Further Generalizations

Uday P. Khedker

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

May 2011

Part 1

About These Slides

Copyright

These slides constitute the lecture notes for

- MACS L111 Advanced Data Flow Analysis course at Cambridge University, and
- CS 618 Program Analysis course at IIT Bombay.

They have been made available under GNU FDL v1.2 or later (purely for academic or research use) as teaching material accompanying the book:

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

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

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

Outline

- Partial Redundancy Elimination (previous lecture)
- Introduction to Constant Propagation (previous lecture)
- Theoretical Abstractions in Data Flow Analysis
  - The world of data flow values (previous lecture)
  - The world of functions and operations that compute data values (today)
  - Results of data flow analysis (today)
  - Algorithms for performing data flow analysis (today)
- Precise Modelling of General flows (today)
  Example: Constant Propagation

Part 2

Flow Functions

Flow Functions: An Outline of Our Discussion

- Defining flow functions
- Properties of flow functions
  (Some properties discussed in the context of solutions of data flow analysis)

The Set of Flow Functions

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

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

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

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

- Properties of f
  - Monotonicity and Distributivity
  - 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*

$$f(x) = Gen \cup (x - Kill)$$

  - 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*

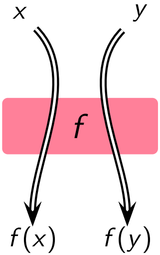
$$f(x) = Gen \cup (x - Kill)$$

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

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

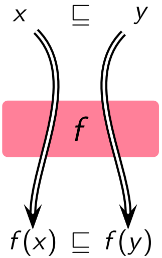
Monotonicity of Flow Functions

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



Monotonicity of Flow Functions

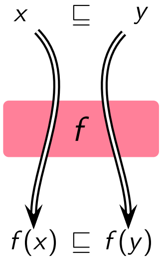
- Partial order is preserved: If x can be safely used in place of y then $f(x)$ can be safely used in place of $f(y)$



Monotonicity of Flow Functions

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

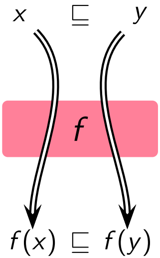
$$\forall x, y \in L, x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$$



Monotonicity of Flow Functions

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

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



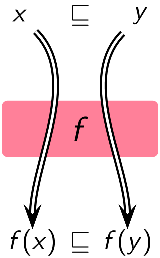
- Alternative definition

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

Monotonicity of Flow Functions

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

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



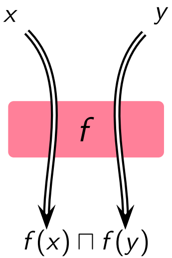
- Alternative definition

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

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

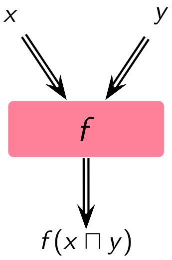
Distributivity of Flow Functions

- Merging distributes over function application



Distributivity of Flow Functions

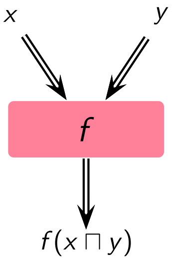
- Merging distributes over function application



Distributivity of Flow Functions

- Merging distributes over function application

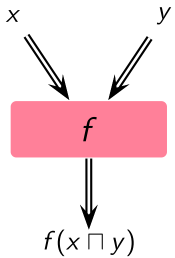
$$\forall x, y \in L, x \sqsubseteq y \Rightarrow f(x \sqcap y) = f(x) \sqcap f(y)$$



Distributivity of Flow Functions

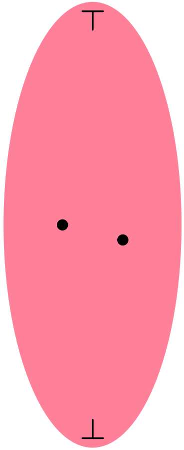
- Merging distributes over function application

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



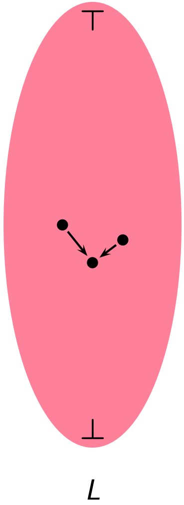
- Merging at intermediate points in shared segments of paths does not lead to imprecision.

Monotonicity and Distributivity

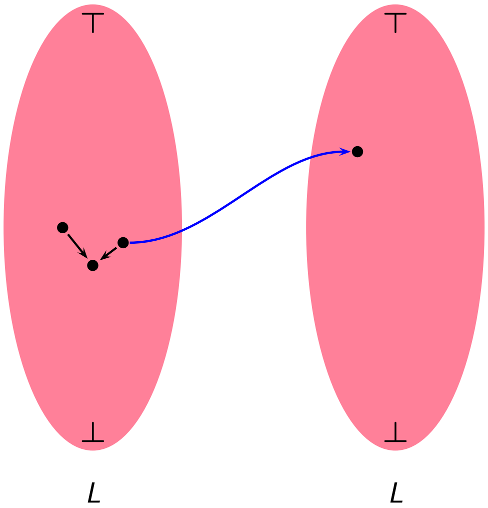


L

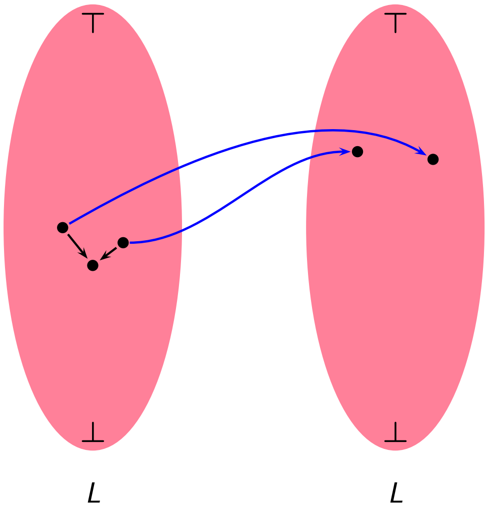
Monotonicity and Distributivity



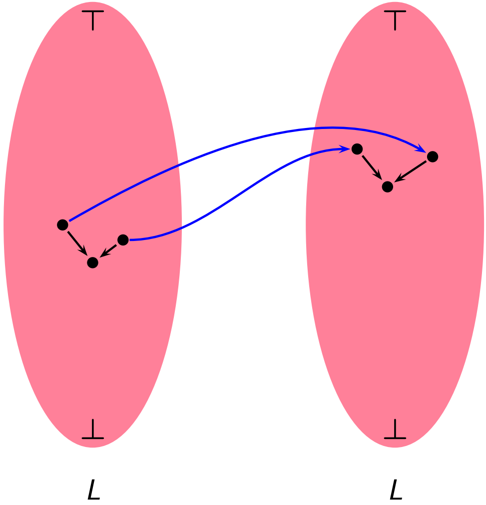
Monotonicity and Distributivity



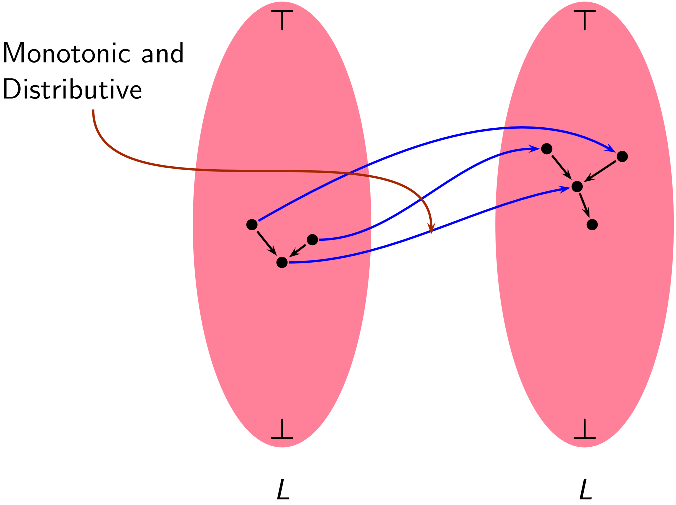
Monotonicity and Distributivity



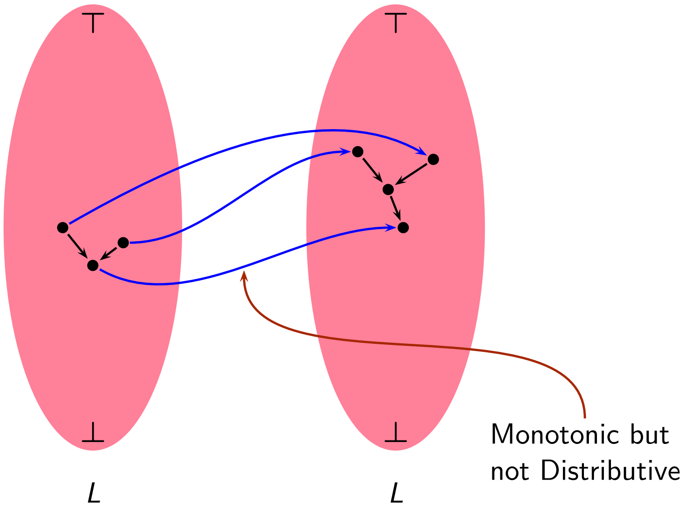
Monotonicity and Distributivity



Monotonicity and Distributivity



Monotonicity and Distributivity



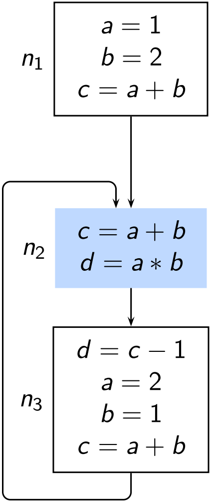
Distributivity of Bit Vector Frameworks

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

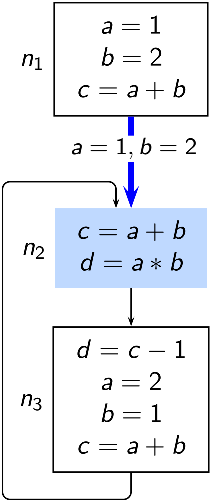
$$
\begin{aligned}
f(x \cup y) &= Gen \cup ((x \cup y) - Kill) \\
&= Gen \cup ((x - Kill) \cup (y - Kill)) \\
&= (Gen \cup (x - Kill) \cup Gen \cup (y - Kill)) \\
&= f(x) \cup f(y)
\end{aligned}
$$

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

Non-Distributivity of Constant Propagation

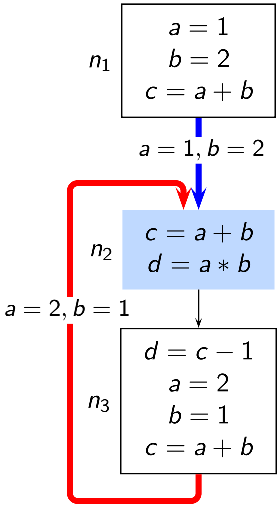


Non-Distributivity of Constant Propagation



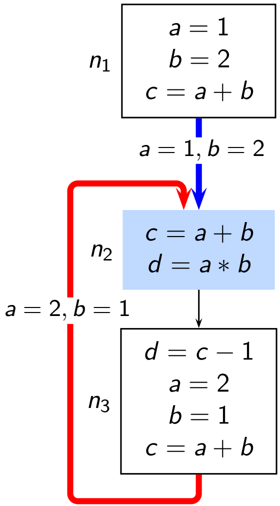
- $x = \langle 1, 2, 3, \mathbf{?} \rangle$ (Along $Out_{n_1} \rightarrow In_{n_2}$)

Non-Distributivity of Constant Propagation



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

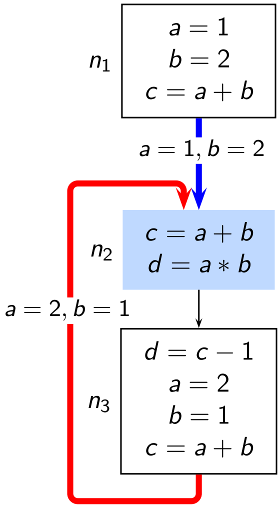
Non-Distributivity of Constant Propagation



- $x = \langle 1, 2, 3, \mathbf{?} \rangle$ (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

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

Non-Distributivity of Constant Propagation



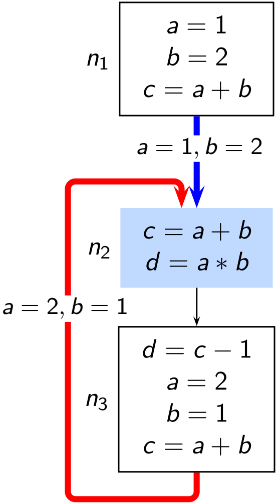
- $x = \langle 1, 2, 3, \mathbf{?} \rangle$ (Along $Out_{n_1} \rightarrow In_{n_2}$)
- $y = \langle 2, 1, 3, 2 \rangle$ (Along $Out_{n_3} \rightarrow In_{n_2}$)
- Function application for block n_2 before merging

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

- Function application for block n_2 after merging

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

Non-Distributivity of Constant Propagation



- $x = \langle 1, 2, 3, \mathbf{?} \rangle$ (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

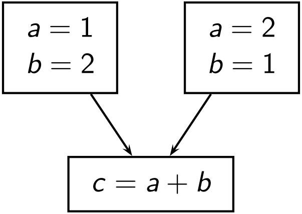
$$\begin{aligned} f(x) \sqcap f(y) &= f(\langle 1, 2, 3, \mathbf{?} \rangle) \sqcap f(\langle 2, 1, 3, 2 \rangle) \\ &= \langle 1, 2, 3, 2 \rangle \sqcap \langle 2, 1, 3, 2 \rangle \\ &= \langle \widehat{\perp}, \widehat{\perp}, 3, 2 \rangle \end{aligned}$$

- Function application for block n_2 after merging

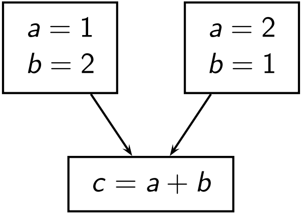
$$\begin{aligned} f(x \sqcap y) &= f(\langle 1, 2, 3, \mathbf{?} \rangle \sqcap \langle 2, 1, 3, 2 \rangle) \\ &= f(\langle \widehat{\perp}, \widehat{\perp}, 3, 2 \rangle) \\ &= \langle \widehat{\perp}, \widehat{\perp}, \widehat{\perp}, \widehat{\perp} \rangle \end{aligned}$$

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

Why is Constant Propagation Non-Distribitive?



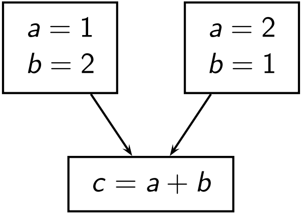
Why is Constant Propagation Non-Distribitive?



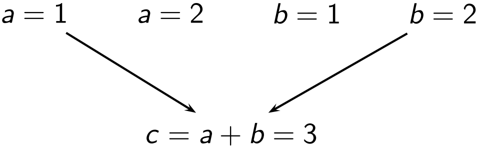
Possible combinations due to merging

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

Why is Constant Propagation Non-Distribitive?

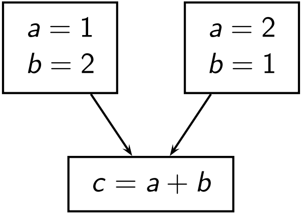


Possible combinations due to merging

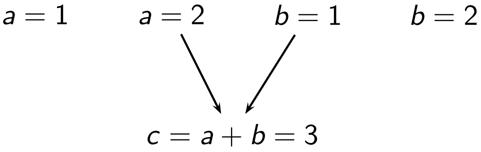


- Correct combination.

Why is Constant Propagation Non-Distribitive?

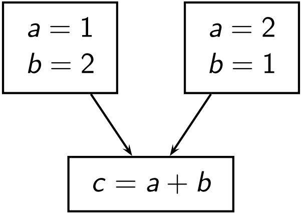


Possible combinations due to merging

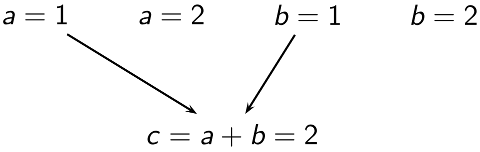


- Correct combination.

Why is Constant Propagation Non-Distribitive?

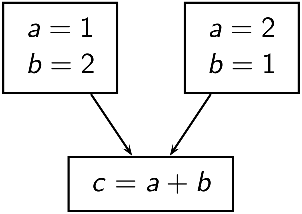


Possible combinations due to merging

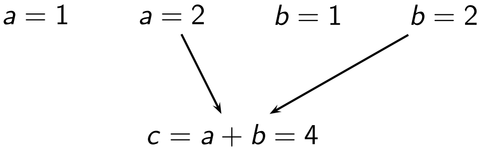


- Wrong combination.
- Mutually exclusive information.
- No execution path along which this information holds.

Why is Constant Propagation Non-Distribitive?



Possible combinations due to merging



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

Part 3

Solutions of Data Flow Analysis

Solutions of Data Flow Analysis: An Outline of Our Discussion

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

Solutions of Data Flow Analysis

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

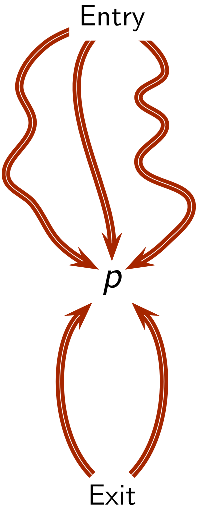
  Given

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

  Find out

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

Meet Over Paths (MoP) Assignment

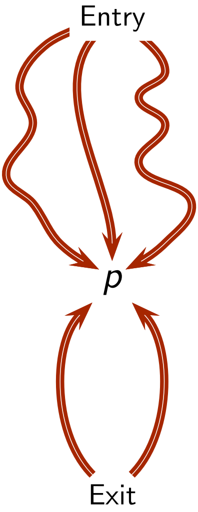


- The largest safe approximation of the information reaching a program point along all information flow paths.

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

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

Meet Over Paths (MoP) Assignment



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

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

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

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

Maximum Fixed Point (MFP) Assignment

- Difficulties in computing MoP assignment

Maximum Fixed Point (MFP) Assignment

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

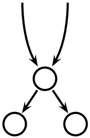
    If all paths need to be traversed $\Rightarrow$ Undecidability

Maximum Fixed Point (MFP) Assignment

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

    If all paths need to be traversed $\Rightarrow$ 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

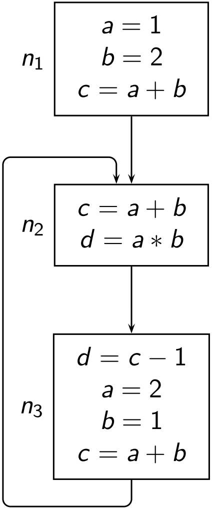
    If all paths need to be traversed $\Rightarrow$ Undecidability
  - Even if a program is acyclic, every conditional multiplies the number of paths by two

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

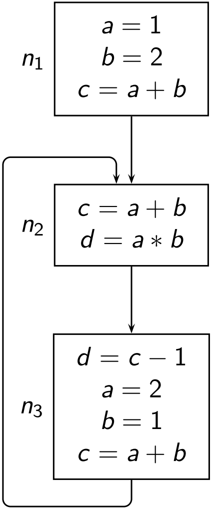
Path based specification

Edge based specifications

Assignments for Constant Propagation Example



Assignments for Constant Propagation Example



MoP

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

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

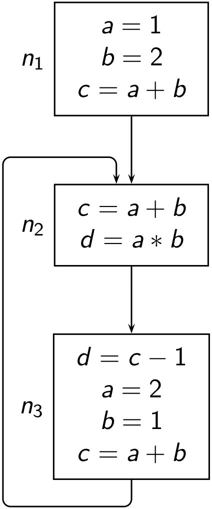
$\langle \widehat{\bot}, \widehat{\bot}, 3, 2 \rangle$

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

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

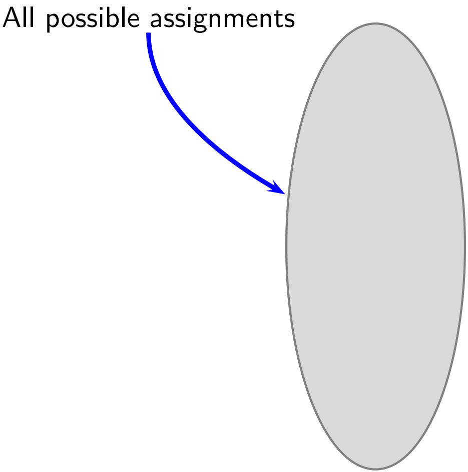
$\langle 2, 1, 3, 2 \rangle$

Assignments for Constant Propagation Example

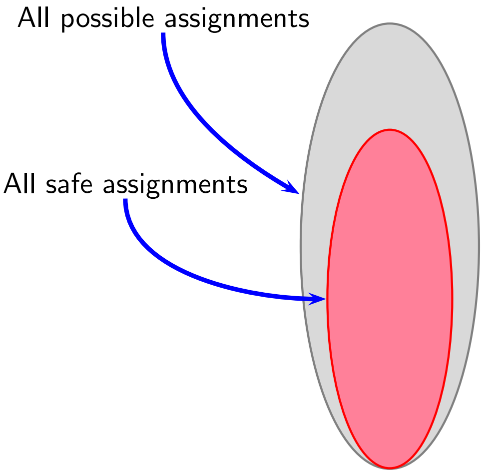


MoP	MFP
$\langle \widehat{\top}, \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle$	$\langle \widehat{\top}, \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle$
$\langle 1, 2, 3, \widehat{\top} \rangle$	$\langle 1, 2, 3, \widehat{\top} \rangle$
$\langle \widehat{\perp}, \widehat{\perp}, 3, 2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

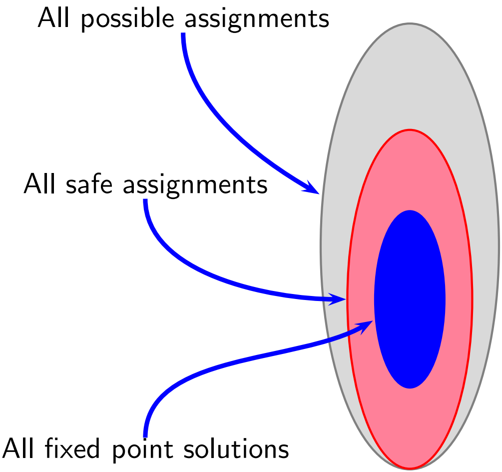
Possible Assignments as Solutions of Data Flow Analyses



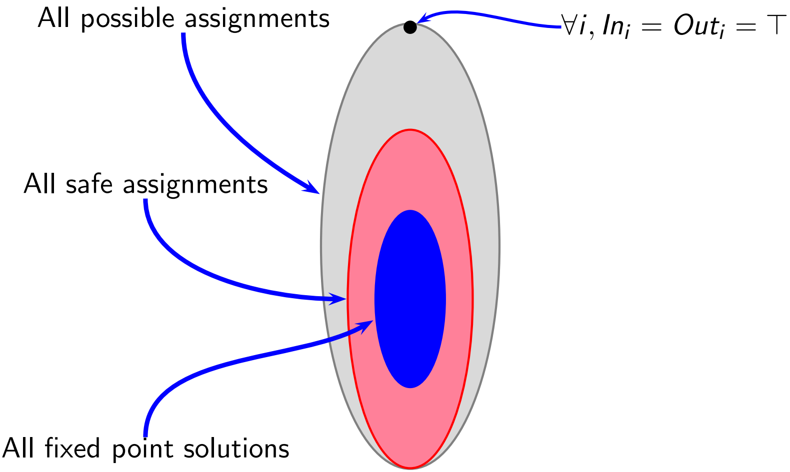
Possible Assignments as Solutions of Data Flow Analyses



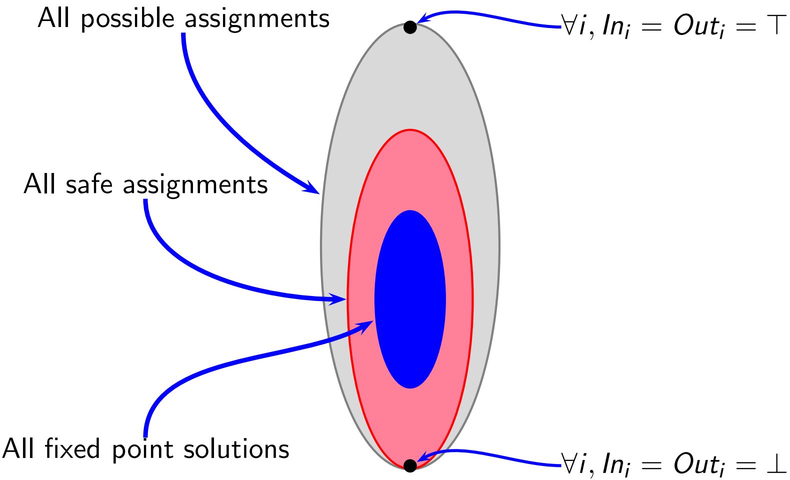
Possible Assignments as Solutions of Data Flow Analyses



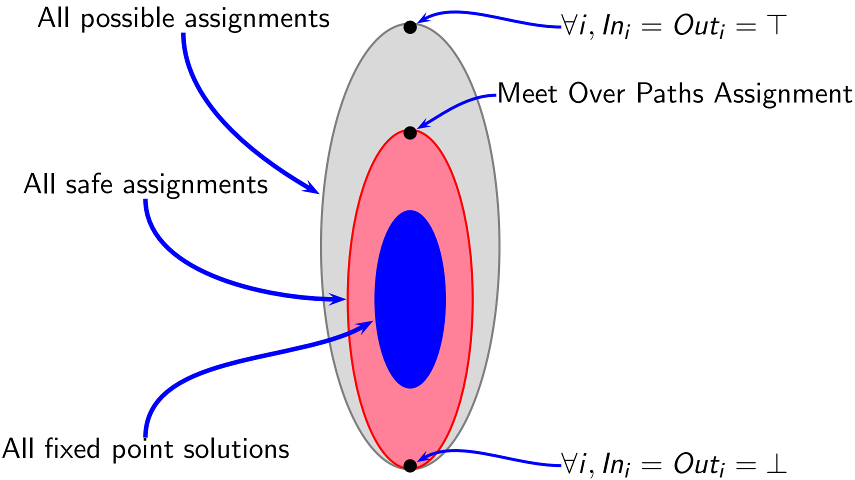
Possible Assignments as Solutions of Data Flow Analyses



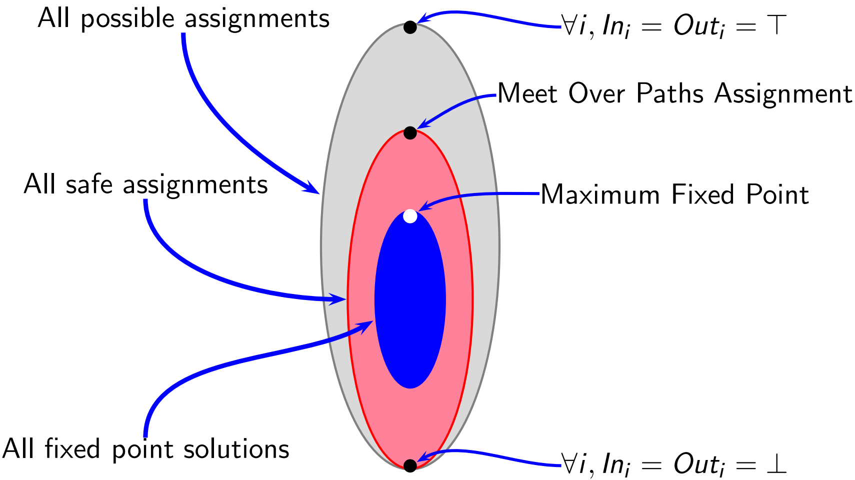
Possible Assignments as Solutions of Data Flow Analyses



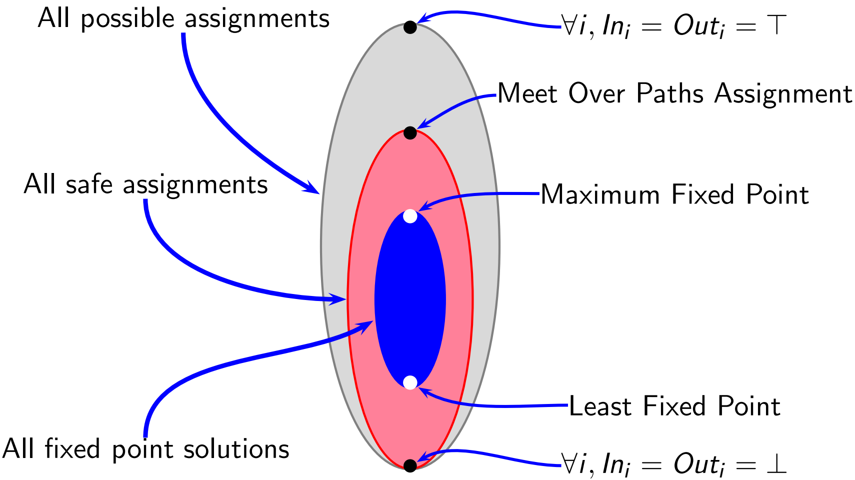
Possible Assignments as Solutions of Data Flow Analyses



Possible Assignments as Solutions of Data Flow Analyses



Possible Assignments as Solutions of Data Flow Analyses



Available Expr. Analysis Framework with Two Expressions

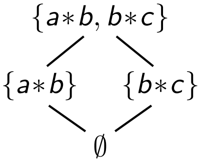
Lattice

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

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

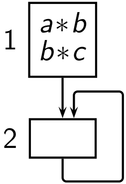
Available Expr. Analysis Framework with Two Expressions

Lattice



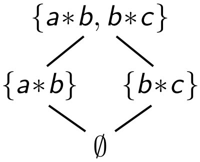
Constant Functions		Dependent Functions	
f	$f(x)$	f	$f(x)$
$f_\top$	$\{a*b, b*c\}$	f_{id}	x
$f_\bot$	$\emptyset$	f_c	$x \cup \{a*b\}$
f_a	$\{a*b\}$	f_d	$x \cup \{b*c\}$
f_b	$\{b*c\}$	f_e	$x - \{a*b\}$
		f_f	$x - \{b*c\}$

Program



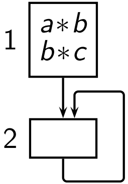
Available Expr. Analysis Framework with Two Expressions

Lattice



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

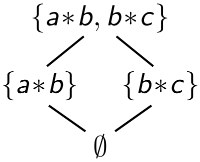
Program



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

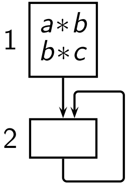
Available Expr. Analysis Framework with Two Expressions

Lattice



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

Program

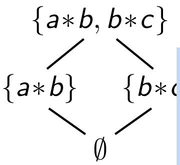


Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

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

Available Expr. Analysis Framework with Two Expressions

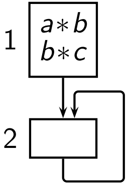
Lattice



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

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

Program

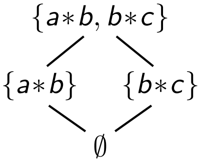


Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

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

Available Expr. Analysis Framework with Two Expressions

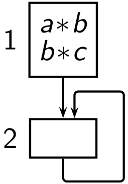
Lattice



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

- Not a fixed point assignment

Program

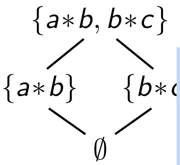


Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

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

Available Expr. Analysis Framework with Two Expressions

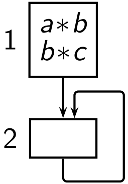
Lattice



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

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

Program

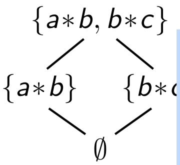


Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

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

Available Expr. Analysis Framework with Two Expressions

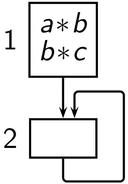
Lattice



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

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

Program

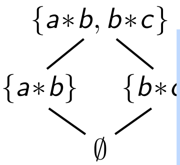


Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

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

Available Expr. Analysis Framework with Two Expressions

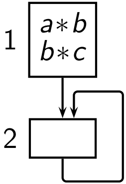
Lattice



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

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

Program

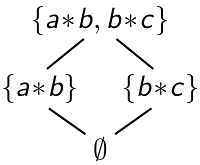


Flow Functions	
Node	Flow Function
1	$f_\top$
2	f_{id}

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

Available Expr. Analysis Framework with Two Expressions

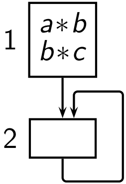
Lattice



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

- Not a fixed point assignment

Program



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

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

Part 4

Performing Data Flow Analysis

Performing Data Flow Analysis

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

Iterative Methods of Performing Data Flow Analysis

Successive recomputation after conservative initialization ($\top$)

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

  Termination : After values stabilise

  + Simplest to understand and implement
  − May perform unnecessary computations

Iterative Methods of Performing Data Flow Analysis

Successive recomputation after conservative initialization ($\top$)

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

  Termination : After values stabilise

  + Simplest to understand and implement
  - May perform unnecessary computations

Our examples use this method.

Iterative Methods of Performing Data Flow Analysis

Successive recomputation after conservative initialization ($\top$)

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

  Termination : After values stabilise

  + Simplest to understand and implement
  − May perform unnecessary computations

Our examples use this method.

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

  Termination : When the list becomes empty

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

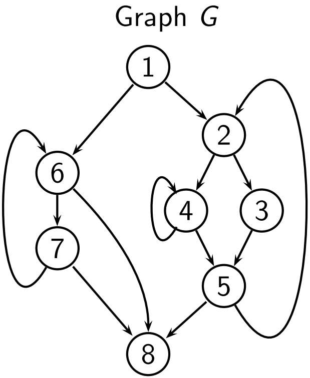
Elimination Methods of Performing Data Flow Analysis

Delayed computations of dependent data flow values of dependent nodes.

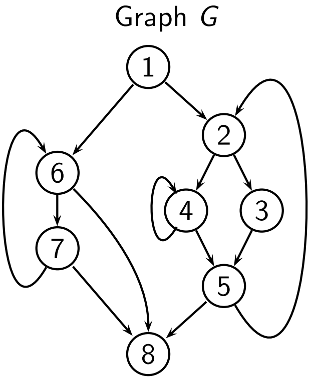
Find suitable single-entry regions.

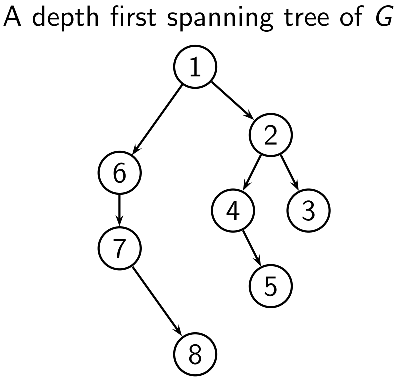
- *Interval Based Analysis*. Uses graph partitioning.
- T_1, T_2 *Based Analysis*. Uses graph parsing.

Classification of Edges in a Graph

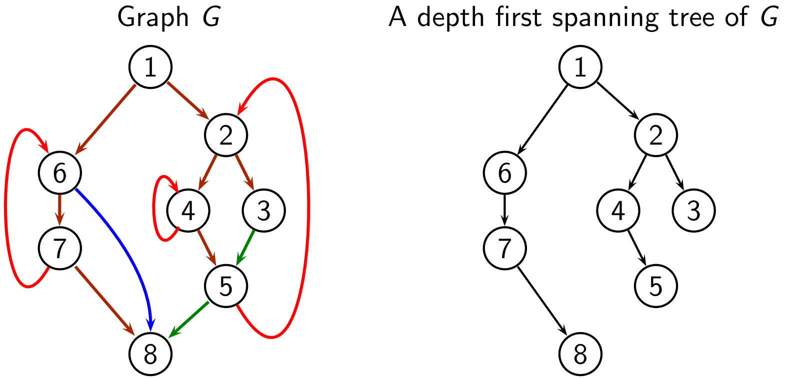


Classification of Edges in a Graph





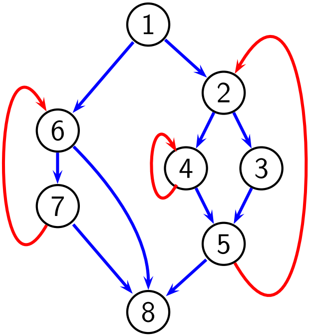
Classification of Edges in a Graph



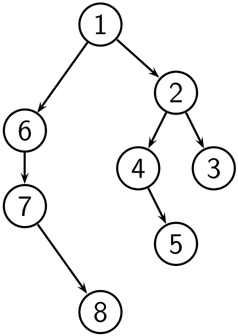
- Back edges
- Forward edges
- Tree edges
- Cross edges

Classification of Edges in a Graph

Graph G



A depth first spanning tree of G



Back edges

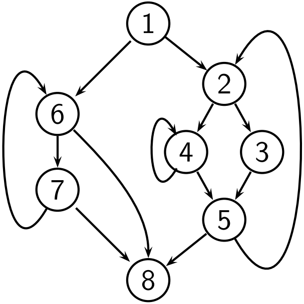
Forward edges

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

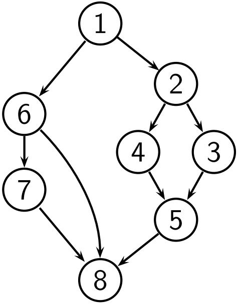
Reverse Post Order Traversal

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

Graph G



G' obtained after removing back edges of G



- Some possible RPOs for G are: $(1, 2, 3, 4, 5, 6, 7, 8)$, $(1, 6, 7, 2, 3, 4, 5, 8)$, $(1, 6, 2, 7, 4, 3, 5, 8)$, 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 $\quad \{ \quad temp = \bigsqcap_{p \in pred(j)} f_p(In_p)$
9 $\quad\quad$ **if** $temp \neq In_j$ **then**
10 $\quad\quad \{ \quad In_j = temp$
11 $\quad\quad\quad change = true$
12 $\quad\quad \}$
13 $\quad \}$
14 $\}$

- Computation of Out_j has been left implicit
  Works fine for unidirectional frameworks
- $\top$ is the identity of $\sqcap$ (line 3)

Round Robin Iterative Algorithm

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

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

Round Robin Iterative Algorithm

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

- Computation of Out_j has been left implicit
  Works fine for unidirectional frameworks
- $\top$ is the identity of $\sqcap$ (line 3)
- Reverse postorder (rpo) traversal for efficiency (line 7)
- rpo traversal AND no loops $\Rightarrow$ no need of initialization

Complexity of Round Robin Iterative Algorithm

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

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

Complexity of Round Robin Iterative Algorithm

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

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

- What about bidirectional bit vector frameworks?

Complexity of Round Robin Iterative Algorithm

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

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

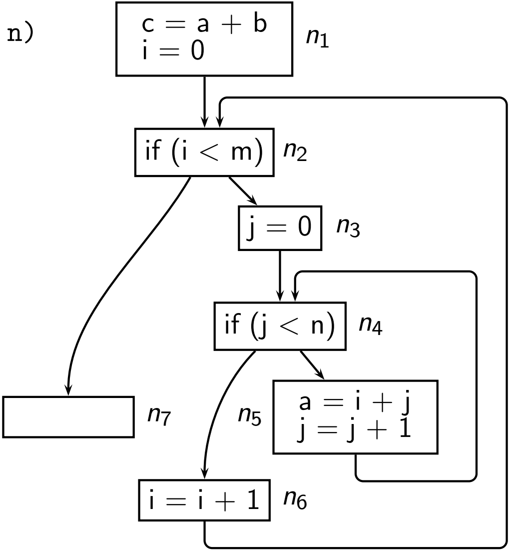
- What about bidirectional bit vector frameworks?
- What about other frameworks?

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

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

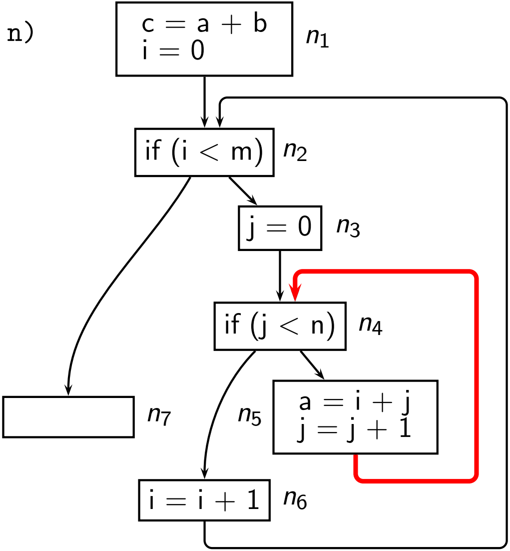
Example C Program with d(G,T) = 2

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



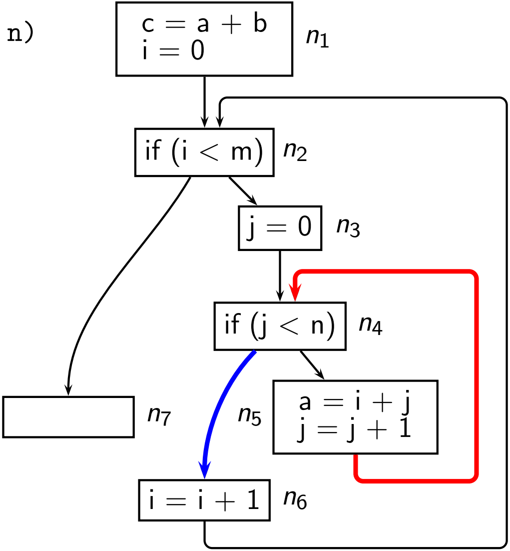
Example C Program with d(G,T) = 2

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



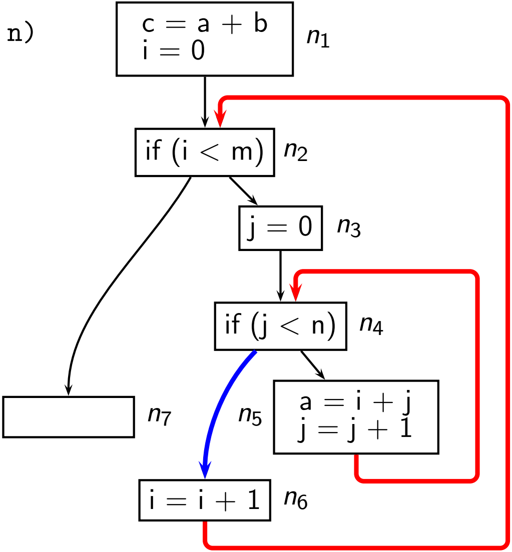
Example C Program with d(G,T) = 2

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



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

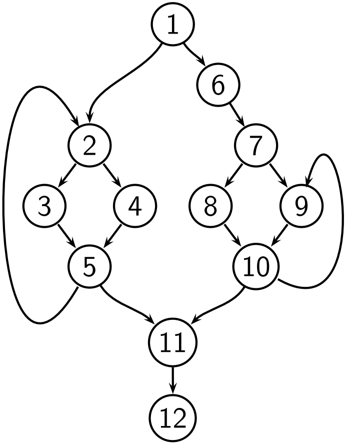
```
void fun(int m, int n)
{
   int i,j,a,b,c;
   c=a+b;
   i=0;
   while(i<m)
   {
         j=0;
         while(j<n)
         {
            a=i+j;
            j=j+1;
         }
         i=i+1;
      }
   }
```



3 + 1 iterations for available expressions analysis

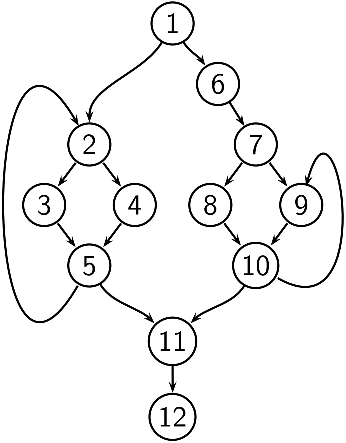
Complexity of Bidirectional Bit Vector Frameworks

Example: Consider the following CFG for PRE



Complexity of Bidirectional Bit Vector Frameworks

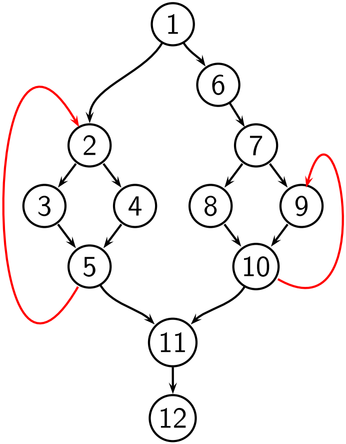
Example: Consider the following CFG for PRE



- Node numbers are in reverse post order

Complexity of Bidirectional Bit Vector Frameworks

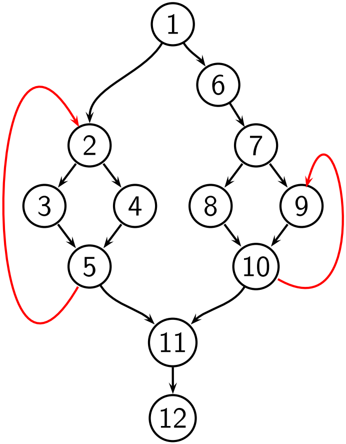
Example: Consider the following CFG for PRE



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

Complexity of Bidirectional Bit Vector Frameworks

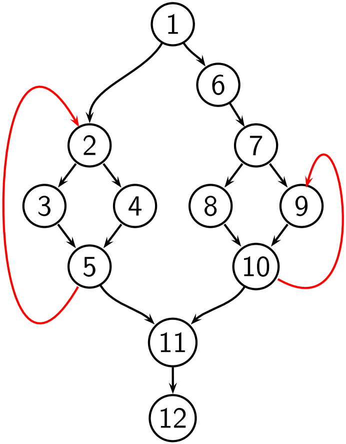
Example: Consider the following CFG for PRE



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

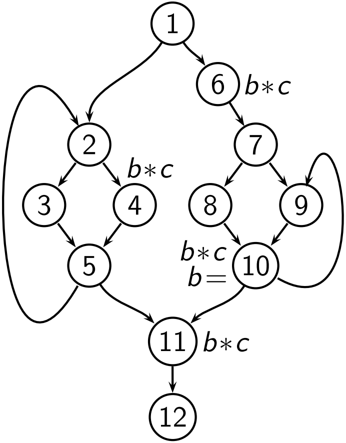
Complexity of Bidirectional Bit Vector Frameworks

Example: Consider the following CFG for PRE



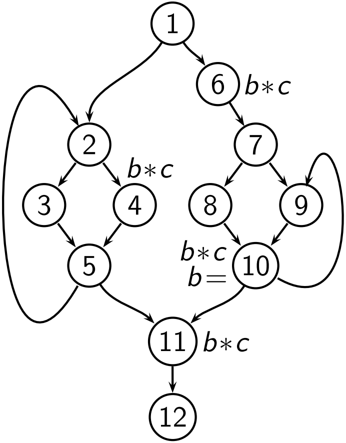
- Node numbers are in reverse post order
- Back edges in the graph are $n_5 \rightarrow n_2$ and $n_{10} \rightarrow n_9$.
- $d(G, T) = 1$
- Actual iterations : 5

Complexity of Bidirectional Bit Vector Frameworks



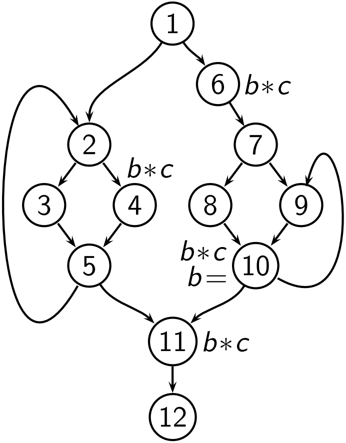
	Pairs of *Out*,*In* Values							
	Initia-lization	Changes in Iterations					Final values & transformation	
		#1	#2	#3	#4	#5		
	O,I	O,I	O,I	O,I	O,I	O,I	O,I	
12	0,1							
11	1,1							
10	1,1							
9	1,1							
8	1,1							
7	1,1							
6	1,1							
5	1,1							
4	1,1							
3	1,1							
2	1,1					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

Complexity of Bidirectional Bit Vector Frameworks



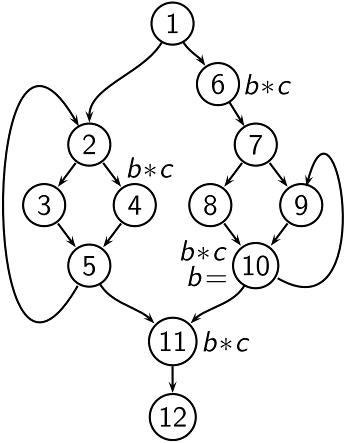
	Pairs of *Out*,*In* Values							
	Initia-lization	Changes in Iterations					Final values & transformation	
		#1	#2	#3	#4	#5		
	O,I	O,I	O,I	O,I	O,I	O,I	O,I	
12	0,1	0,0						
11	1,1	0,1						
10	1,1							
9	1,1							
8	1,1							
7	1,1							
6	1,1	1,0						
5	1,1							
4	1,1							
3	1,1							
2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

Complexity of Bidirectional Bit Vector Frameworks



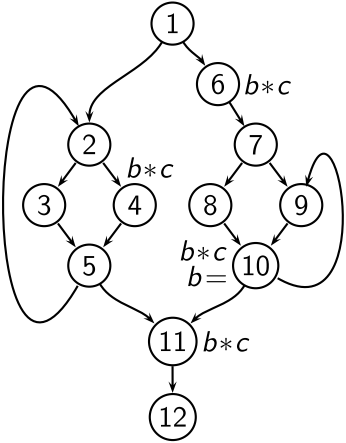
	Pairs of *Out*,*In* Values							
	Initia-lization	Changes in Iterations					Final values & transformation	
		#1	#2	#3	#4	#5		
	O,I	O,I	O,I	O,I	O,I	O,I	O,I	
12	0,1	0,0						
11	1,1	0,1						
10	1,1							
9	1,1							
8	1,1							
7	1,1							
6	1,1	1,0						
5	1,1							
4	1,1							
3	1,1							
2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

Complexity of Bidirectional Bit Vector Frameworks



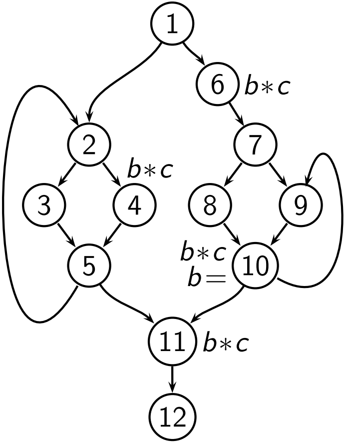
	Pairs of *Out*,*In* Values							
	Initia-lization	Changes in Iterations #1	#2	#3	#4	#5	Final values & transformation	
	O,I	O,I	O,I	O,I	O,I	O,I	O,I	
12	0,1	0,0						
11	1,1	0,1						
10	1,1							
9	1,1							
8	1,1							
7	1,1							
6	1,1	1,0						
5	1,1			0,0				
4	1,1			0,1				
3	1,1			0,0				
2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

Complexity of Bidirectional Bit Vector Frameworks



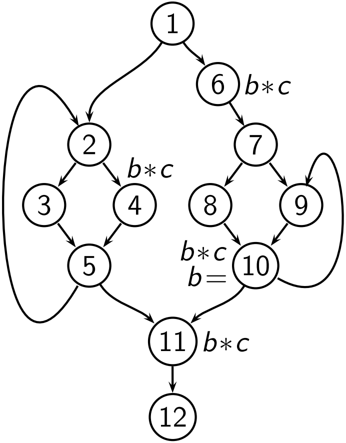
	Pairs of *Out*,*In* Values							
	Initia-lization	Changes in Iterations					Final values & transformation	
		#1	#2	#3	#4	#5		
	O,I	O,I	O,I	O,I	O,I	O,I	O,I	
12	0,1	0,0						
11	1,1	0,1			0,0			
10	1,1				0,1			
9	1,1				1,0			
8	1,1							
7	1,1				0,0			
6	1,1	1,0			0,0			
5	1,1			0,0				
4	1,1			0,1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

Complexity of Bidirectional Bit Vector Frameworks



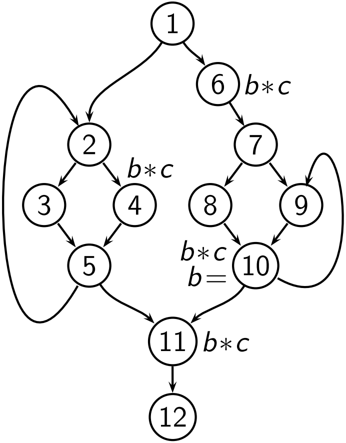
	Pairs of *Out*,*In* Values							
	Initia-lization	Changes in Iterations					Final values & transformation	
		#1	#2	#3	#4	#5		
	O,I	O,I	O,I	O,I	O,I	O,I	O,I	
12	0,1	0,0						
11	1,1	0,1			0,0			
10	1,1				0,1			
9	1,1				1,0			
8	1,1					1,0		
7	1,1				0,0			
6	1,1	1,0			0,0			
5	1,1			0,0				
4	1,1	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

Complexity of Bidirectional Bit Vector Frameworks



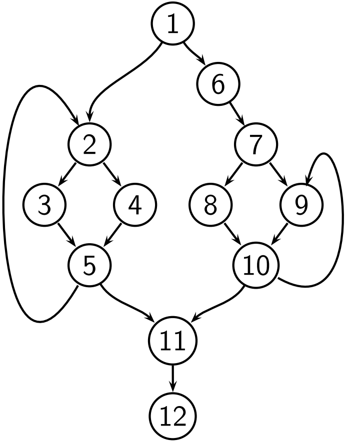
	Pairs of *Out*,*In* Values							
	Initia-lization	Changes in Iterations					Final values & transformation	
		#1	#2	#3	#4	#5		
	O,I	O,I	O,I	O,I	O,I	O,I	O,I	
12	0,1	0,0					0,0	
11	1,1	0,1			0,0		0,0	
10	1,1				0,1		0,1	
9	1,1				1,0		1,0	
8	1,1					1,0	1,0	
7	1,1				0,0		0,0	
6	1,1	1,0			0,0		0,0	
5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

Complexity of Bidirectional Bit Vector Frameworks



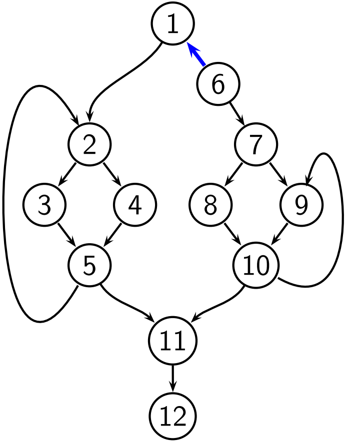
	Pairs of *Out*,*In* Values							
	Initia-lization	Changes in Iterations					Final values & transformation	
		#1	#2	#3	#4	#5		
	O,I	O,I	O,I	O,I	O,I	O,I	O,I	
12	0,1	0,0					0,0	
11	1,1	0,1			0,0		0,0	
10	1,1				0,1		0,1	Delete
9	1,1				1,0		1,0	Insert
8	1,1					1,0	1,0	Insert
7	1,1				0,0		0,0	
6	1,1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

An Example of Information Flow in Our PRE Analysis



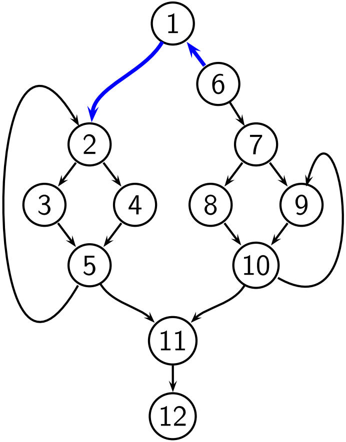
- $PavIn_6$ becomes 0 in the first itereation
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)

An Example of Information Flow in Our PRE Analysis



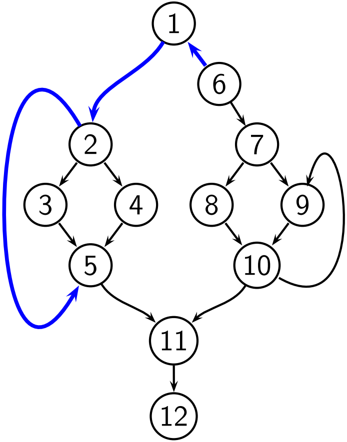
- $PavIn_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)

An Example of Information Flow in Our PRE Analysis



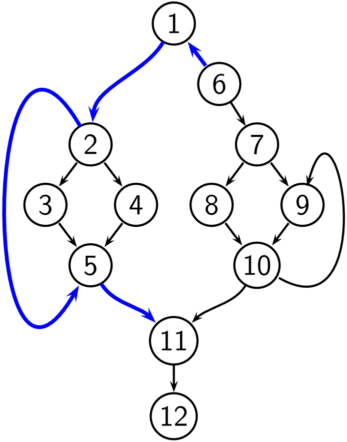
- $PavIn_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)

An Example of Information Flow in Our PRE Analysis



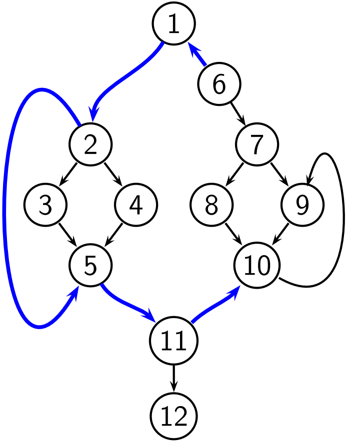
- $PavIn_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)

An Example of Information Flow in Our PRE Analysis



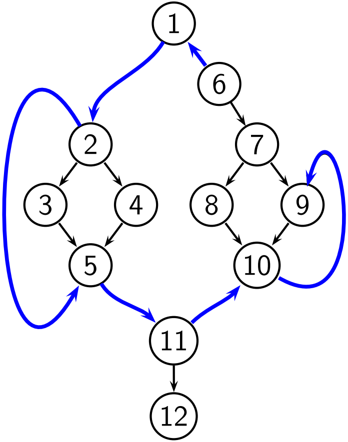
- $PavIn_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)

An Example of Information Flow in Our PRE Analysis



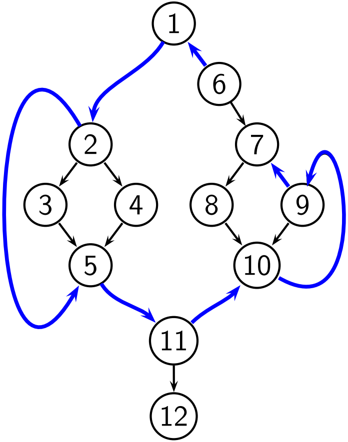
- $PavIn_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)

An Example of Information Flow in Our PRE Analysis



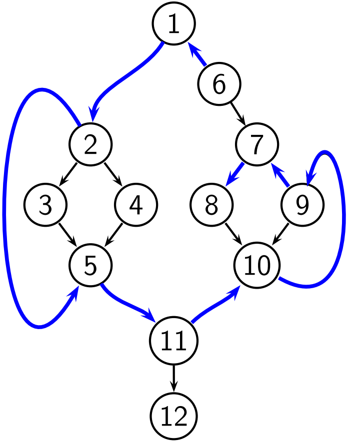
- $PavIn_6$ becomes 0 in the first iteation
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)

An Example of Information Flow in Our PRE Analysis



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

An Example of Information Flow in Our PRE Analysis



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

Information Flow and Information Flow Paths

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

Information Flow and Information Flow Paths

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

  Sequence of adjacent program points

Information Flow and Information Flow Paths

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

  Sequence of adjacent program points
  along which data flow values change

Information Flow and Information Flow Paths

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

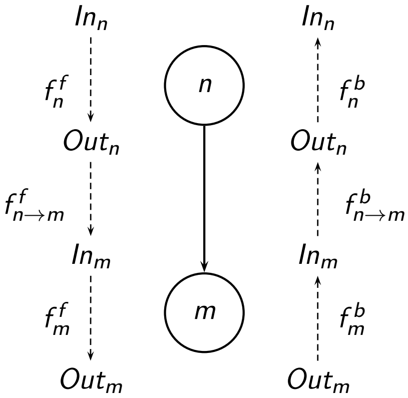
  Sequence of adjacent program points
  along which data flow values change
- A change in the data flow at a program point could be
  - *Generation of information*
    Change from $\top$ to a non-$\top$ due to local effect (i.e. $f(\top) \neq \top$)
  - *Propagation of information*
    Change from x to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

Information Flow and Information Flow Paths

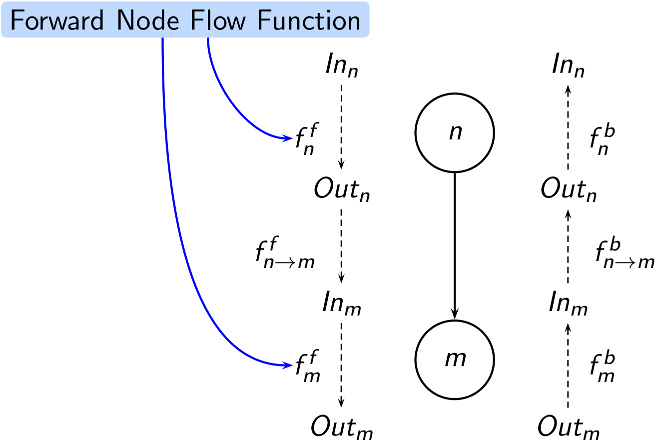
- Default value at each program point: $\top$
- *Information flow path*

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

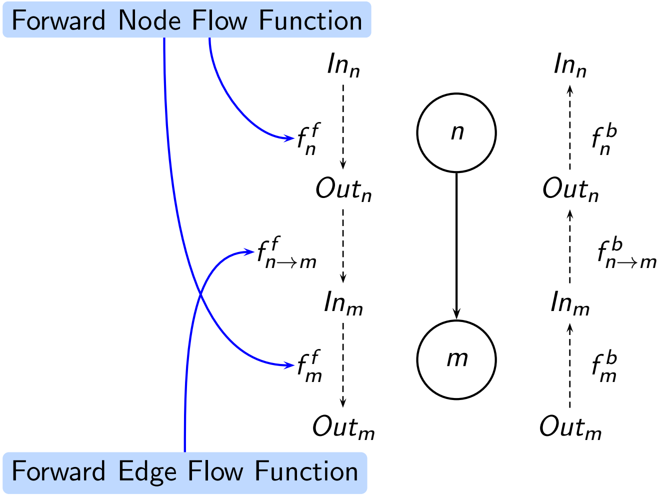
Edge and Node Flow Functions



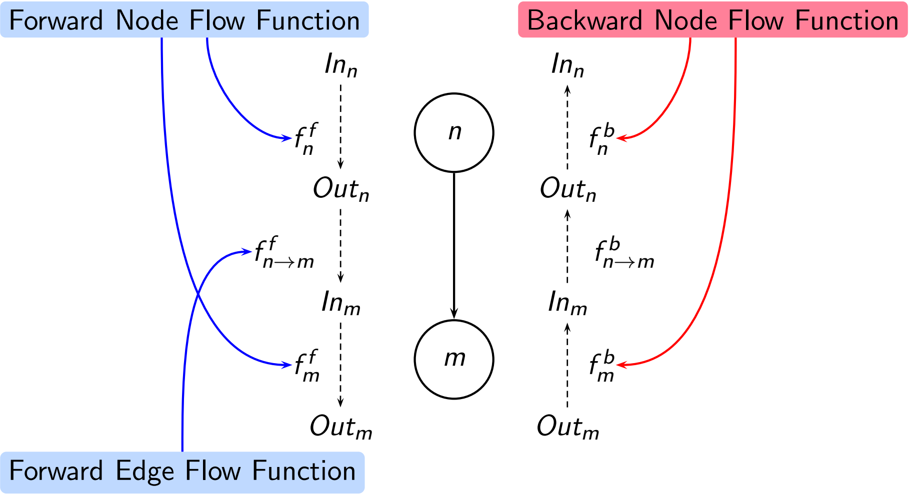
Edge and Node Flow Functions



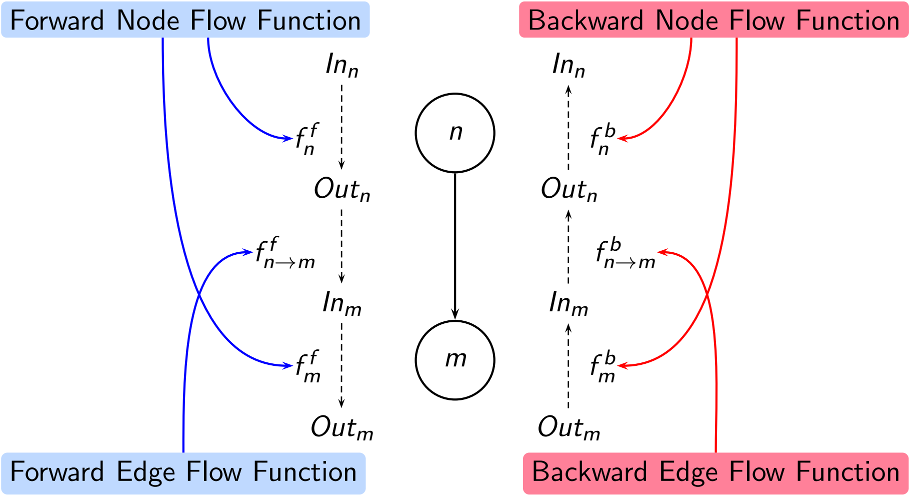
Edge and Node Flow Functions



Edge and Node Flow Functions



Edge and Node Flow Functions



General Data Flow Equations

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

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

- Edge flow functions are typically identity

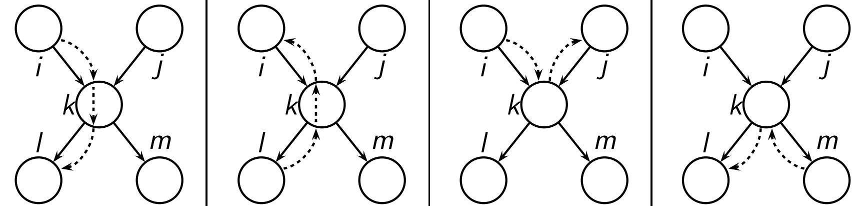
$$\forall x \in L,\ f(x) = x$$

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

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

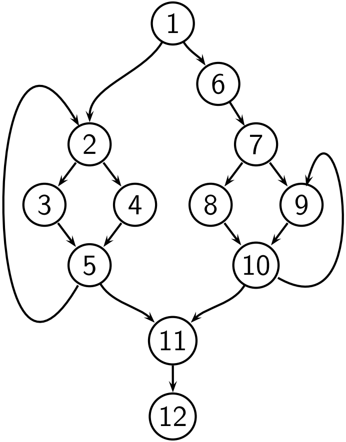
Modelling Information Flows Using Edge and Node Flow Functions

Forward	Backward	Bidirectional	Bidirectional



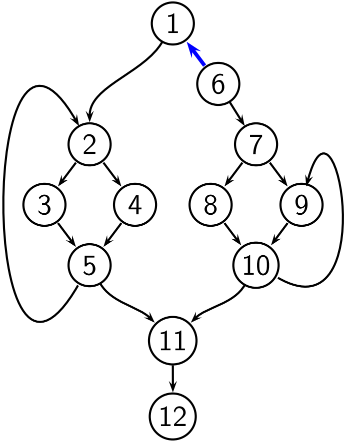
Forward	Backward	Bidirectional	Bidirectional
$f^f_{k\to l} \circ f^f_k \circ f^f_{i\to k}$	$f^b_{i\to k} \circ f^b_k \circ f^b_{k\to l}$	$f^f_{i\to k} \circ f^b_{k\to j}$	$f^f_{k\to l} \circ f^b_{k\to m}$

Information Flow Paths in PRE



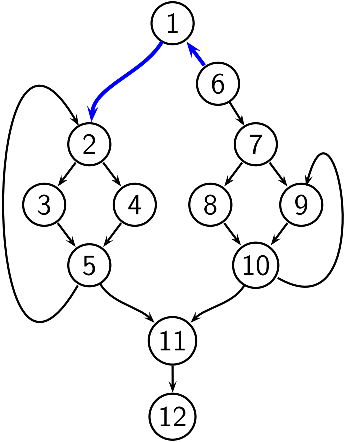
- Information could flow along arbitrary paths

Information Flow Paths in PRE



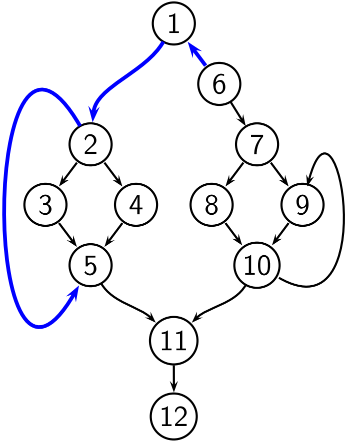
- Information could flow along arbitrary paths

Information Flow Paths in PRE



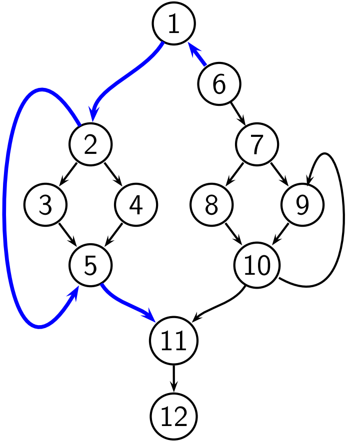
- Information could flow along arbitrary paths

Information Flow Paths in PRE



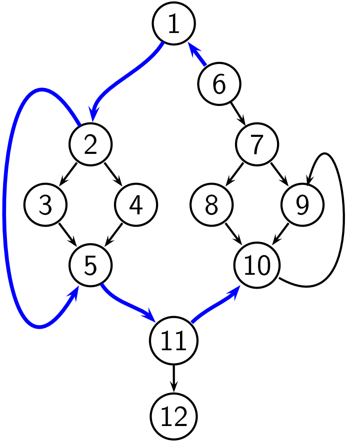
- Information could flow along arbitrary paths

Information Flow Paths in PRE



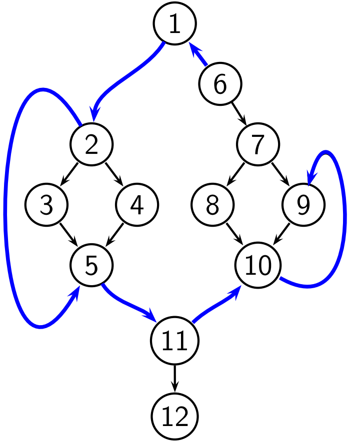
- Information could flow along arbitrary paths

Information Flow Paths in PRE



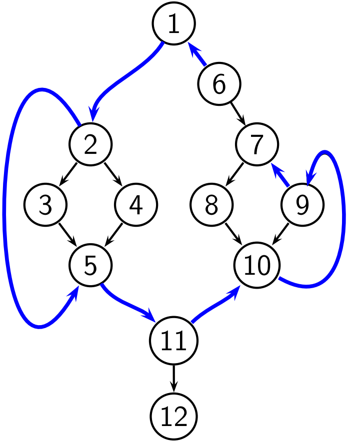
- Information could flow along arbitrary paths

Information Flow Paths in PRE



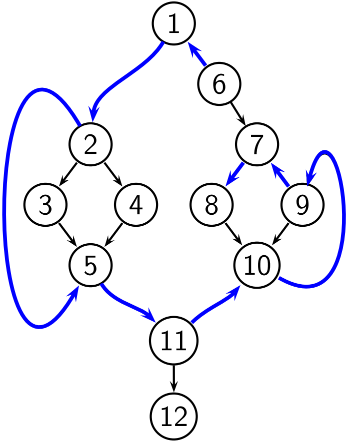
- Information could flow along arbitrary paths

Information Flow Paths in PRE



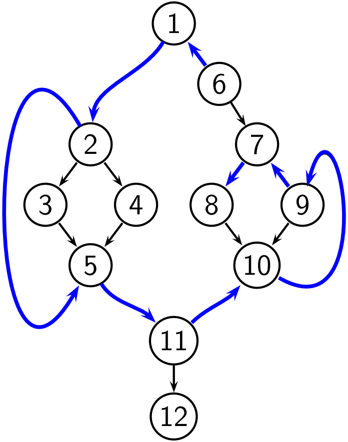
- Information could flow along arbitrary paths

Information Flow Paths in PRE



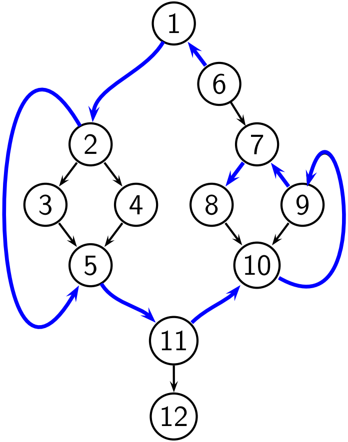
- Information could flow along arbitrary paths
- Theoretically predicted number : 144

Information Flow Paths in PRE



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

Information Flow Paths in PRE



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

Lacuna with PRE Complexity

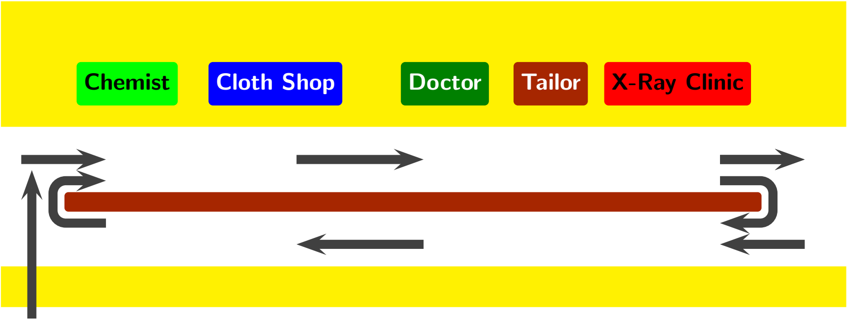
- Lacuna with PRE : Complexity $O(n^2)$ traversals.

  Practical graphs may have upto 50 nodes.

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

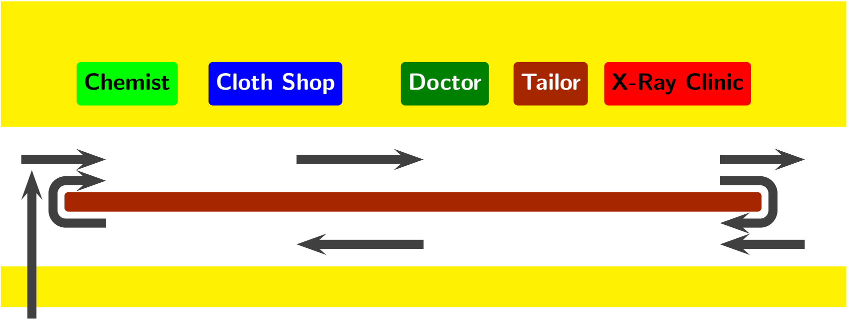
- No explanation for about 14 years despite dozens of efforts.
- Not much experimentation with performing advanced optimizations involving bidirectional dependency.

Complexity of Round Robin Iterative Method



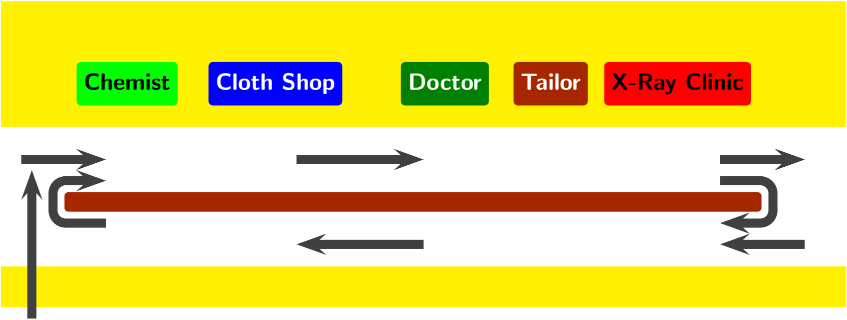
- Buy OTC (Over-The-Counter) medicine. No U-Turn 1 Trip

Complexity of Round Robin Iterative Method



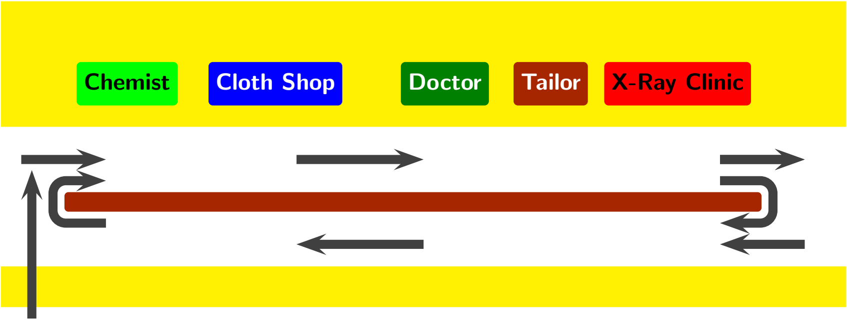
- Buy OTC (Over-The-Counter) medicine. No U-Turn 1 Trip
- Buy cloth. Give it to the tailor for stitching. No U-Turn 1 Trip

Complexity of Round Robin Iterative Method



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

Complexity of Round Robin Iterative Method



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

Information Flow Paths and Width of a Graph

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

Information Flow Paths and Width of a Graph

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

Information Flow Paths and Width of a Graph

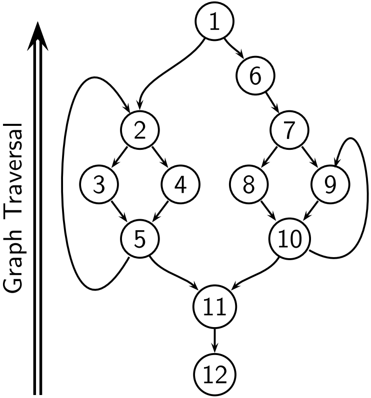
- A traversal $u \rightarrow v$ in an ifp is
  - *Compatible* if u is visited *before* v in the chosen graph traversal
  - *Incompatible* if u is visited *after* v in the chosen graph traversal
- Every incompatible edge traversal requires one additional iteration
- Width of a program flow graph with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

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

Information Flow Paths and Width of a Graph

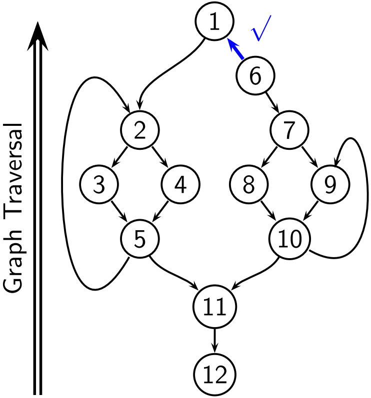
- A traversal $u \rightarrow v$ in an ifp is
  - *Compatible* if u is visited *before* v in the chosen graph traversal
  - *Incompatible* if u is visited *after* v in the chosen graph traversal
- Every incompatible edge traversal requires one additional iteration
- Width of a program flow graph with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



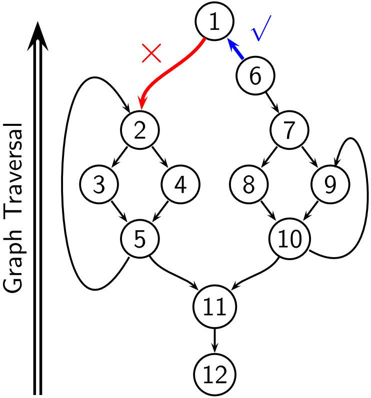
- Every "incompatible" edge traversal
  ⇒ **One additional graph traversal**

Complexity of Bidirectional Bit Vector Frameworks



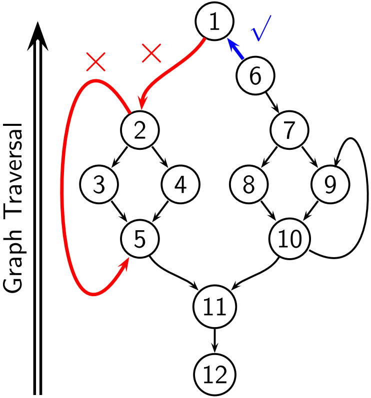
- Every "incompatible" edge traversal $\Rightarrow$ **One additional graph traversal**
- Max. Incompatible edge traversals $=$ *Width* of the graph $=$ **0?**
- Maximum number of traversals $=$ $1 +$ Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



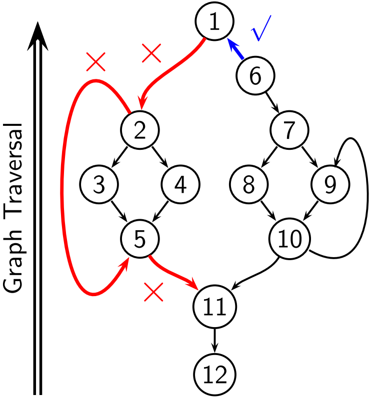
- **Every "incompatible" edge traversal $\Rightarrow$ One additional graph traversal**
- Max. Incompatible edge traversals $=$ *Width* of the graph $=$ **1?**
- Maximum number of traversals $=$ 1 $+$ Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



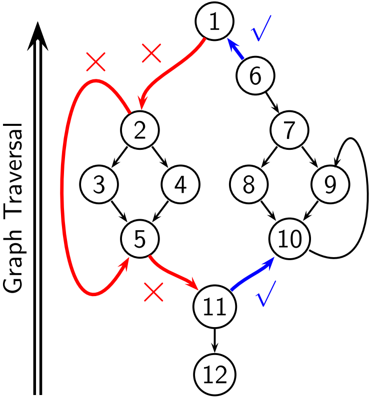
- Every "incompatible" edge traversal $\Rightarrow$ **One additional graph traversal**
- Max. Incompatible edge traversals $=$ *Width* of the graph $=$ **2?**
- Maximum number of traversals $=$ $1 +$ Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



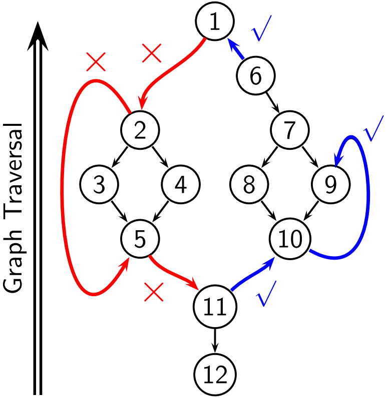
- Every "incompatible" edge traversal $\Rightarrow$ **One additional graph traversal**
- Max. Incompatible edge traversals $=$ *Width* of the graph $=$ **3?**
- Maximum number of traversals $=$ 1 $+$ Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



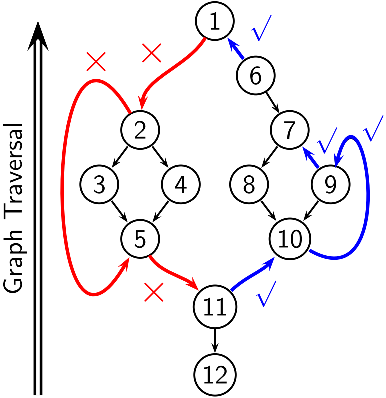
- Every "incompatible" edge traversal
  $\Rightarrow$ **One additional graph traversal**
- Max. Incompatible edge traversals
  = *Width* of the graph = **3?**
- Maximum number of traversals =
  1 + Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



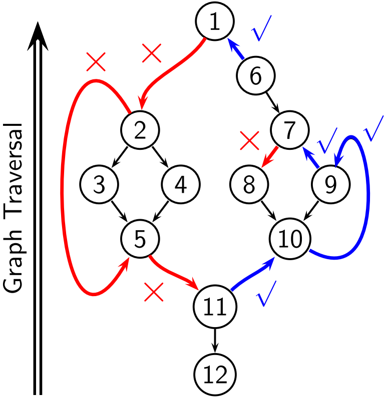
- **Every "incompatible" edge traversal**
  ⇒ One additional graph traversal
- Max. Incompatible edge traversals
  = *Width* of the graph = **3?**
- Maximum number of traversals =
  1 + Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



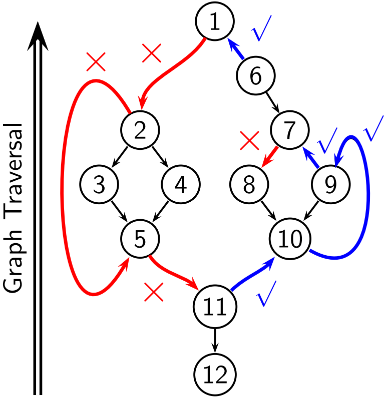
- Every "incompatible" edge traversal
  ⇒ **One additional graph traversal**
- Max. Incompatible edge traversals
  = *Width* of the graph = **3?**
- Maximum number of traversals =
  1 + Max. incompatible edge traversals

Complexity of Bidirectional Bit Vector Frameworks



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

Complexity of Bidirectional Bit Vector Frameworks



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

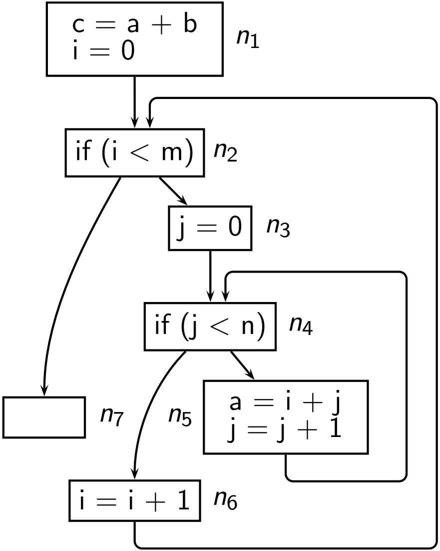
Width Subsumes Depth

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

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

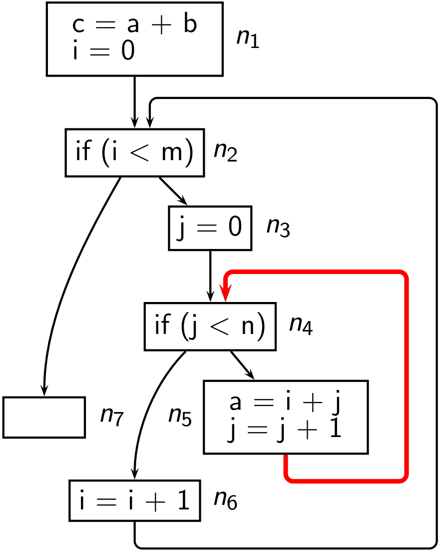
Width and Depth



Assuming reverse postorder traversal for available expressions analysis

- Depth = 2

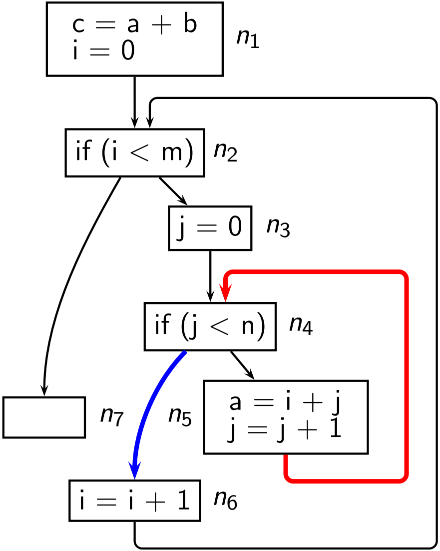
Width and Depth



Assuming reverse postorder traversal for available expressions analysis

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

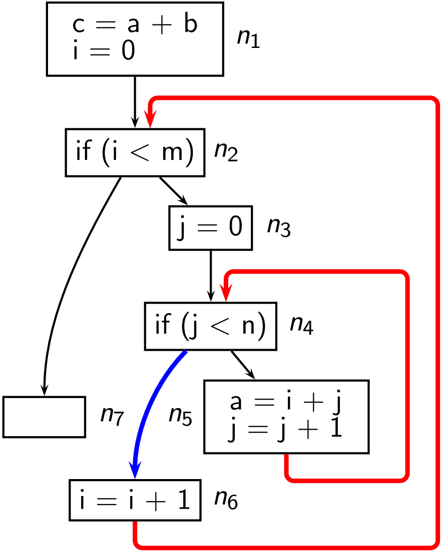
Width and Depth



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

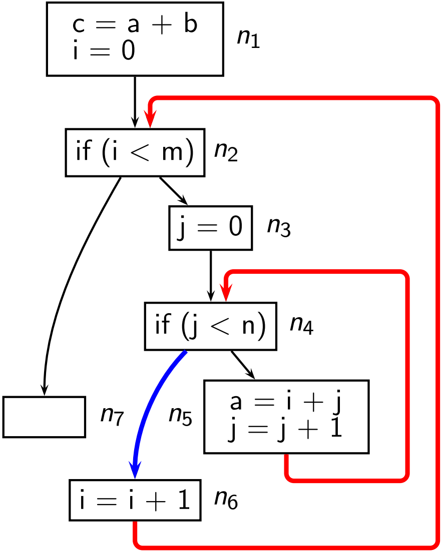
Width and Depth



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

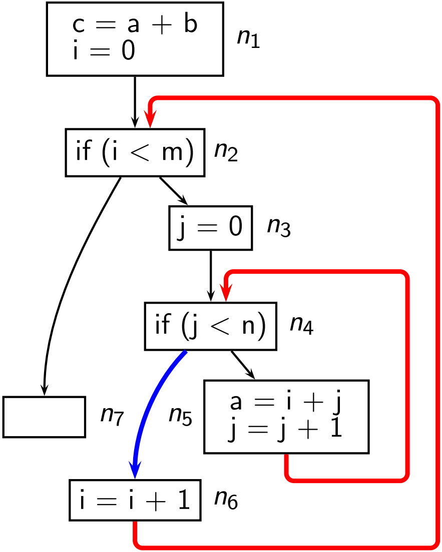
Width and Depth



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"?

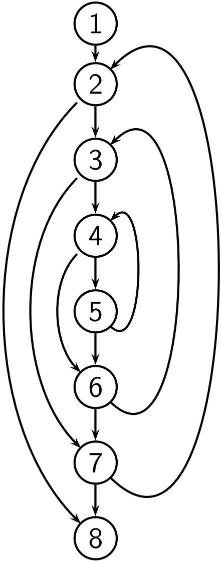
Width and Depth



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

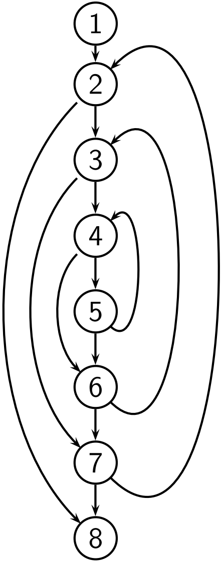
Width and Depth



Structures resulting from repeat-until loops with premature exits

- Depth = 3

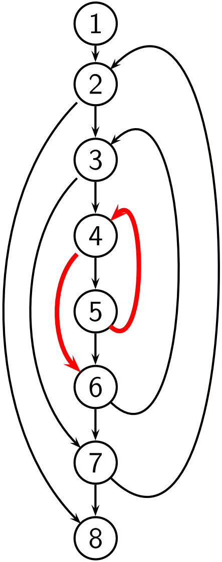
Width and Depth



Structures resulting from repeat-until loops with premature exits

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

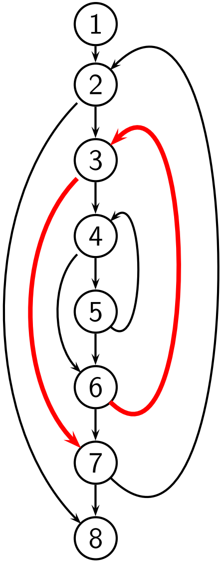
Width and Depth



Structures resulting from repeat-until loops with premature exits

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

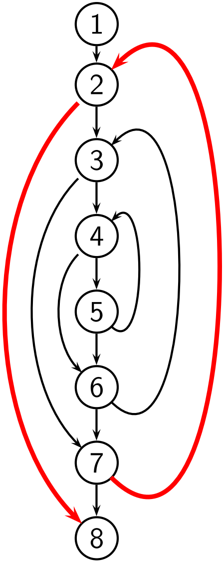
Width and Depth



Structures resulting from repeat-until loops with premature exits

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

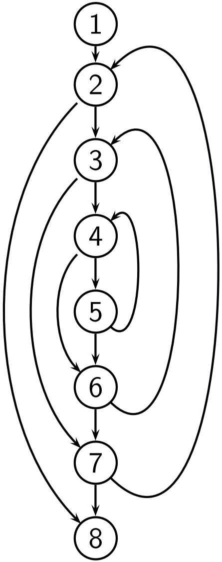
Width and Depth



Structures resulting from repeat-until loops with premature exits

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

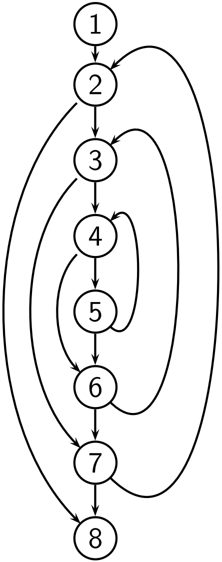
Width and Depth



Structures resulting from repeat-until loops with premature exits

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

Width and Depth



Structures resulting from repeat-until loops with premature exits

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

Work List Based Iterative Algorithm

Directly traverses information flow paths

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

Tutorial Problem

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

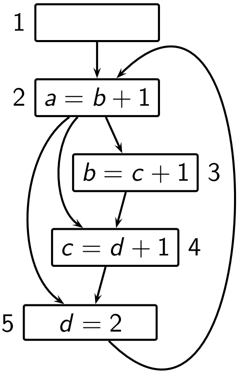
Show the trace of the analysis in the folloing format:

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

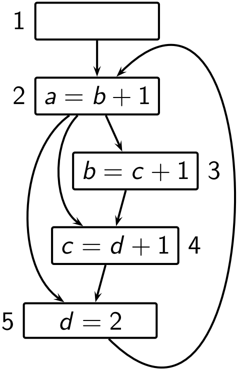
Part 5

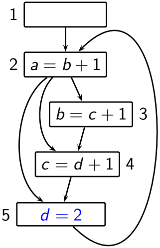
Precise Modelling of General Flows

Complexity of Constant Propagation?



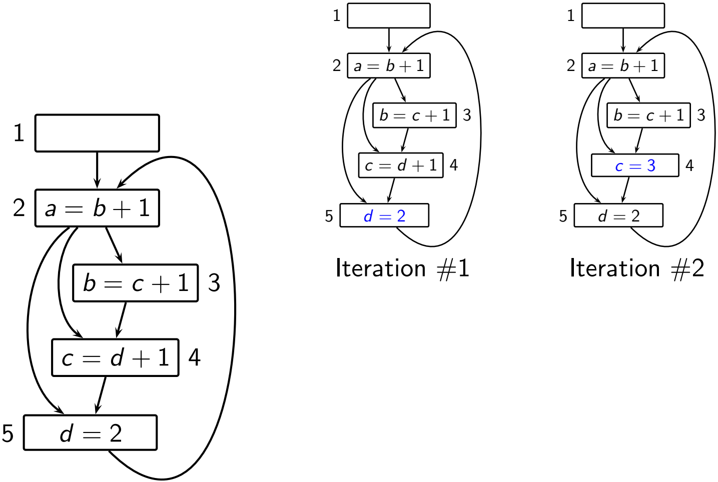
Complexity of Constant Propagation?



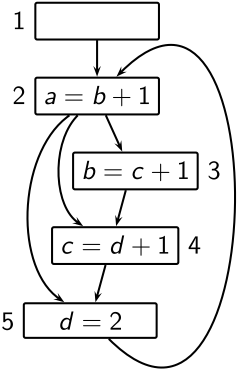


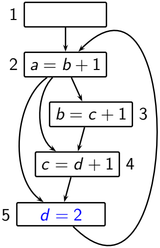
Iteration #1

Complexity of Constant Propagation?

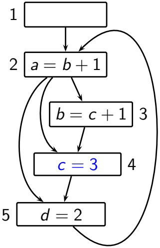


Complexity of Constant Propagation?

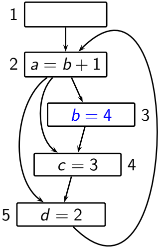




Iteration #1

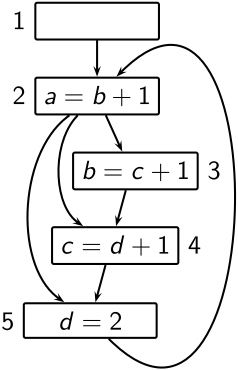


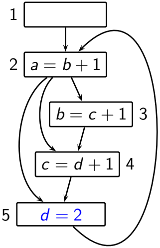
Iteration #2



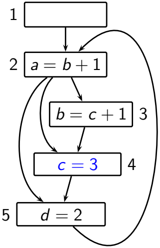
Iteration #3

Complexity of Constant Propagation?

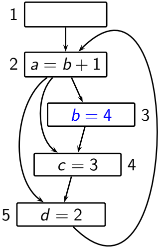




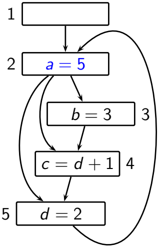
Iteration #1



Iteration #2

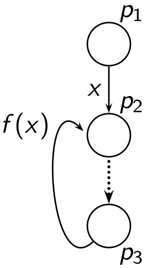


Iteration #3



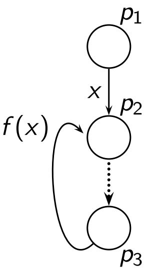
Iteration #4

Loop Closures of Flow Functions



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

Loop Closures of Flow Functions

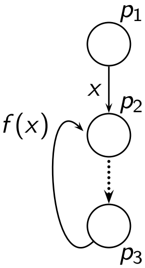


Paths Terminating at p_2	Data Flow Value
p_1, p_2	x
p_1, p_2, p_3, p_2	$f(x)$
$p_1, p_2, p_3, p_2, p_3, p_2$	$f(f(x)) = f^2(x)$
$p_1, p_2, p_3, p_2, p_3, p_2, p_3, p_2$	$f(f(f(x))) = f^3(x)$
...	...

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

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

Loop Closures of Flow Functions



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

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

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

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

Loop Closures in Bit Vector Frameworks

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

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

Loop Closures in Bit Vector Frameworks

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

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

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

Loop Closures in Bit Vector Frameworks

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

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

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

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

Larger Values of Loop Closure Bounds

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

Separability

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

Separability

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

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

Separability

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

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

Separability

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

Separable

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

$\widehat{h}_2$

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

Example: All bit vector frameworks

Non-Separable

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

$\Downarrow$

f

$\Downarrow$

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

Example: Constant Propagation

Separability

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

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

Example: All bit vector frameworks

Example: Constant Propagation

Separability

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

Separable

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

$\widehat{h}_2$

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

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

Example: All bit vector frameworks

Non-Separable

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

$\widehat{h}_2$

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

Example: Constant Propagation

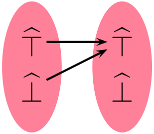
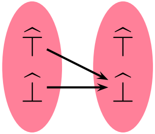
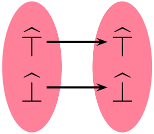
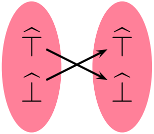
Separability

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

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

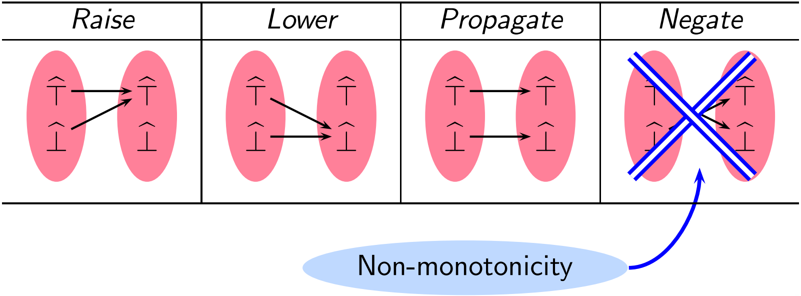
Separability of Bit Vector Frameworks

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

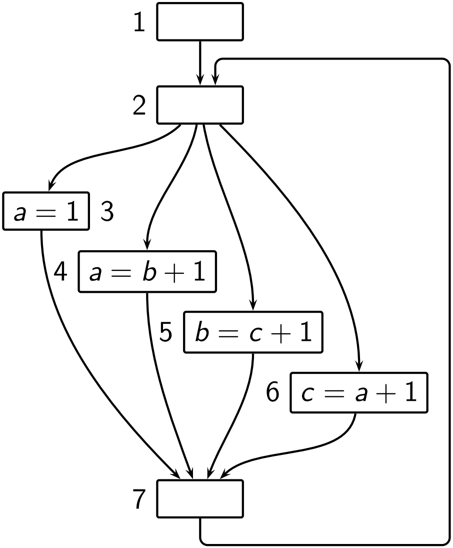
Raise	*Lower*	*Propagate*	*Negate*
			

Separability of Bit Vector Frameworks

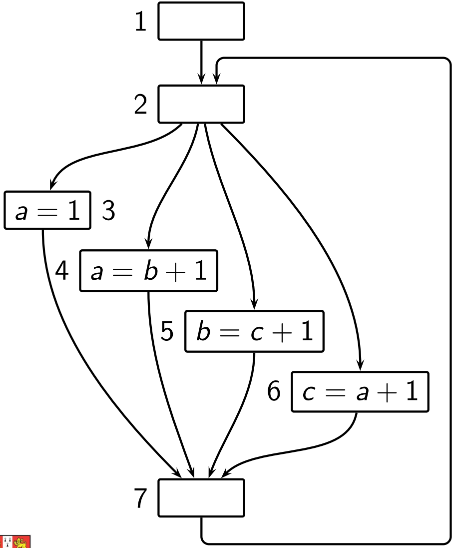
- $\widehat{L}$ is $\{0,1\}$, L is $\{0,1\}^m$
- $\widehat{\sqcap}$ is either boolean AND or boolean OR
- $\widehat{\top}$ and $\widehat{\bot}$ are 0 or 1 depending on $\widehat{\sqcap}$.
- $\widehat{h}$ is a *bit function* and could be one of the following:



Boundedness of Constant Propagation



Boundedness of Constant Propagation

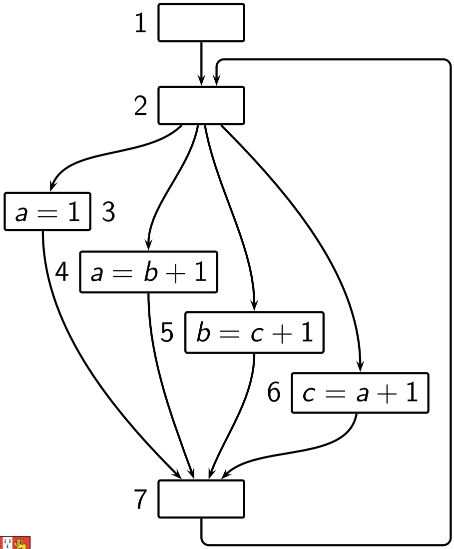


Summary flow function:

(data flow value at node 7)

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

Boundedness of Constant Propagation

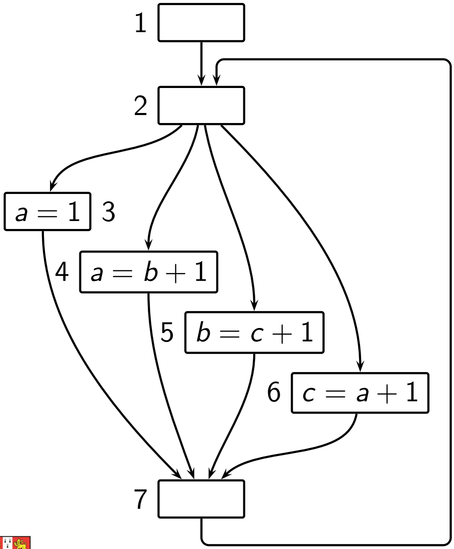


Summary flow function:
(data flow value at node 7)

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

$$f^0(\top) = \langle \widehat{\top}, \widehat{\top}, \widehat{\top}\rangle$$
$$f^1(\top) = \langle 1,\ \widehat{\top}, \widehat{\top}\rangle$$

Boundedness of Constant Propagation

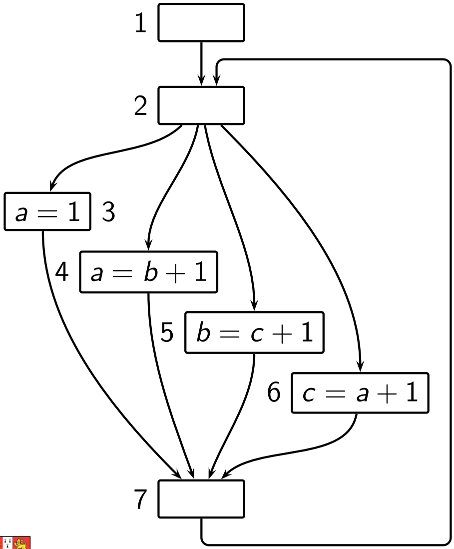


Summary flow function:
(data flow value at node 7)

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

$$f^0(\top) = \langle \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle$$
$$f^1(\top) = \langle 1, \widehat{\top}, \widehat{\top} \rangle$$
$$f^2(\top) = \langle 1, \widehat{\top}, 2 \rangle$$

Boundedness of Constant Propagation

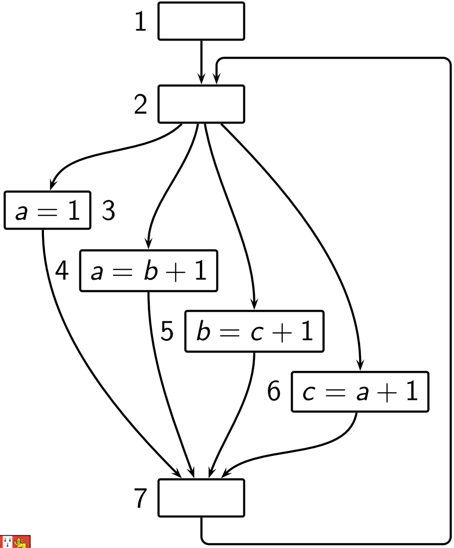


Summary flow function:
(data flow value at node 7)

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

$$\begin{aligned}
f^0(\top) &= \langle \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle \\
f^1(\top) &= \langle 1, \widehat{\top}, \widehat{\top} \rangle \\
f^2(\top) &= \langle 1, \widehat{\top}, 2 \rangle \\
f^3(\top) &= \langle 1, 3, 2 \rangle
\end{aligned}$$

Boundedness of Constant Propagation

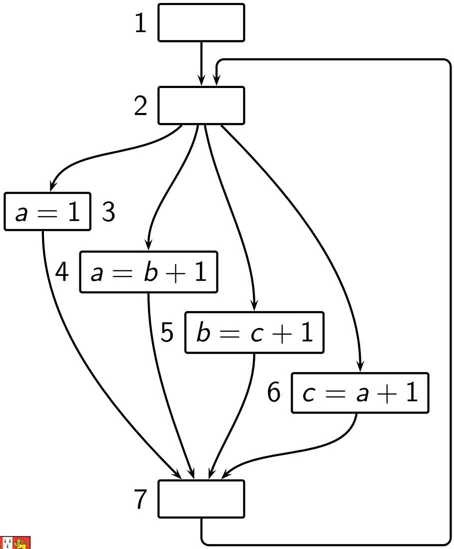


Summary flow function:
(data flow value at node 7)

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

$$\begin{aligned}
f^0(\top) &= \langle \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle \\
f^1(\top) &= \langle 1, \widehat{\top}, \widehat{\top} \rangle \\
f^2(\top) &= \langle 1, \widehat{\top}, 2 \rangle \\
f^3(\top) &= \langle 1, 3, 2 \rangle \\
f^4(\top) &= \langle \widehat{\perp}, 3, 2 \rangle
\end{aligned}$$

Boundedness of Constant Propagation

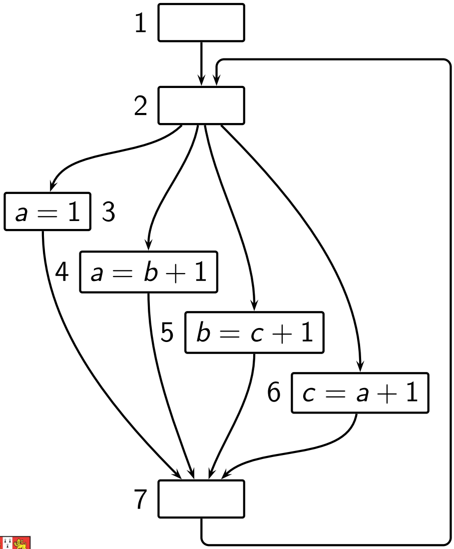


Summary flow function:
(data flow value at node 7)

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

$$\begin{aligned}
f^0(\top) &= \langle \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle \\
f^1(\top) &= \langle 1, \widehat{\top}, \widehat{\top} \rangle \\
f^2(\top) &= \langle 1, \widehat{\top}, 2 \rangle \\
f^3(\top) &= \langle 1, 3, 2 \rangle \\
f^4(\top) &= \langle \widehat{\bot}, 3, 2 \rangle \\
f^5(\top) &= \langle \widehat{\bot}, 3, \widehat{\bot} \rangle
\end{aligned}$$

Boundedness of Constant Propagation

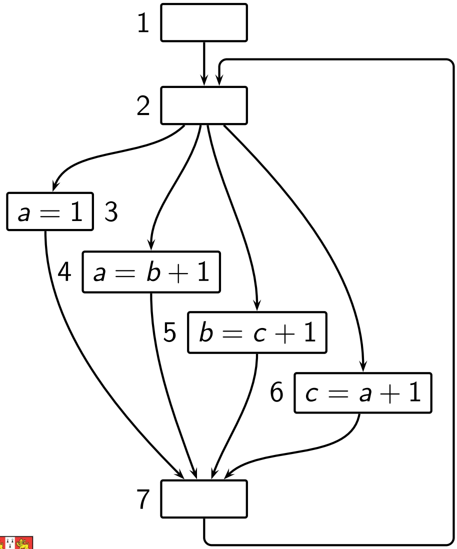


Summary flow function:
(data flow value at node 7)

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

$$\begin{aligned}
f^0(\top) &= \langle \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle \\
f^1(\top) &= \langle 1, \widehat{\top}, \widehat{\top} \rangle \\
f^2(\top) &= \langle 1, \widehat{\top}, 2 \rangle \\
f^3(\top) &= \langle 1, 3, 2 \rangle \\
f^4(\top) &= \langle \widehat{\bot}, 3, 2 \rangle \\
f^5(\top) &= \langle \widehat{\bot}, 3, \widehat{\bot} \rangle \\
f^6(\top) &= \langle \widehat{\bot}, \widehat{\bot}, \widehat{\bot} \rangle
\end{aligned}$$

Boundedness of Constant Propagation



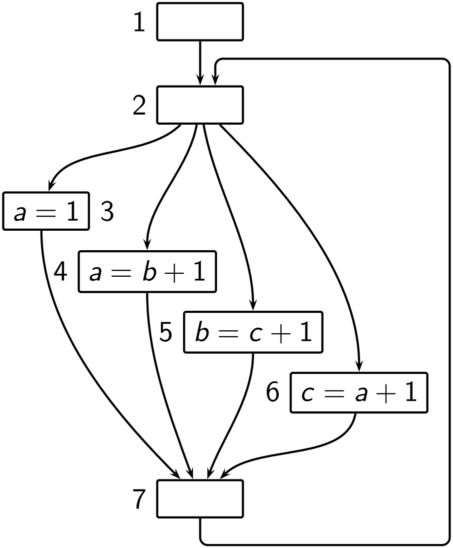
Summary flow function:

(data flow value at node 7)

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

$$\begin{aligned}
f^0(\top) &= \langle \widehat{\top}, \widehat{\top}, \widehat{\top} \rangle \\
f^1(\top) &= \langle 1, \widehat{\top}, \widehat{\top} \rangle \\
f^2(\top) &= \langle 1, \widehat{\top}, 2 \rangle \\
f^3(\top) &= \langle 1, 3, 2 \rangle \\
f^4(\top) &= \langle \widehat{\bot}, 3, 2 \rangle \\
f^5(\top) &= \langle \widehat{\bot}, 3, \widehat{\bot} \rangle \\
f^6(\top) &= \langle \widehat{\bot}, \widehat{\bot}, \widehat{\bot} \rangle \\
f^7(\top) &= \langle \widehat{\bot}, \widehat{\bot}, \widehat{\bot} \rangle
\end{aligned}$$

Boundedness of Constant Propagation



$$f^*(\top) = \prod_{i=0}^{6} f^i(\top)$$

Boundedness of Constant Propagation

The moral of the story:

- The data flow value of every variable could change twice

Boundedness of Constant Propagation

The moral of the story:

- The data flow value of every variable could change twice
- In the worst case, only one change may happen in every step of a function application

Boundedness of Constant Propagation

The moral of the story:

- The data flow value of every variable could change twice
- In the worst case, only one change may happen in every step of a function application
- Maximum number of steps: $2 \times |\mathbb{V}\text{ar}|$

Boundedness of Constant Propagation

The moral of the story:

- The data flow value of every variable could change twice
- In the worst case, only one change may happen in every step of a function application
- Maximum number of steps: $2 \times |\mathbb{V}\text{ar}|$
- Boundedness parameter k is $(2 \times |\mathbb{V}\text{ar}|) + 1$

Modelling Flow Functions for General Flows

- General flow functions can be written as

$$f_n(X) = (X - Kill_n(X)) \cup Gen_n(X)$$

where *Gen* and *Kill* have constant and dependent parts

$$\begin{aligned} Gen_n(X) &= ConstGen_n \cup DepGen_n(X) \\ Kill_n(X) &= ConstKill_n \cup DepKill_n(X) \end{aligned}$$

Modelling Flow Functions for General Flows

- General flow functions can be written as

$$f_n(X) \quad = \quad (X - Kill_n(X)) \cup Gen_n(X)$$

  where *Gen* and *Kill* have constant and dependent parts

$$\begin{aligned} Gen_n(X) &= ConstGen_n \cup DepGen_n(X) \\ Kill_n(X) &= ConstKill_n \cup DepKill_n(X) \end{aligned}$$

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

Modelling Flow Functions for General Flows

- General flow functions can be written as

$$f_n(X) \quad = \quad (X - Kill_n(X)) \cup Gen_n(X)$$

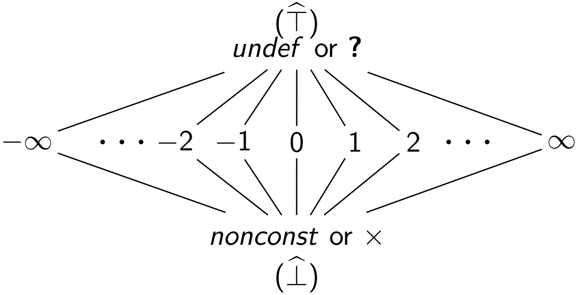
where *Gen* and *Kill* have constant and dependent parts

$$\begin{aligned} Gen_n(X) &= ConstGen_n \cup DepGen_n(X) \\ Kill_n(X) &= ConstKill_n \cup DepKill_n(X) \end{aligned}$$

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

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

Component Lattice for Integer Constant Propagation



- Overall lattice L is the product of $\widehat{L}$ for all variables.
- $\sqcap$ and $\widehat{\sqcap}$ get defined by $\sqsubseteq$ and $\widehat{\sqsubseteq}$.

$\widehat{\sqcap}$	$\langle v, \textbf{?} \rangle$	$\langle v, \times \rangle$	$\langle v, c_1 \rangle$
$\langle v, \textbf{?} \rangle$	$\langle v, \textbf{?} \rangle$	$\langle v, \times \rangle$	$\langle v, c_1 \rangle$
$\langle v, \times \rangle$	$\langle v, \times \rangle$	$\langle v, \times \rangle$	$\langle v, \times \rangle$
$\langle v, c_2 \rangle$	$\langle v, c_2 \rangle$	$\langle v, \times \rangle$	If $c_1 = c_2$ then $\langle v, c_1 \rangle$ else $\langle v, \times \rangle$

Flow Functions for Constant Propagation

- Flow function for $r = a_1 * a_2$

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

Defining Data Flow Equations for Constant Propagation

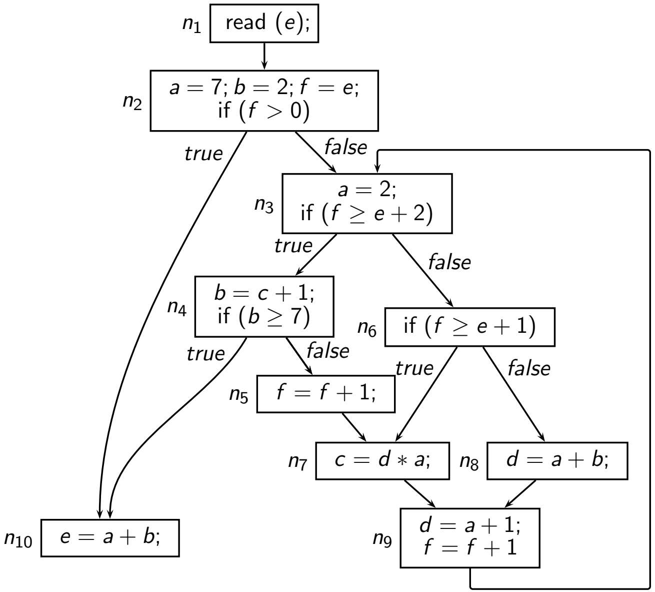
	$ConstGen_n$	$DepGen_n(X)$	$ConstKill_n$	$DepKill_n(X)$
$v = c$, $c \in \mathbb{C}\text{onst}$	$\{\langle v, c\rangle\}$	$\emptyset$	$\emptyset$	$\{\langle v, d\rangle \mid \langle v, d\rangle \in X\}$
$v = e$, $e \in \mathbb{E}\text{xpr}$	$\emptyset$	$\{\langle v, eval(e,X)\rangle\}$	$\emptyset$	$\{\langle v, d\rangle \mid \langle v, d\rangle \in X\}$
$read(v)$	$\{\langle v, \times\rangle\}$	$\emptyset$	$\emptyset$	$\{\langle v, d\rangle \mid \langle v, d\rangle \in X\}$
$other$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Defining Data Flow Equations for Constant Propagation

	$ConstGen_n$	$DepGen_n(X)$	$ConstKill_n$	$DepKill_n(X)$
$v = c$, $c \in \mathbb{C}\text{onst}$	$\{\langle v, c\rangle\}$	$\emptyset$	$\emptyset$	$\{\langle v, d\rangle \mid \langle v, d\rangle \in X\}$
$v = e$, $e \in \mathbb{E}\text{xpr}$	$\emptyset$	$\{\langle v, eval(e,X)\rangle\}$	$\emptyset$	$\{\langle v, d\rangle \mid \langle v, d\rangle \in X\}$
$read(v)$	$\{\langle v, \times\rangle\}$	$\emptyset$	$\emptyset$	$\{\langle v, d\rangle \mid \langle v, d\rangle \in X\}$
$other$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

$eval(a_1\ op\ a_2, X)$			
	$\langle a_1, \textbf{?}\rangle \in X$	$\langle a_1, \times\rangle \in X$	$\langle a_1, c_1\rangle \in X$
$\langle a_2, \textbf{?}\rangle \in X$	**?**	$\times$	**?**
$\langle a_2, \times\rangle \in X$	$\times$	$\times$	$\times$
$\langle a_2, c_2\rangle \in X$	**?**	$\times$	$c_1\ op\ c_2$

Example Program for Constant Propagation



Result of Constant Propagation

	Iteration #1	Changes in iteration #2	Changes in iteration #3	Changes in iteration #4
In_{n_1}	$\widehat{\top},\widehat{\top},\widehat{\top},\widehat{\top},\widehat{\top},\widehat{\top}$			
Out_{n_1}	$\widehat{\top},\widehat{\top},\widehat{\top},\widehat{\top},\widehat{\bot},\widehat{\top}$			
In_{n_2}	$\widehat{\top},\widehat{\top},\widehat{\top},\widehat{\top},\widehat{\bot},\widehat{\top}$			
Out_{n_2}	$7,2,\widehat{\top},\widehat{\top},\widehat{\bot},\widehat{\bot}$			
In_{n_3}	$7,2,\widehat{\top},\widehat{\top},\widehat{\bot},\widehat{\bot}$	$\widehat{\bot},2,\widehat{\top},3,\widehat{\bot},\widehat{\bot}$	$\widehat{\bot},2,6,3,\widehat{\bot},\widehat{\bot}$	$\widehat{\bot},\widehat{\bot},6,3,\widehat{\bot},\widehat{\bot}$
Out_{n_3}	$2,2,\widehat{\top},\widehat{\top},\widehat{\bot},\widehat{\bot}$	$2,2,\widehat{\top},3,\widehat{\bot},\widehat{\bot}$	$2,2,6,3,\widehat{\bot},\widehat{\bot}$	$2,\widehat{\bot},6,3,\widehat{\bot},\widehat{\bot}$
In_{n_4}	$2,2,\widehat{\top},\widehat{\top},\widehat{\bot},\widehat{\bot}$	$2,2,\widehat{\top},3,\widehat{\bot},\widehat{\bot}$	$2,2,6,3,\widehat{\bot},\widehat{\bot}$	$2,\widehat{\bot},6,3,\widehat{\bot},\widehat{\bot}$
Out_{n_4}	$2,\widehat{\top},\widehat{\top},\widehat{\top},\widehat{\bot},\widehat{\bot}$	$2,\widehat{\top},\widehat{\top},3,\widehat{\bot},\widehat{\bot}$	$2,7,6,3,\widehat{\bot},\widehat{\bot}$	
In_{n_5}	$2,\widehat{\top},\widehat{\top},\widehat{\top},\widehat{\bot},\widehat{\bot}$	$2,\widehat{\top},\widehat{\top},3,\widehat{\bot},\widehat{\bot}$	$2,7,6,3,\widehat{\bot},\widehat{\bot}$	
Out_{n_5}	$2,\widehat{\top},\widehat{\top},\widehat{\top},\widehat{\bot},\widehat{\bot}$	$2,\widehat{\top},\widehat{\top},3,\widehat{\bot},\widehat{\bot}$	$2,7,6,3,\widehat{\bot},\widehat{\bot}$	
In_{n_6}	$2,2,\widehat{\top},\widehat{\top},\widehat{\bot},\widehat{\bot}$	$2,2,\widehat{\top},3,\widehat{\bot},\widehat{\bot}$	$2,2,6,3,\widehat{\bot},\widehat{\bot}$	$2,\widehat{\bot},6,3,\widehat{\bot},\widehat{\bot}$
Out_{n_6}	$2,2,\widehat{\top},\widehat{\top},\widehat{\bot},\widehat{\bot}$	$2,2,\widehat{\top},3,\widehat{\bot},\widehat{\bot}$	$2,2,6,3,\widehat{\bot},\widehat{\bot}$	$2,\widehat{\bot},6,3,\widehat{\bot},\widehat{\bot}$
In_{n_7}	$2,2,\widehat{\top},\widehat{\top},\widehat{\bot},\widehat{\bot}$	$2,2,\widehat{\top},3,\widehat{\bot},\widehat{\bot}$	$2,\widehat{\bot},6,3,\widehat{\bot},\widehat{\bot}$	
Out_{n_7}	$2,2,\widehat{\top},\widehat{\top},\widehat{\bot},\widehat{\bot}$	$2,2,6,3,\widehat{\bot},\widehat{\bot}$	$2,\widehat{\bot},6,3,\widehat{\bot},\widehat{\bot}$	
In_{n_8}	$2,2,\widehat{\top},\widehat{\top},\widehat{\bot},\widehat{\bot}$	$2,2,\widehat{\top},3,\widehat{\bot},\widehat{\bot}$	$2,2,6,3,\widehat{\bot},\widehat{\bot}$	$2,\widehat{\bot},6,3,\widehat{\bot},\widehat{\bot}$
Out_{n_8}	$2,2,\widehat{\top},4,\widehat{\bot},\widehat{\bot}$	$2,2,\widehat{\top},4,\widehat{\bot},\widehat{\bot}$	$2,2,6,4,\widehat{\bot},\widehat{\bot}$	$2,\widehat{\bot},6,\widehat{\bot},\widehat{\bot},\widehat{\bot}$
In_{n_9}	$2,2,\widehat{\top},4,\widehat{\bot},\widehat{\bot}$	$2,2,6,\widehat{\bot},\widehat{\bot},\widehat{\bot}$	$2,\widehat{\bot},6,\widehat{\bot},\widehat{\bot},\widehat{\bot}$	
Out_{n_9}	$2,2,\widehat{\top},3,\widehat{\bot},\widehat{\bot}$	$2,2,6,3,\widehat{\bot},\widehat{\bot}$	$2,\widehat{\bot},6,3,\widehat{\bot},\widehat{\bot}$	
$In_{n_{10}}$	$\widehat{\bot},2,\widehat{\top},\widehat{\top},\widehat{\bot},\widehat{\bot}$	$\widehat{\bot},2,\widehat{\top},3,\widehat{\bot},\widehat{\bot}$	$\widehat{\bot},\widehat{\bot},6,3,\widehat{\bot},\widehat{\bot}$	
$Out_{n_{10}}$	$\widehat{\bot},2,\widehat{\top},\widehat{\top},\widehat{\bot},\widehat{\bot}$	$\widehat{\bot},2,\widehat{\top},3,\widehat{\bot},\widehat{\bot}$	$\widehat{\bot},\widehat{\bot},6,3,\widehat{\bot},\widehat{\bot}$	

Monotonicity of Constant Propagation

- Flow function $f_n(X) = (X - Kill_n(X)) \cup Gen_n(X)$ where

$$\begin{aligned} Gen_n(X) &= ConstGen_n \cup DepGen_n(X) \\ Kill_n(X) &= ConstKill_n \cup DepKill_n(X) \end{aligned}$$

- $ConstGen_n$ and $ConstKill_n$ are trivially monotonic
- To show $X_1 \sqsubseteq X_2 \Rightarrow DepGen_n(X_1) \sqsubseteq DepGen_n(X_2)$
  we need to show that $X_1 \sqsubseteq X_2 \Rightarrow eval(e, X_1) \sqsubseteq eval(e, X_2)$.
  This follows from definition of $eval(e, X)$.
- To show $X_1 \sqsubseteq X_2 \Rightarrow (X_1 - DepKill_n(X_1)) \sqsubseteq (X_2 - DepKill_n(X_2))$
  observe that $DepKill_n$ removes the pair corresponding to the variable modified in statement n. Data flow values of other variables remain unaffected.