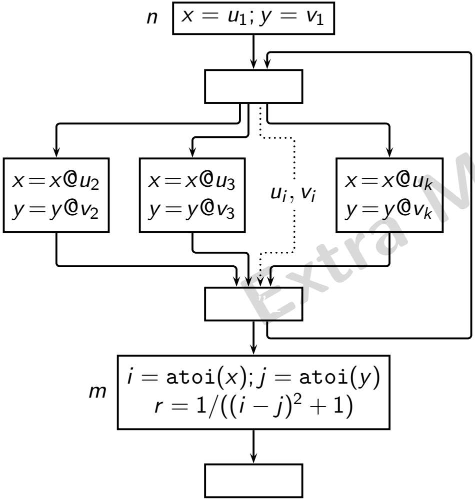
String to integer conversion

Integer division

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

Hecht's Reduction of MPCP to Constant Propagation

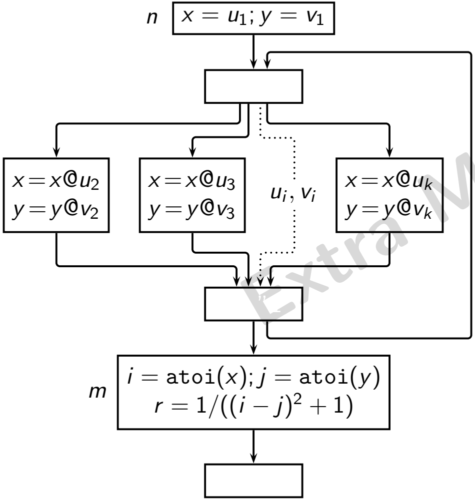
Given: An instance of MPCP with $\Sigma = \{0, 1\}$



Every path from node n to node m represents a separate pair of strings to be checked for equality

Hecht's Reduction of MPCP to Constant Propagation

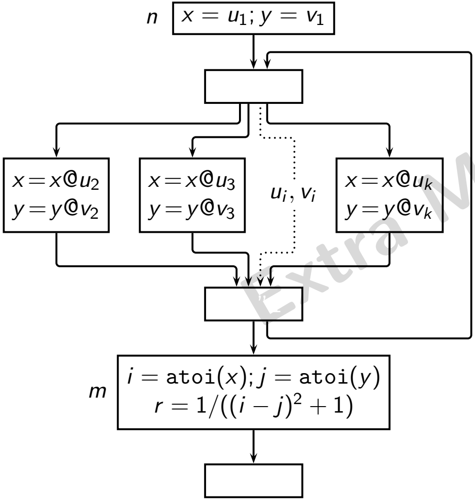
Given: An instance of MPCP with $\Sigma = \{0, 1\}$



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

Hecht's Reduction of MPCP to Constant Propagation

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



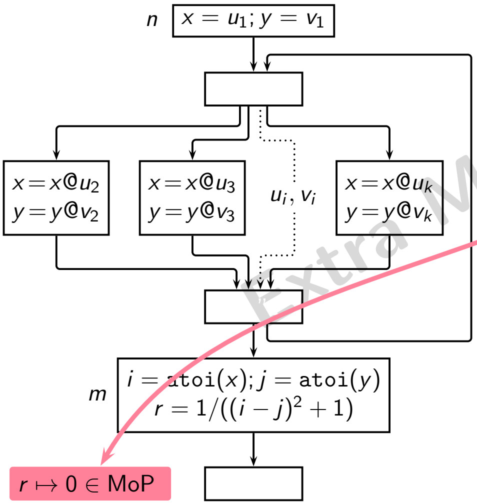
- $i = j \Rightarrow r = 1$
  $i \neq j \Rightarrow r = 0$
- **If** there exists an algorithm which can determine that

  {

  }

Hecht's Reduction of MPCP to Constant Propagation

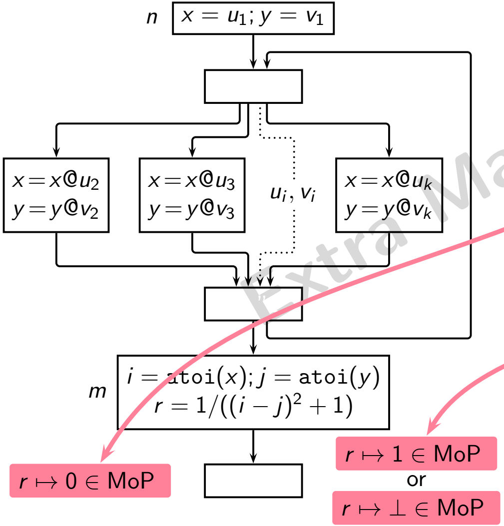
Given: An instance of MPCP with $\Sigma = \{0, 1\}$



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

Hecht's Reduction of MPCP to Constant Propagation

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



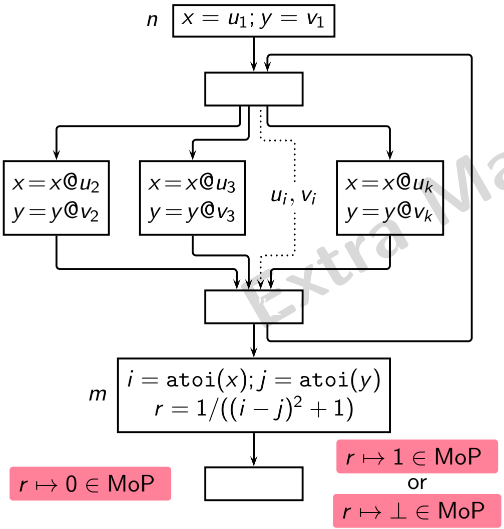
- $i = j \Rightarrow r = 1$
  $i \neq j \Rightarrow r = 0$
- **If** there exists an algorithm which can determine that
  {
  - $r = 0$ along every path (x is never equal to y, MPCP instance does not have a solution)
  - $r = 1$ along some path (some x is equal to y, MPCP instance has a solution)

  }

  Then MPCP is decidable

Hecht's Reduction of MPCP to Constant Propagation

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



The tricky part!!

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

  $\}$

  Then MPCP is decidable

Hecht's Reduction of MPCP to Constant Propagation

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

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

The tricky part!!

$i = j \Rightarrow r = 1$

$i \neq j \Rightarrow r = 0$

If there exists an algorithm which can determine that

{

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

}

Then MPCP is decidable

Hecht's Reduction of MPCP to Constant Propagation

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

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

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

The tricky part!!

$i = j \Rightarrow r = 1$

$i \neq j \Rightarrow r = 0$

If there exists an algorithm which can determine that

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

}

Then MPCP is decidable

Hecht's Reduction of MPCP to Constant Propagation

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

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

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

- The values computed at the entry of m consist of sets of pairs of strings (x, y)

  Under the substring relation between strings, these sets follow descending chains

  Each iteration produces longer strings without any bound; hence DCC is violated

The tricky part!!

$i = j \Rightarrow r = 1$

$i \neq j \Rightarrow r = 0$

If there exists an algorithm which can determine that

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

}

Then MPCP is decidable

Tarski's Fixed Point Theorem

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

Define

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

Then

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

Tarski's Fixed Point Theorem

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

Define

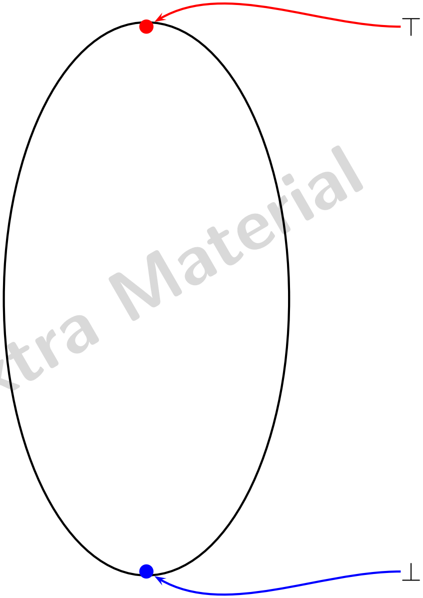
$$
\begin{array}{lrcl}
p \text{ is a fixed point of } f: & Fix(f) & = & \{p \mid f(p) = p\} \\
f \text{ is reductive at } p: & Red(f) & = & \{p \mid f(p) \sqsubseteq p\} \\
f \text{ is extensive at } p: & Ext(f) & = & \{p \mid f(p) \sqsupseteq p\}
\end{array}
$$

Then

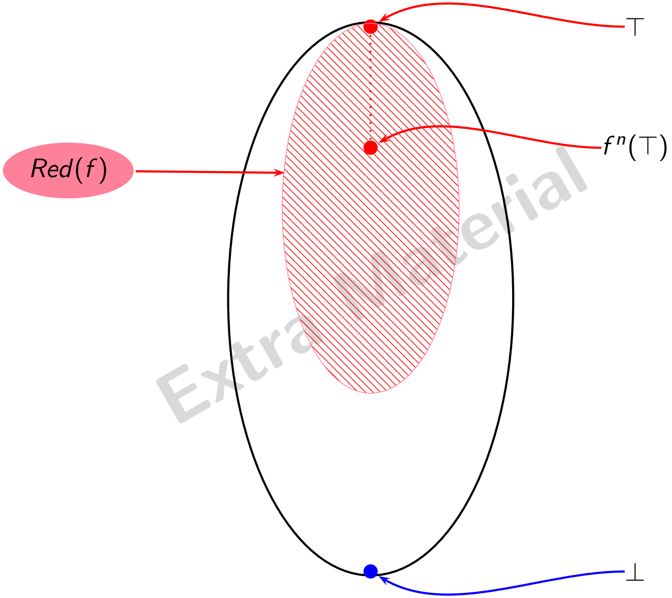
$$
\begin{array}{rcl}
LFP(f) & = & \bigsqcap Red(f) \in Fix(f) \\
MFP(f) & = & \bigsqcup Ext(f) \in Fix(f)
\end{array}
$$

Guarantees only existence, not computability of fixed points

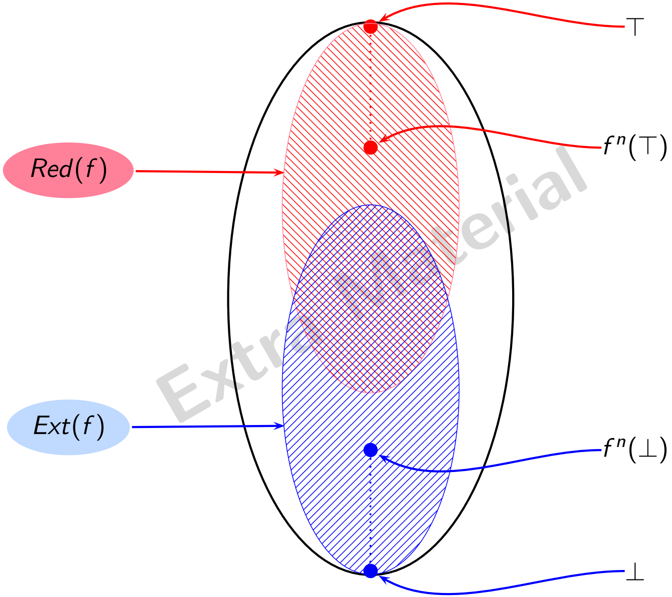
Fixed Points of a Function



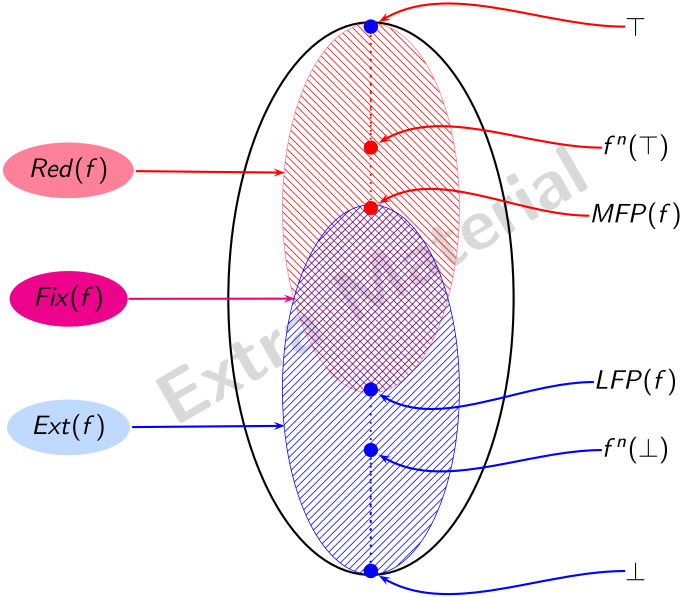
Fixed Points of a Function



Fixed Points of a Function

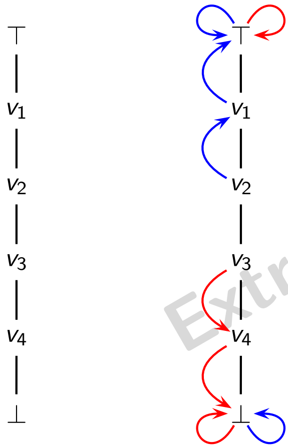


Fixed Points of a Function



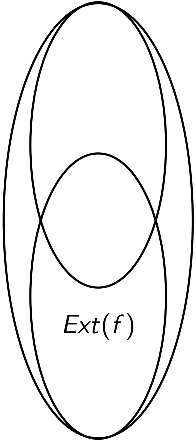
Examples of Reductive and Extensive Sets

Finite L Monotonic $f : L \to L$

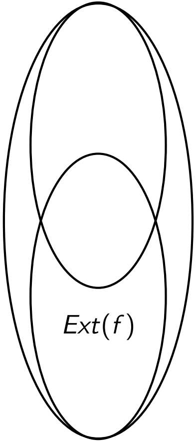


$$
\begin{aligned}
\textcolor{red}{Red}(f) &= \{\top, v_3, v_4, \bot\} \\
\textcolor{blue}{Ext}(f) &= \{\top, v_1, v_2, \bot\} \\
Fix(f) &= \textcolor{red}{Red}(f) \cap \textcolor{blue}{Ext}(f) \\
&= \{\top, \bot\} \\
MFP(f) &= lub\,(\textcolor{blue}{Ext}(f)) \\
&= lub\,(Fix(f)) \\
&= \top \\
LFP(f)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

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

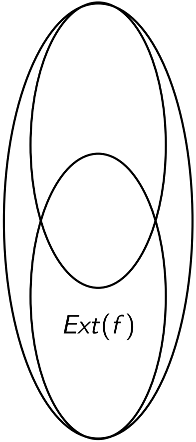


Existence of MFP: Proof of Tarski's Fixed Point Theorem



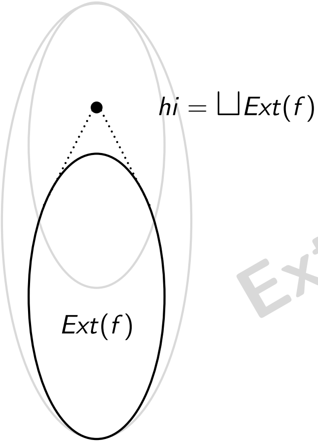
1. Claim 1: Let $X \subseteq L$.
   $\forall x \in X,\ p \sqsupseteq x \Rightarrow p \sqsupseteq \bigsqcup(X)$.

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



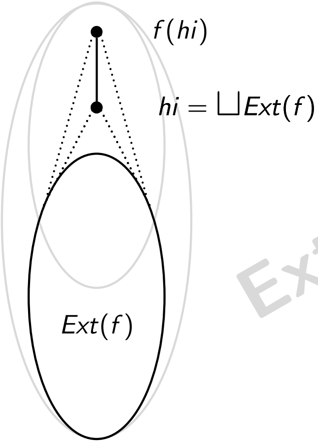
1. Claim 1: Let $X \subseteq L$.
   $\forall x \in X,\ p \sqsupseteq x \Rightarrow p \sqsupseteq \bigsqcup(X)$.
2. In the following we use $Ext(f)$ as X

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



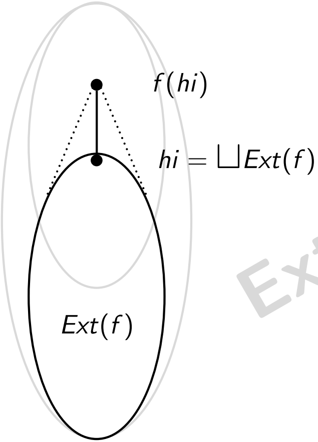
1. Claim 1: Let $X \subseteq L$.
   $\forall x \in X,\ p \sqsupseteq x \Rightarrow p \sqsupseteq \bigsqcup(X)$.
2. In the following we use $Ext(f)$ as X
3. $\forall p \in Ext(f),\ hi \sqsupseteq p$

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



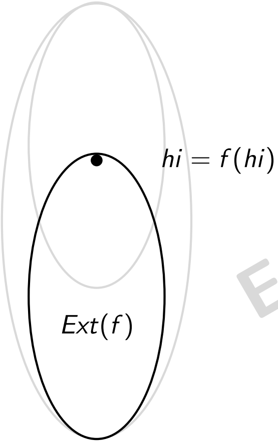
1. Claim 1: Let $X \subseteq L$.
   $\forall x \in X,\ p \sqsupseteq x \Rightarrow p \sqsupseteq \bigsqcup(X)$.
2. In the following we use $Ext(f)$ as X
3. $\forall p \in Ext(f),\ hi \sqsupseteq p$
4. $hi \sqsupseteq p \ \Rightarrow f(hi) \sqsupseteq f(p) \sqsupseteq p$ (monotonicity)
   $\Rightarrow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

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



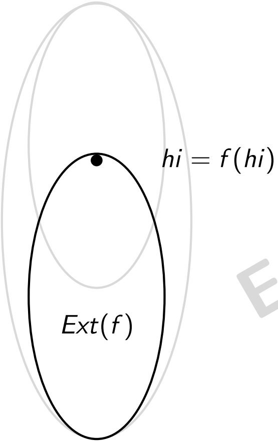
1. Claim 1: Let $X \subseteq L$.
   $\forall x \in X,\ p \sqsupseteq x \Rightarrow p \sqsupseteq \bigsqcup(X)$.
2. In the following we use $Ext(f)$ as X
3. $\forall p \in Ext(f),\ hi \sqsupseteq p$
4. $hi \sqsupseteq p \;\Rightarrow f(hi) \sqsupseteq f(p) \sqsupseteq p$ (monotonicity)
   $\Rightarrow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

Existence and Computation of the Maximum Fixed Point

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

Existence and Computation of the Maximum Fixed Point

- For monotonic $f : L \to L$
  - Existence: $MFP(f) = \bigsqcup Ext(f) \in Fix(f)$
    Requires L to be complete
  - Computation: $MFP(f) = f^{k+1}(\top) = f^{k}(\top)$ such that $f^{j+1}(\top) \neq f^{j}(\top)$, $j < k$.
    Requires all *strictly descending* chains to be finite

IIT Bombay
cs618: Program Analysis

Topic:
Theoretical Abstractions

Section:
More General Setting
Data Flow Values
Flow Functions
Solutions
Algorithms
Modelling General Flows
Extra Material

Existence and Computation of the Maximum Fixed Point

- For monotonic $f : L \to L$
  - Existence: $MFP(f) = \bigsqcup Ext(f) \in Fix(f)$
    Requires L to be complete
  - Computation: $MFP(f) = f^{k+1}(\top) = f^{k}(\top)$ such that $f^{j+1}(\top) \neq f^{j}(\top)$, $j < k$.
    Requires all *strictly descending* chains to be finite
- Finite strictly descending and ascending chains

  $\Rightarrow$ Completeness of lattice

Existence and Computation of the Maximum Fixed Point

- For monotonic $f : L \rightarrow L$
  - Existence: $MFP(f) = \bigsqcup Ext(f) \in Fix(f)$
    Requires L to be complete
  - Computation: $MFP(f) = f^{k+1}(\top) = f^k(\top)$ such that $f^{j+1}(\top) \neq f^j(\top)$, $j < k$.
    Requires all *strictly descending* chains to be finite
- Finite strictly descending and ascending chains

  $\Rightarrow$ Completeness of lattice
- Completeness of lattice $\not\Rightarrow$ Finite strictly descending chains

Existence and Computation of the Maximum Fixed Point

- For monotonic $f : L \to L$
  - Existence: $MFP(f) = \bigsqcup Ext(f) \in Fix(f)$
    Requires L to be complete
  - Computation: $MFP(f) = f^{k+1}(\top) = f^{k}(\top)$ such that $f^{j+1}(\top) \neq f^{j}(\top)$, $j < k$.
    Requires all *strictly descending* chains to be finite
- 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

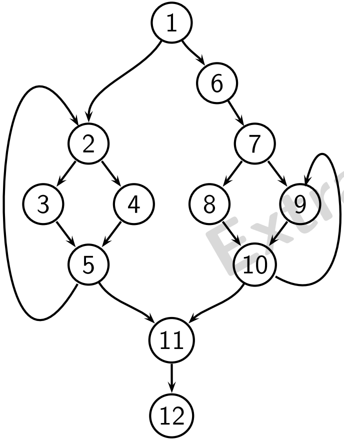
Complexity of Round Robin Iterative Algorithm

- Unidirectional rapid frameworks

Task	Number of iterations	
	Irreducible G	Reducible G
Initialization	1	1
Convergence (until *change* remains true)	$d(G, T) + 1$	$d(G, T)$
Verifying convergence (*change* becomes false)	1	1

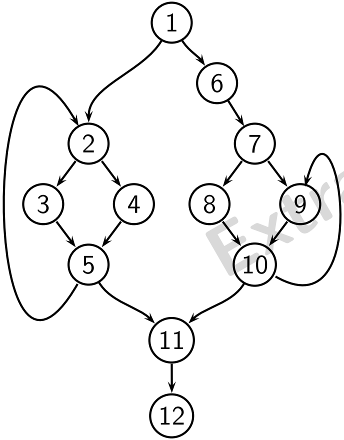
Complexity of Bidirectional Bit Vector Frameworks

Example: Consider the following CFG for PRE



Complexity of Bidirectional Bit Vector Frameworks

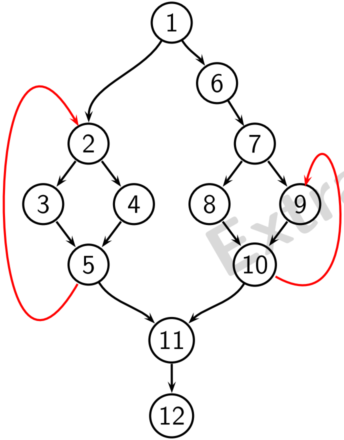
Example: Consider the following CFG for PRE



- Node numbers are in reverse post order

Complexity of Bidirectional Bit Vector Frameworks

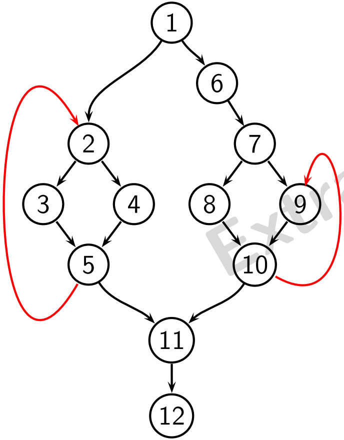
Example: Consider the following CFG for PRE



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

Complexity of Bidirectional Bit Vector Frameworks

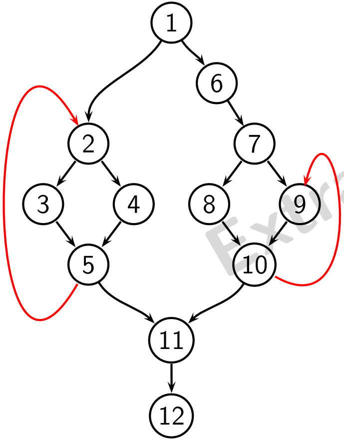
Example: Consider the following CFG for PRE



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

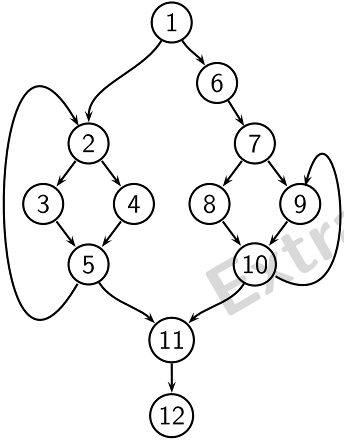
Complexity of Bidirectional Bit Vector Frameworks

Example: Consider the following CFG for PRE



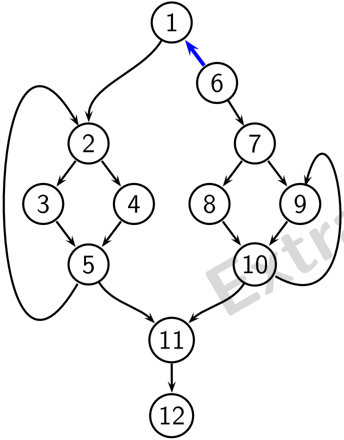
- Node numbers are in reverse post order
- Back edges in the graph are $n_5 \rightarrow n_2$ and $n_{10} \rightarrow n_9$
- $d(G, T) = 1$
- Actual iterations : 5

Information Flow Paths in PRE



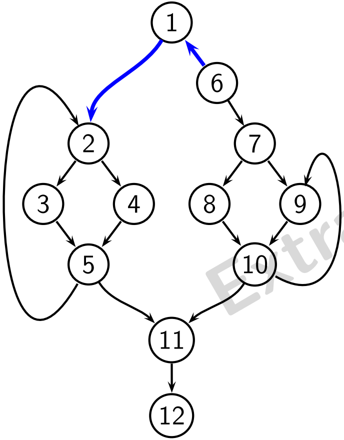
- Information could flow along arbitrary paths

Information Flow Paths in PRE



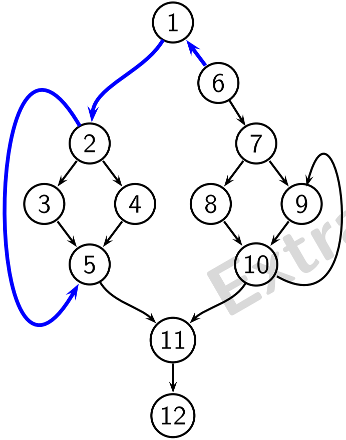
- Information could flow along arbitrary paths

Information Flow Paths in PRE



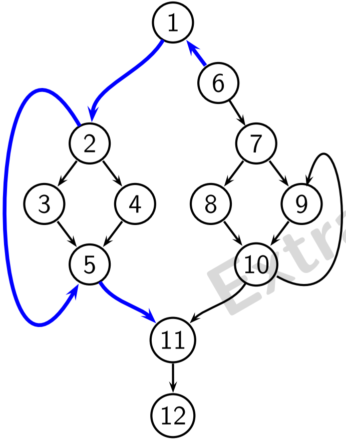
- Information could flow along arbitrary paths

Information Flow Paths in PRE



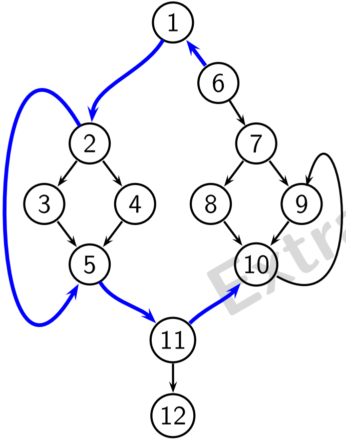
- Information could flow along arbitrary paths

Information Flow Paths in PRE



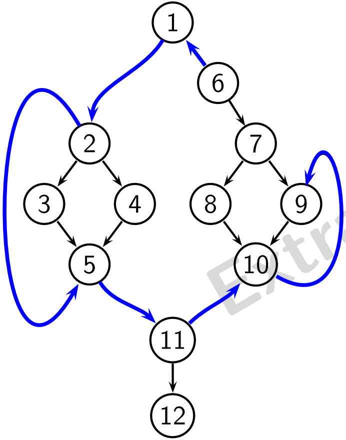
- Information could flow along arbitrary paths

Information Flow Paths in PRE



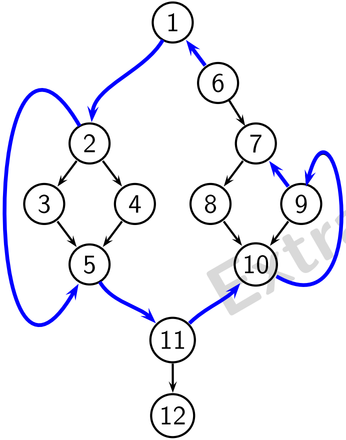
- Information could flow along arbitrary paths

Information Flow Paths in PRE



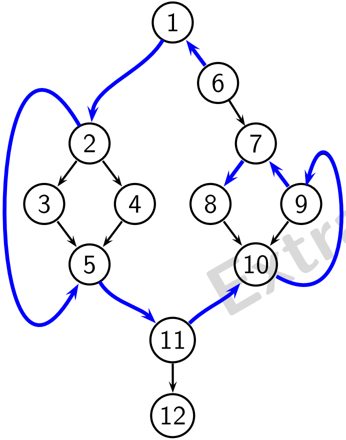
- Information could flow along arbitrary paths

Information Flow Paths in PRE



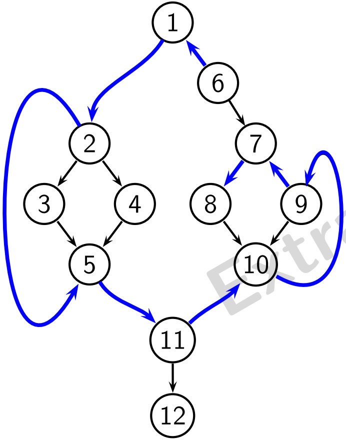
- Information could flow along arbitrary paths

Information Flow Paths in PRE



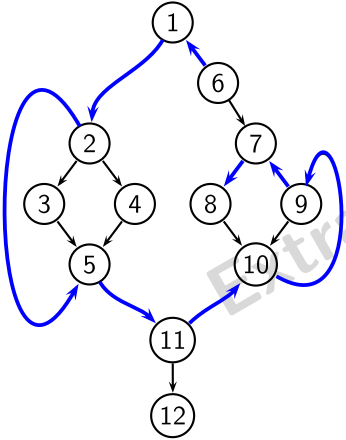
- Information could flow along arbitrary paths
- Theoretically predicted number : 144

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)

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

Complexity of Worklist Algorithms for Bit Vector Frameworks

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

Complexity of Worklist Algorithms for Bit Vector Frameworks

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

Lacuna with Older Estimates of PRE Complexity

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

Lacuna with Older Estimates of PRE Complexity

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

Lacuna with Older Estimates of PRE Complexity

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

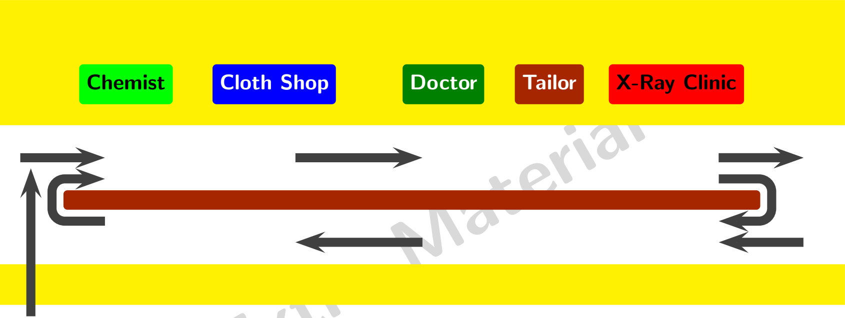
Lacuna with Older Estimates of PRE Complexity

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

Lacuna with Older Estimates of PRE Complexity

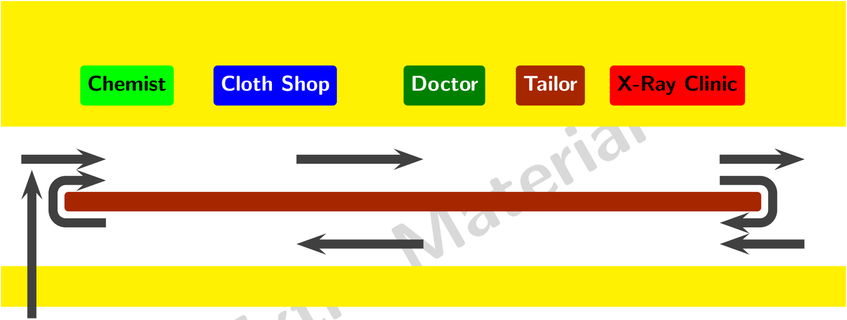
- Lacuna with PRE : Complexity
    - r is typically $\mathcal{O}(n)$
    - Assuming that at most one data flow value changes in one traversal
    - Worst case number of traversals $= \mathcal{O}\left(n^2\right)$
- Practical graphs may have upto 50 nodes
    - Predicted number of traversals : 2,500
 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

Complexity of Round Robin Iterative Method



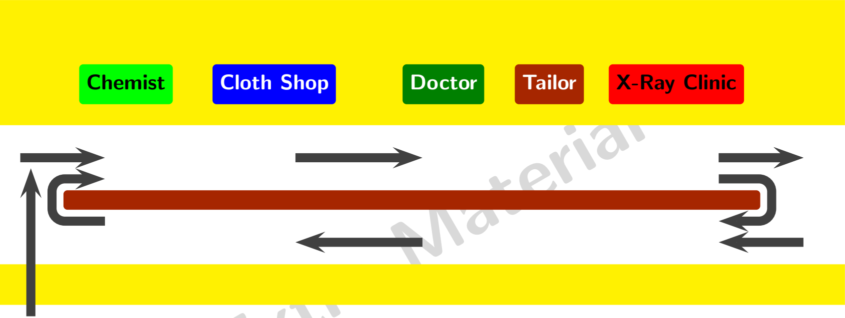
- Buy OTC (Over-The-Counter) medicine   No U-Turn   1 Trip

Complexity of Round Robin Iterative Method



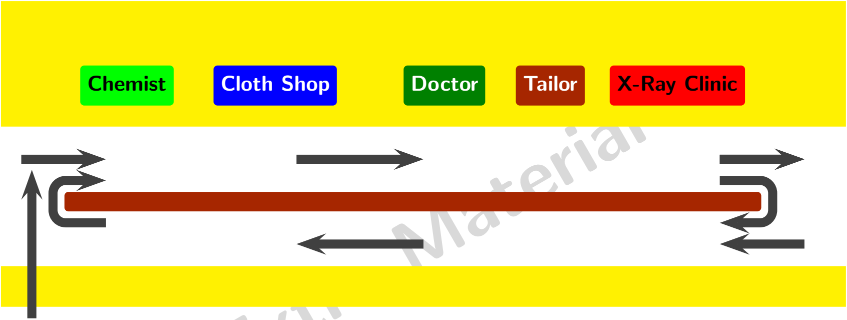
• Buy OTC (Over-The-Counter) medicine	No U-Turn	1 Trip
• Buy cloth. Give it to the tailor for stitching	No U-Turn	1 Trip

Complexity of Round Robin Iterative Method



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

Complexity of Round Robin Iterative Method



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

Information Flow Paths and Width of a Graph

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

Information Flow Paths and Width of a Graph

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

Information Flow Paths and Width of a Graph

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

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

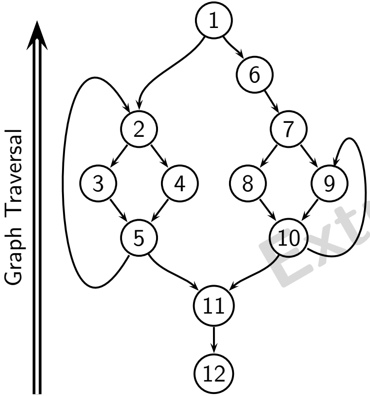
Information Flow Paths and Width of a Graph

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

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

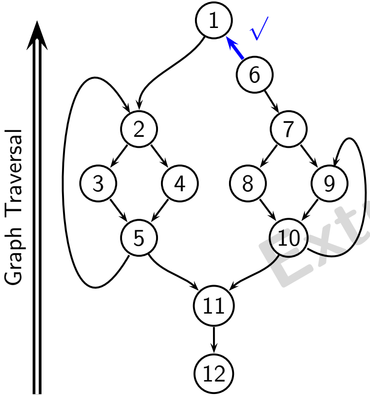
  (1 additional iteration may be required for verifying convergence)

Complexity of Bidirectional Bit Vector Frameworks



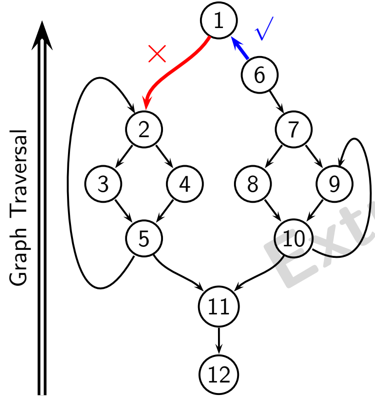
- Every "incompatible" edge traversal
  $\Rightarrow$ **One additional graph traversal**

Complexity of Bidirectional Bit Vector Frameworks



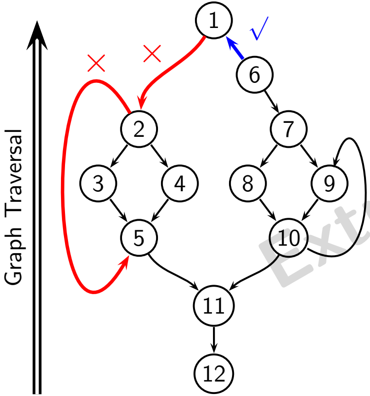
- Every "incompatible" edge traversal $\Rightarrow$ **One additional graph traversal**
- Max. Incompatible edge traversals $=$ *Width* of the graph $=$ **0?**
- Maximum number of traversals $=$ $1 +$ Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



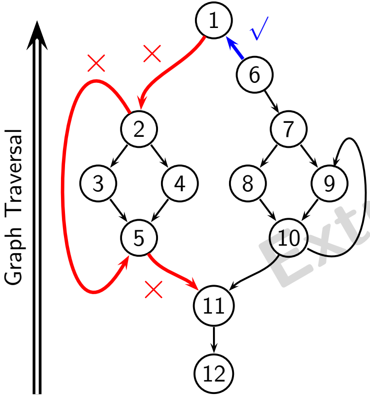
- Every "incompatible" edge traversal
  $\Rightarrow$ **One additional graph traversal**
- Max. Incompatible edge traversals
  $=$ *Width* of the graph $=$ **1?**
- Maximum number of traversals $=$
  $1 +$ Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



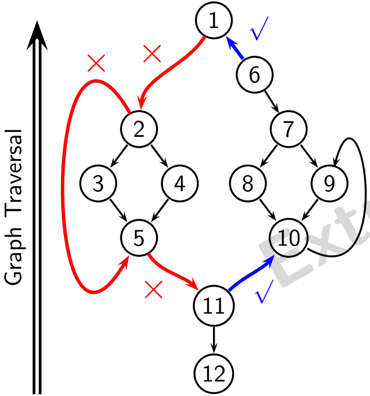
- Every "incompatible" edge traversal
  $\Rightarrow$ **One additional graph traversal**
- Max. Incompatible edge traversals
  = *Width* of the graph = **2?**
- Maximum number of traversals =
  1 + Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



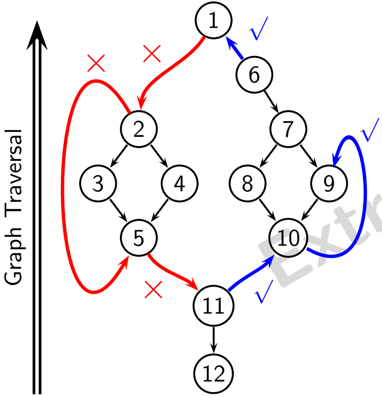
- Every "incompatible" edge traversal
  $\Rightarrow$ **One additional graph traversal**
- Max. Incompatible edge traversals
  = *Width* of the graph = **3?**
- Maximum number of traversals =
  1 + Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



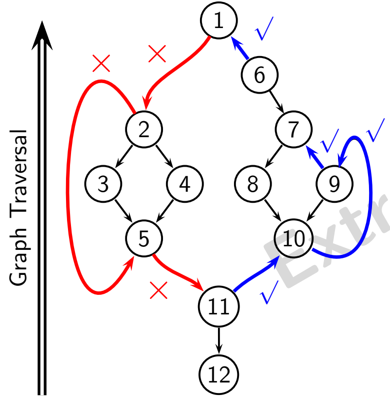
- $\boxed{\text{Every "incompatible" edge traversal} \Rightarrow \textbf{One additional graph traversal}}$
- Max. Incompatible edge traversals $=$ *Width* of the graph $=$ **3?**
- Maximum number of traversals $=$ 1 $+$ Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



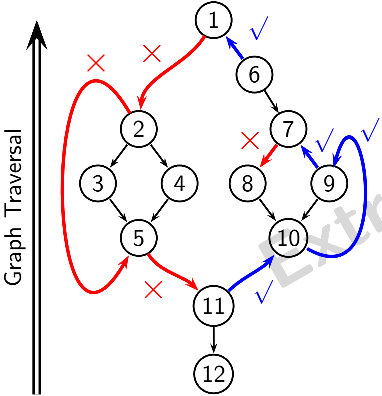
- Every "incompatible" edge traversal
  $\Rightarrow$ **One additional graph traversal**
- Max. Incompatible edge traversals
  = *Width* of the graph = **3?**
- Maximum number of traversals =
  1 + Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



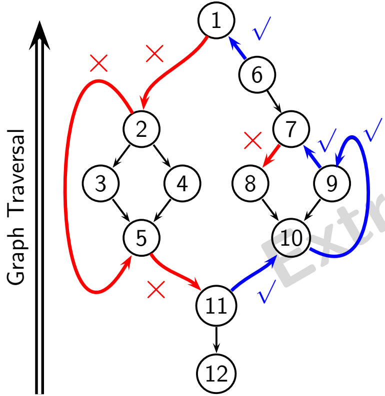
- $\boxed{\text{Every "incompatible" edge traversal } \Rightarrow \textbf{One additional graph traversal}}$
- Max. Incompatible edge traversals = *Width* of the graph = **3?**
- Maximum number of traversals = 1 + Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



- $\boxed{\text{Every "incompatible" edge traversal } \Rightarrow \textbf{ One additional graph traversal}}$
- 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 $=$
  $1 + 4 = 5$

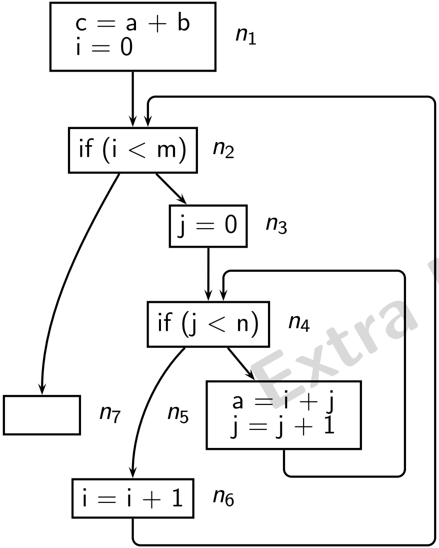
Width Subsumes Depth

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

Comparison Between Width and Depth

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

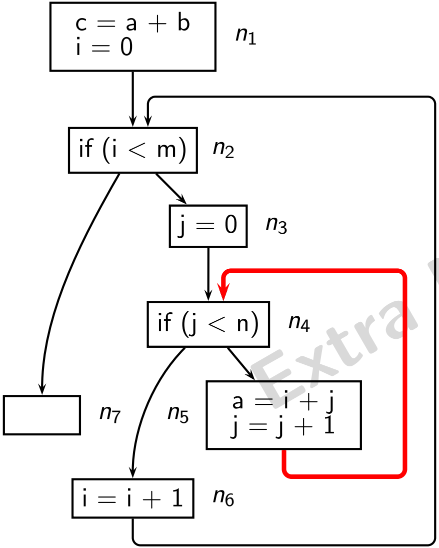
Width and Depth



Assuming reverse postorder traversal for available expressions analysis

- Depth = 2

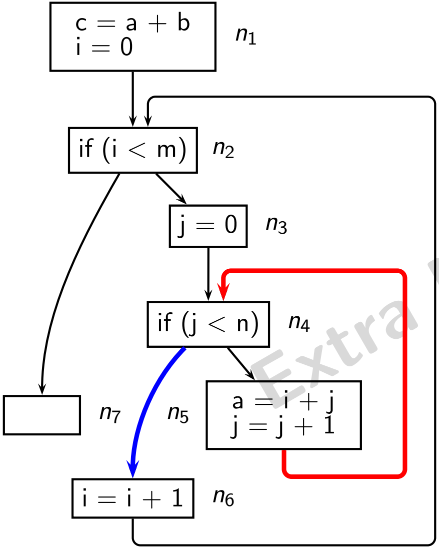
Width and Depth



Assuming reverse postorder traversal for available expressions analysis

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

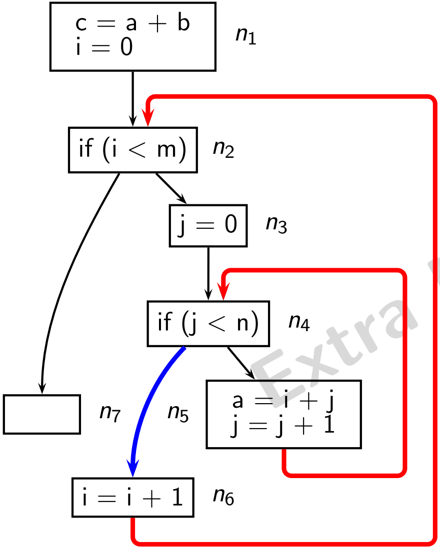
Width and Depth



Assuming reverse postorder traversal for available expressions analysis

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

Width and Depth



Assuming reverse postorder traversal for available expressions analysis

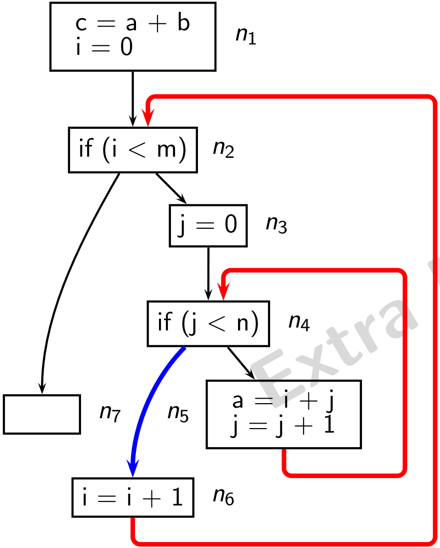
- Depth = 2
- Information generation point

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

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

  No Gen or Kill for "a + b" along this path
- Width = 2

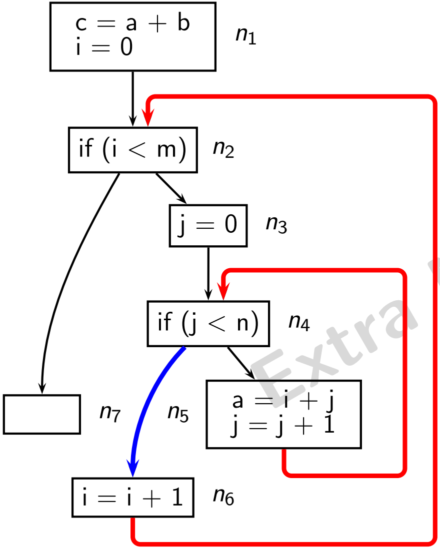
Width and Depth



Assuming reverse postorder traversal for available expressions analysis

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

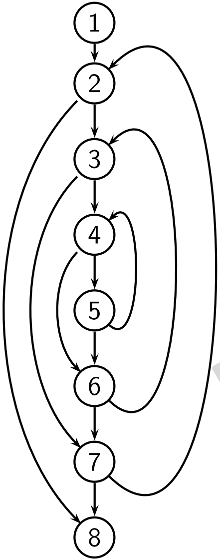
Width and Depth



Assuming reverse postorder traversal for available expressions analysis

- Depth = 2
- Information generation point
  n_5 kills expression "a + b"
- Information propagation path
  $n_5 \rightarrow n_4 \rightarrow n_6 \rightarrow n_2$
  No Gen or Kill for "a + b" along this path
- Width = 2
- What about "j + 1"?
- Not available on entry to the loop

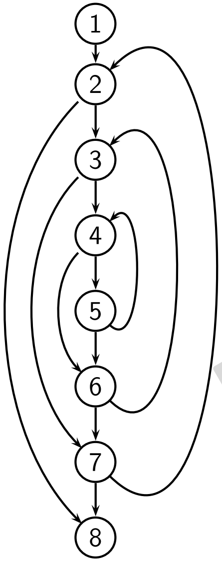
Width and Depth



Structures resulting from repeat-until loops with premature exits

- Depth = 3

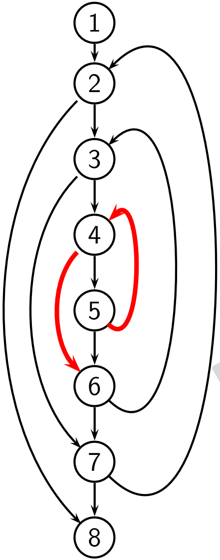
Width and Depth



Structures resulting from repeat-until loops with premature exits

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

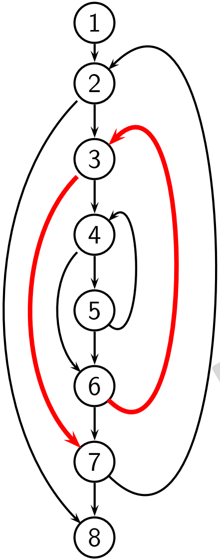
Width and Depth



Structures resulting from repeat-until loops with premature exits

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

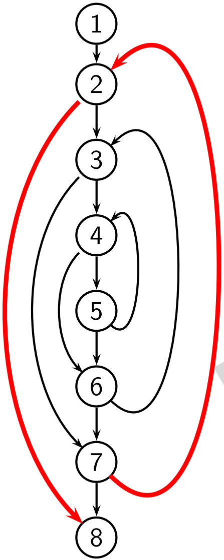
Width and Depth



Structures resulting from repeat-until loops with premature exits

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

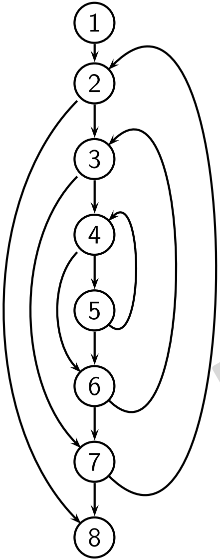
Width and Depth



Structures resulting from repeat-until loops with premature exits

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

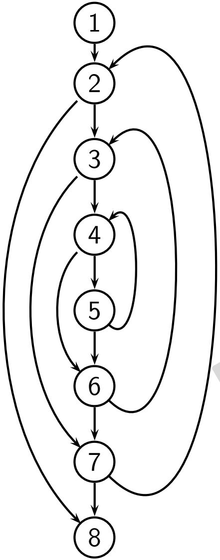
Width and Depth



Structures resulting from repeat-until loops with premature exits

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

Width and Depth



Structures resulting from repeat-until loops with premature exits

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