Some 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.

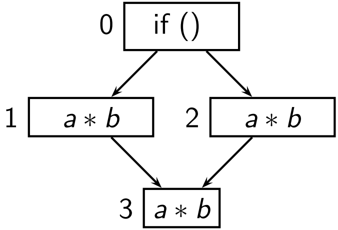
Outline

- Partial Redundancy Elimination
- Introduction to Constant Propagation
- Theoretical Abstractions in Data Flow Analysis
  - The world of data flow values
  - The world of functions and operations that compute data values (Not today)
  - Results of data flow analysis (Not today)
  - Algorithms for performing data flow analysis (Not today)

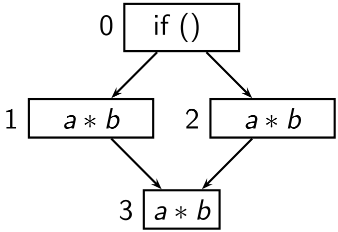
Part 2

Partial Redundancy Elimination

Precursor: Common Subexpression Elimination

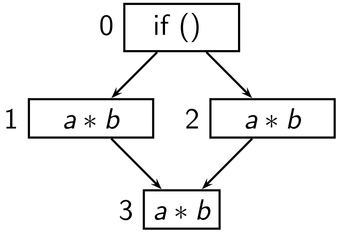


Precursor: Common Subexpression Elimination



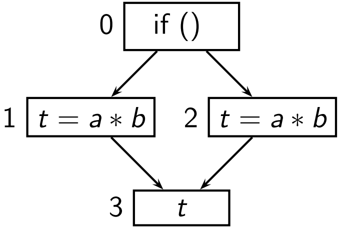
- a and b are not modified along paths $1 \rightarrow 3$ and $2 \rightarrow 3$

Precursor: Common Subexpression Elimination



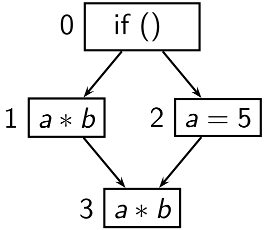
- a and b are not modified along paths $1 \rightarrow 3$ and $2 \rightarrow 3$
- Computation of $a * b$ in 3 is redundant

Precursor: Common Subexpression Elimination

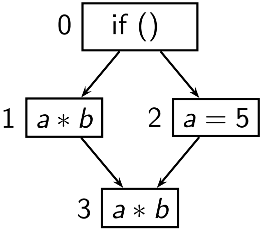


- a and b are not modified along paths $1 \rightarrow 3$ and $2 \rightarrow 3$
- Computation of $a * b$ in 3 is redundant
- Previous value can be used

Partial Redundancy Elimination

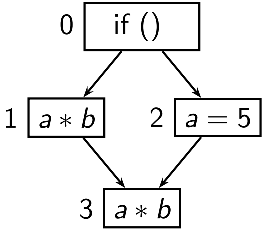


Partial Redundancy Elimination



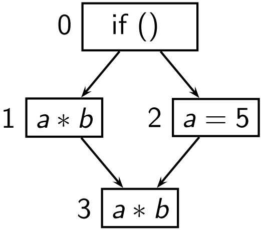
- Computation of $a * b$ in 3 is

Partial Redundancy Elimination



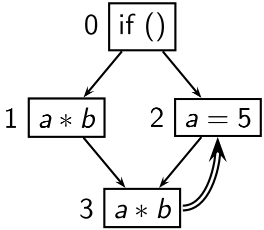
- Computation of $a * b$ in 3 is
  - redundant along path $1 \rightarrow 3$, but ...

Partial Redundancy Elimination

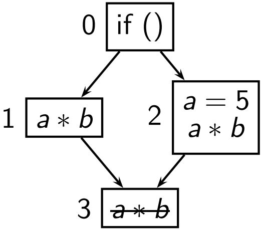


- Computation of $a * b$ in 3 is
  - redundant along path $1 \rightarrow 3$, but ...
  - not redundant along path $2 \rightarrow 3$

Code Hoisting for Partial Redundancy Elimination

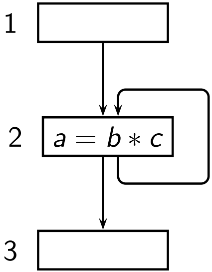


Code Hoisting for Partial Redundancy Elimination

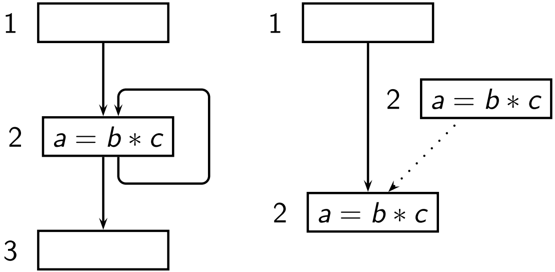


- Computation of $a * b$ in 3 becomes totally redundant
- Can be deleted

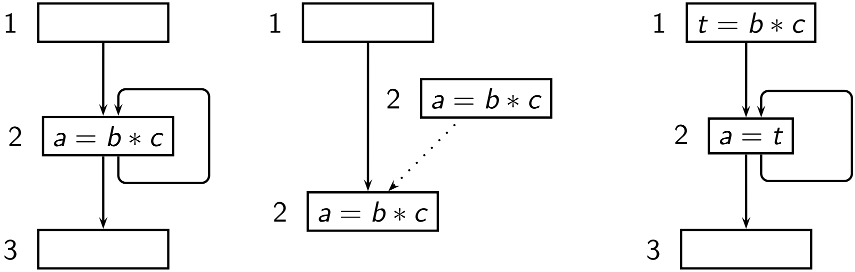
PRE Subsumes Loop Invariant Movement



PRE Subsumes Loop Invariant Movement



PRE Subsumes Loop Invariant Movement

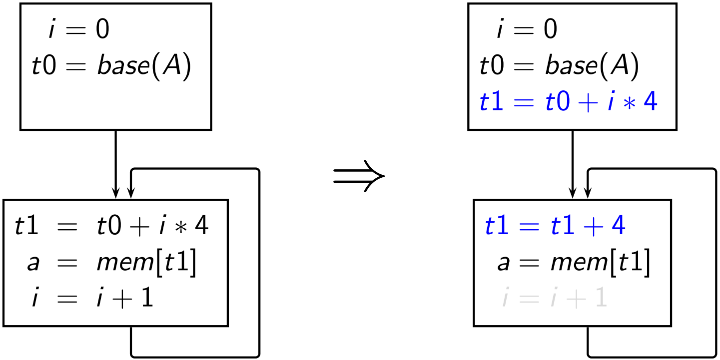


PRE Can be Used for Strength Reduction

$$
\begin{aligned}
i &= 0 \\
t0 &= base(A)
\end{aligned}
$$

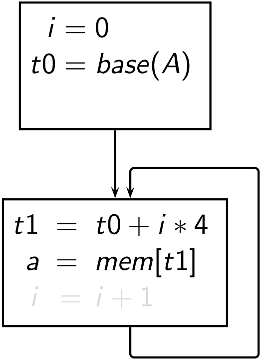
$$
\begin{aligned}
t1 &= t0 + i * 4 \\
a &= mem[t1] \\
i &= i + 1
\end{aligned}
$$

PRE Can be Used for Strength Reduction



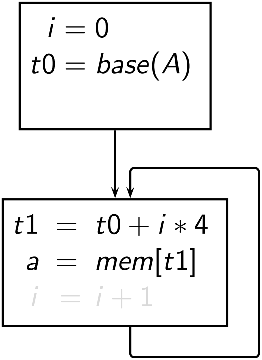
- $*$ and $+$ in the loop have been replaced by $+$
- $i = i + 1$ in the loop has been eliminated

PRE Can be Used for Strength Reduction



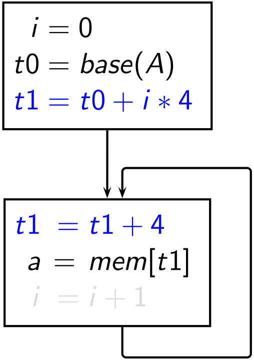
- Delete $i = i + 1$

PRE Can be Used for Strength Reduction



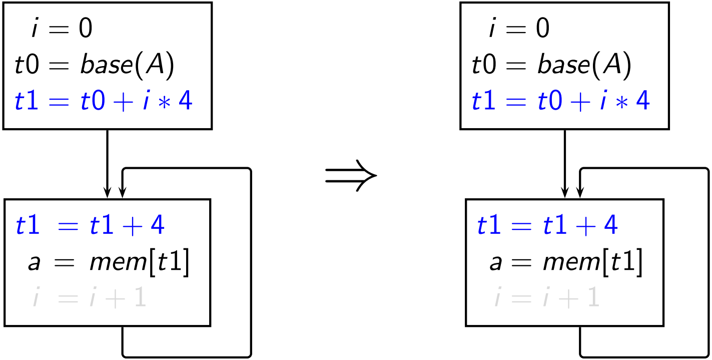
- Delete $i = i + 1$
- Expression $t0 + i * 4$ becomes loop invariant

PRE Can be Used for Strength Reduction



- Delete $i = i + 1$
- Expression $t0 + i * 4$ becomes loop invariant
- Hoist it and increment $t1$ in the loop

PRE Can be Used for Strength Reduction



- $*$ and $+$ in the loop have been replaced by $+$
- $i = i + 1$ in the loop has been eliminated

Performing Partial Redundancy Elimination

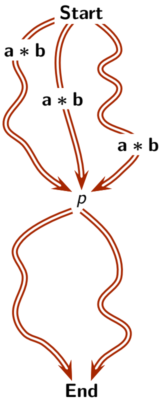
1. Identify partial redundancies
2. Identify program points where computations can be inserted
3. Insert expressions
4. Partial redundancies become total redundancies
   $\Longrightarrow$ Delete them.

Morel-Renvoise Algorithm (*CACM*, 1979.)

Defining Hoisting Criteria

- An expression can be safely inserted at a program point p if it is

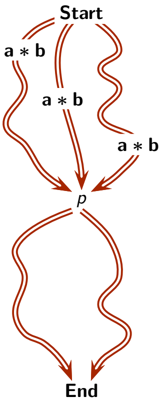
Available at p



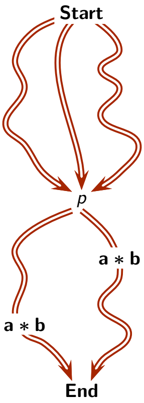
Defining Hoisting Criteria

- An expression can be safely inserted at a program point p if it is

Available at p

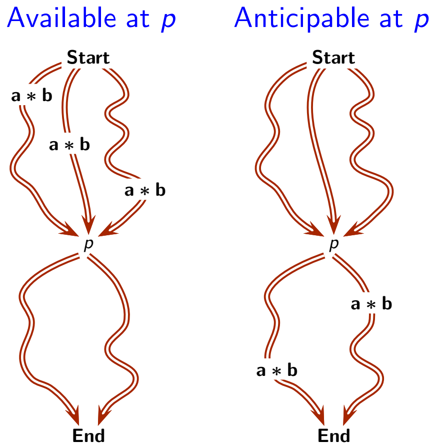


Anticipable at p



Defining Hoisting Criteria

- An expression can be safely inserted at a program point p if it is

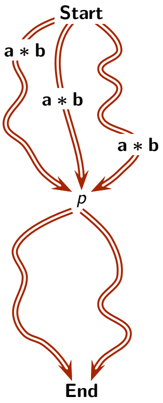


- If it is available at p, then there is no need to insert it at p.

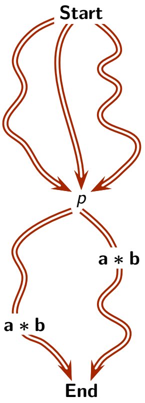
Defining Hoisting Criteria

- An expression can be safely inserted at a program point p if it is

Available at p



Anticipable at p

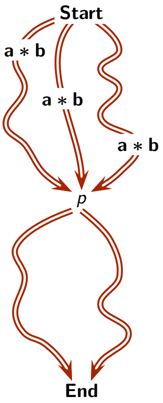


- If it is available at p, then there is no need to insert it at p.
- If it is anticipable at p then all such occurrence should be hoisted to p.

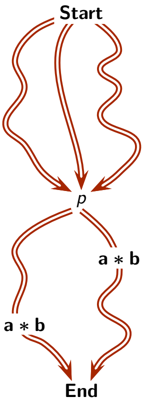
Defining Hoisting Criteria

- An expression can be safely inserted at a program point p if it is

Available at p



Anticipable at p



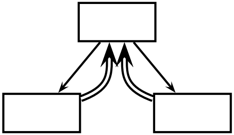
- If it is available at p, then there is no need to insert it at p.
- If it is anticipable at p then all such occurrence should be hoisted to p.
- *An expression should be hoisted to p provided it can be hoisted to p along all paths from p to exit.*

Hoisting Criteria

- *Safety of hoisting to the exit of a block*.
  - S.1 Should be hoisted only if it can be hoisted to the entry of all succesors

Hoisting Criteria

- *Safety of hoisting to the exit of a block*.
  - S.1 Should be hoisted only if it can be hoisted to the entry of all succesors



Hoisting Criteria

- *Safety of hoisting to the exit of a block*.
  - S.1 Should be hoisted only if it can be hoisted to the entry of all succesors
- *Safety of hoisting to the entry of a block*.
  Should be hoisted only if
  - S.2 it is upwards exposed, or

Hoisting Criteria

- *Safety of hoisting to the exit of a block*.
  - S.1 Should be hoisted only if it can be hoisted to the entry of all succesors

- *Safety of hoisting to the entry of a block*.
  Should be hoisted only if
  - S.2 it is upwards exposed, or

$\boxed{a * c}$

$\boxed{\begin{array}{c} a * c \\ a = \end{array}}$

Hoisting Criteria

- *Safety of hoisting to the exit of a block*.

  S.1 Should be hoisted only if it can be hoisted to the entry of all succesors

- *Safety of hoisting to the entry of a block*.
  Should be hoisted only if

  S.2 it is upwards exposed, or

  S.3 it can be hoisted to its exit and is transparent in the block

$a * c$

$a * c$
$a =$

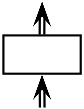
Hoisting Criteria

- *Safety of hoisting to the exit of a block*.

  S.1 Should be hoisted only if it can be hoisted to the entry of all succesors

- *Safety of hoisting to the entry of a block*.
  Should be hoisted only if

  S.2 it is upwards exposed, or
  S.3 it can be hoisted to its exit and is transparent in the block



Hoisting Criteria

- *Safety of hoisting to the exit of a block*.
  - S.1 Should be hoisted only if it can be hoisted to the entry of all succesors
- *Safety of hoisting to the entry of a block*.
  Should be hoisted only if
  - S.2 it is upwards exposed, or
  - S.3 it can be hoisted to its exit and is transparent in the block
- *Desirability of hoisting to the entry of a block*.
  Should be hoisted only if
  - D.1 it is partially available, and
  - D.2 For each predecessor

Hoisting Criteria

- *Safety of hoisting to the exit of a block*.
  - S.1 Should be hoisted only if it can be hoisted to the entry of all succesors
- *Safety of hoisting to the entry of a block*.
  Should be hoisted only if
  - S.2 it is upwards exposed, or
  - S.3 it can be hoisted to its exit and is transparent in the block
- *Desirability of hoisting to the entry of a block*.
  Should be hoisted only if
  - D.1 it is partially available, and
  - D.2 For each predecessor
    - D.2.a it is hoisted to its exit, or

Hoisting Criteria

- *Safety of hoisting to the exit of a block*.

  S.1 Should be hoisted only if it can be hoisted to the entry of all succesors

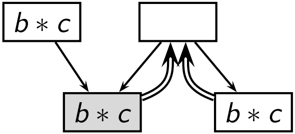
- *Safety of hoisting to the entry of a block*.
  Should be hoisted only if

  S.2 it is upwards exposed, or
  S.3 it can be hoisted to its exit and is transparent in the block

- *Desirability of hoisting to the entry of a block*.
  Should be hoisted only if

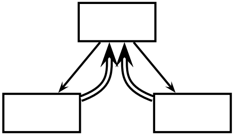
  D.1 it is partially available, and
  D.2 For each predecessor

  D.2.a it is hoisted to its exit, or
  D.2.b is available at its exit.

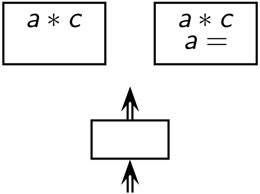


Hoisting Criteria

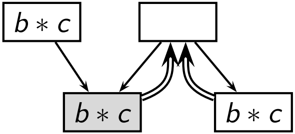
- *Safety of hoisting to the exit of a block*.
  - S.1 Should be hoisted only if it can be hoisted to the entry of all succesors



- *Safety of hoisting to the entry of a block*.
  Should be hoisted only if
  - S.2 it is upwards exposed, or
  - S.3 it can be hoisted to its exit and is transparent in the block



- *Desirability of hoisting to the entry of a block*.
  Should be hoisted only if
  - D.1 it is partially available, and
  - D.2 For each predecessor
    - D.2.a it is hoisted to its exit, or
    - D.2.b is available at its exit.



Applying the Hoisting Criteria

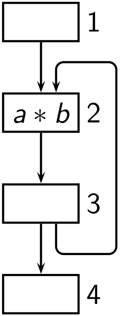
- *Safety of hoisting to the exit of a block.*
  - S.1 Should be hoisted only if it can be hoisted to the entry of all succesors
- *Safety of hoisting to the entry of a block.*
  Should be hoisted only if
  - S.2 it is upwards exposed, or
  - S.3 it can be hoisted to its exit an
    in the block
- *Desirability of hoisting to the en*
  Should be hoisted only if
  - D.1 it is partially available, and
  - D.2 For each predecessor
    - D.2.a it is hoisted to its exit, or
    - D.2.b is available at its exit.

What does this slide show?

- Four examples
- For each example
  - statements in blue enable hoisting
  - statements in red prohibit hoisting

Applying the Hoisting Criteria

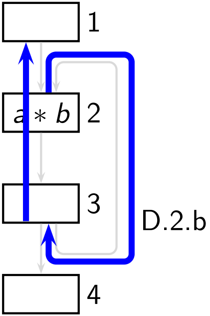
- *Safety of hoisting to the exit of a block.*
  - S.1 Should be hoisted only if it can be hoisted to the entry of all succesors
- *Safety of hoisting to the entry of a block.* Should be hoisted only if
  - S.2 it is upwards exposed, or
  - S.3 it can be hoisted to its exit and is transparent in the block
- *Desirability of hoisting to the entry of a block.* Should be hoisted only if
  - D.1 it is partially available, and
  - D.2 For each predecessor
    - D.2.a it is hoisted to its exit, or
    - D.2.b is available at its exit.



(Example 1)

Applying the Hoisting Criteria

- *Safety of hoisting to the exit of a block.*
  - S.1 Should be hoisted only if it can be hoisted to the entry of all succesors
- *Safety of hoisting to the entry of a block.*
  Should be hoisted only if
  - S.2 it is upwards exposed, or
  - S.3 it can be hoisted to its exit and is transparent in the block
- *Desirability of hoisting to the entry of a block.*
  Should be hoisted only if
  - D.1 it is partially available, and
  - D.2 For each predecessor
    - D.2.a it is hoisted to its exit, or
    - D.2.b is available at its exit.



(Example 1)

Applying the Hoisting Criteria

- *Safety of hoisting to the exit of a block.*

  S.1 Should be hoisted only if it can be hoisted to the entry of all succesors

- *Safety of hoisting to the entry of a block.*
  Should be hoisted only if

  S.2 it is upwards exposed, or
  S.3 it can be hoisted to its exit and is transparent in the block

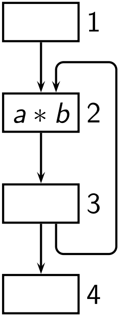
- *Desirability of hoisting to the entry of a block.*
  Should be hoiste

  D.1 it is partially
  D.2 For each pred

    D.2.a it is hoisted to its exit, or
    D.2.b is available at its exit.

What if we insert

- $a = 2$ in 3?
- $a = 2$ in 3 and $a * b$ in 4?



(Example 1)

Applying the Hoisting Criteria

- *Safety of hoisting to the exit of a block.*

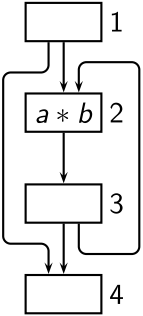
  S.1 Should be hoisted only if it can be hoisted to the entry of all succesors

- *Safety of hoisting to the entry of a block.*
  Should be hoisted only if

  S.2 it is upwards exposed, or
  S.3 it can be hoisted to its exit and is transparent in the block

- *Desirability of hoisting to the entry of a block.*
  Should be hoisted only if

  D.1 it is partially available, and
  D.2 For each predecessor
    D.2.a it is hoisted to its exit, or
    D.2.b is available at its exit.



(Example 2)

Applying the Hoisting Criteria

- *Safety of hoisting to the exit of a block.*

  S.1 Should be hoisted only if it can be hoisted to the entry of all succesors

- *Safety of hoisting to the entry of a block.*
  Should be hoisted only if

  S.2 it is upwards exposed, or

  S.3 it can be hoisted to its exit and is transparent in the block

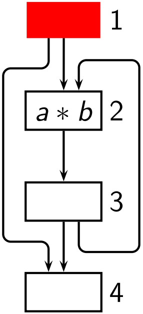
- *Desirability of hoisting to the entry of a block.*
  Should be hoisted only if

  D.1 it is partially available, and

  D.2 For each predecessor

  D.2.a it is hoisted to its exit, or

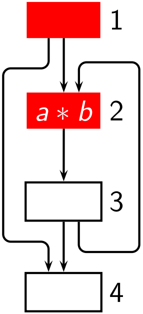
  D.2.b is available at its exit.



(Example 2)

Applying the Hoisting Criteria

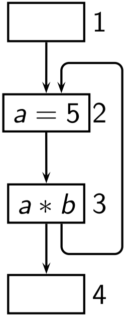
- *Safety of hoisting to the exit of a block.*
  - S.1 Should be hoisted only if it can be hoisted to the entry of all succesors
- *Safety of hoisting to the entry of a block.* Should be hoisted only if
  - S.2 it is upwards exposed, or
  - S.3 it can be hoisted to its exit and is transparent in the block
- *Desirability of hoisting to the entry of a block.* Should be hoisted only if
  - D.1 it is partially available, and
  - D.2 For each predecessor
    - D.2.a it is hoisted to its exit, or
    - D.2.b is available at its exit.



(Example 2)

Applying the Hoisting Criteria

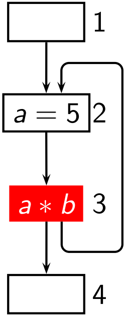
- *Safety of hoisting to the exit of a block.*
  - S.1 Should be hoisted only if it can be hoisted to the entry of all succesors
- *Safety of hoisting to the entry of a block.* Should be hoisted only if
  - S.2 it is upwards exposed, or
  - S.3 it can be hoisted to its exit and is transparent in the block
- *Desirability of hoisting to the entry of a block.* Should be hoisted only if
  - D.1 it is partially available, and
  - D.2 For each predecessor
    - D.2.a it is hoisted to its exit, or
    - D.2.b is available at its exit.



(Example 3)

Applying the Hoisting Criteria

- *Safety of hoisting to the exit of a block.*
  - S.1 Should be hoisted only if it can be hoisted to the entry of all succesors
- *Safety of hoisting to the entry of a block.* Should be hoisted only if
  - S.2 it is upwards exposed, or
  - S.3 it can be hoisted to its exit and is transparent in the block
- *Desirability of hoisting to the entry of a block.* Should be hoisted only if
  - D.1 it is partially available, and
  - D.2 For each predecessor
    - D.2.a it is hoisted to its exit, or
    - D.2.b is available at its exit.



(Example 3)

Applying the Hoisting Criteria

- *Safety of hoisting to the exit of a block.*

  S.1 Should be hoisted only if it can be hoisted to the entry of all succesors

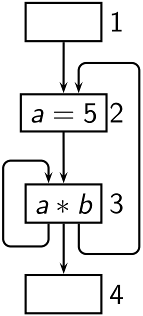
- *Safety of hoisting to the entry of a block.*
  Should be hoisted only if

  S.2 it is upwards exposed, or
  S.3 it can be hoisted to its exit and is transparent in the block

- *Desirability of hoisting to the entry of a block.*
  Should be hoisted only if

  D.1 it is partially available, and
  D.2 For each predecessor

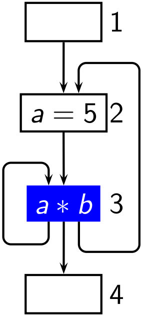
    D.2.a it is hoisted to its exit, or
    D.2.b is available at its exit.



(Example 4)

Applying the Hoisting Criteria

- *Safety of hoisting to the exit of a block.*
  - S.1 Should be hoisted only if it can be hoisted to the entry of all succesors
- *Safety of hoisting to the entry of a block.* Should be hoisted only if
  - S.2 it is upwards exposed, or
  - S.3 it can be hoisted to its exit and is transparent in the block
- *Desirability of hoisting to the entry of a block.* Should be hoisted only if
  - D.1 it is partially available, and
  - D.2 For each predecessor
    - D.2.a it is hoisted to its exit, or
    - D.2.b is available at its exit.



(Example 4)

Applying the Hoisting Criteria

- *Safety of hoisting to the exit of a block.*

  S.1 Should be hoisted only if it can be hoisted to the entry of all succesors

- *Safety of hoisting to the entry of a block.*
  Should be hoisted only if

  S.2 it is upwards exposed, or

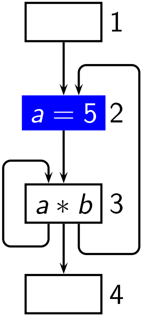
  S.3 it can be hoisted to its exit and is transparent in the block

- *Desirability of hoisting to the entry of a block.*
  Should be hoisted only if

  D.1 it is partially available, and

  D.2 For each predecessor

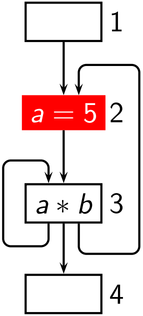
  - D.2.a it is hoisted to its exit, or
  - D.2.b is available at its exit.



(Example 4)

Applying the Hoisting Criteria

- S.1 Should be hoisted only if it can be hoisted to the entry of all succesors
- Should be hoisted only if
  - S.2 it is upwards exposed, or
  - S.3 it can be hoisted to its exit and is transparent in the block
- Should be hoisted only if
  - D.1 it is partially available, and
  - D.2 For each predecessor
    - D.2.a it is hoisted to its exit, or
    - D.2.b is available at its exit.



(Example 4)

First Level Global Data Flow Properties in PRE

- Partial Availability.

$$PavIn_n = \begin{cases} BI & n \text{ is } Start \text{ block} \\ \bigcup_{p \in pred(n)} PavOut_p & \text{otherwise} \end{cases}$$

$$PavOut_n = Gen_n \cup (PavIn_n - Kill_n)$$

- Total Availability.

$$AvIn_n = \begin{cases} BI & n \text{ is } Start \text{ block} \\ \bigcap_{p \in pred(n)} AvOut_p & \text{otherwise} \end{cases}$$

$$AvOut_n = Gen_n \cup (AvIn_n - Kill_n)$$

PRE Data Flow Equations

Desirability: D.1

$$In_n = \boxed{PavIn_n} \cap \left(AntGen_n \cup (Out_n - Kill_n) \right)$$
$$\bigcap_{p \in pred(n)} \left(Out_p \cup AvOut_p \right)$$

$$Out_n = \begin{cases} BI & n \text{ is } End \text{ block} \\ \bigcap_{s \in succ(n)} In_s & \text{otherwise} \end{cases}$$

Expressions should be partially available, and

PRE Data Flow Equations

Safety: S.2

$$In_n = PavIn_n \cap \Big(AntGen_n \cup (Out_n - Kill_n) \Big) \bigcap_{p \in pred(n)} \Big(Out_p \cup AvOut_p \Big)$$

$$Out_n = \begin{cases} BI & n \text{ is } End \text{ block} \\ \bigcap_{s \in succ(n)} In_s & \text{otherwise} \end{cases}$$

Expressions should be upwards exposed, or

PRE Data Flow Equations

Safety: S.3

$$In_n = PavIn_n \cap \Big(AntGen_n \cup \big(Out_n - Kill_n\big) \Big) \bigcap_{p \in pred(n)} \Big(Out_p \cup AvOut_p \Big)$$

$$Out_n = \begin{cases} BI & n \text{ is } End \text{ block} \\ \bigcap_{s \in succ(n)} In_s & \text{otherwise} \end{cases}$$

Expressions can be hoisted to the exit and are transparent in the block

PRE Data Flow Equations

Desirability: D.2.a

$$In_n = PavIn_n \cap \left(AntGen_n \cup (Out_n - Kill_n) \right) \bigcap_{p \in pred(n)} \left(Out_p \cup AvOut_p \right)$$

$$Out_n = \begin{cases} BI & n \text{ is } End \text{ block} \\ \bigcap_{s \in succ(n)} In_s & \text{otherwise} \end{cases}$$

For every predecessor, expressions can be hoisted to its exit, or

PRE Data Flow Equations

Desirability: D.2.b

$$In_n = PavIn_n \cap \left(AntGen_n \cup (Out_n - Kill_n) \right) \bigcap_{p \in pred(n)} \left(Out_p \cup AvOut_p \right)$$

$$Out_n = \begin{cases} BI & n \text{ is } End \text{ block} \\ \bigcap_{s \in succ(n)} In_s & \text{otherwise} \end{cases}$$

...expressions are available at the exit of the same predecessor

PRE Data Flow Equations

Safety: S.1

$$In_n = PavIn_n \cap \left(AntGen_n \cup (Out_n - Kill_n) \right) \bigcap_{p \in pred(n)} \left(Out_p \cup AvOut_p \right)$$

$$Out_n = \begin{cases} BI & n \text{ is } End \text{ block} \\ \bigcap_{s \in succ(n)} In_s & \text{otherwise} \end{cases}$$

Expressions should be hoisted to the exit of a block if they can be hoisted to the entry of all succesors

PRE Data Flow Equations

$$
\begin{aligned}
In_n &= PavIn_n \cap \Big(AntGen_n \cup (Out_n - Kill_n) \Big) \\
& \qquad \bigcap_{p \in pred(n)} \Big(Out_p \cup AvOut_p \Big) \\
Out_n &= \begin{cases} BI & n \text{ is } End \text{ block} \\ \displaystyle\bigcap_{s \in succ(n)} In_s & \text{otherwise} \end{cases}
\end{aligned}
$$

Deletion Criteria in PRE

- An expression is redundant in node n if
  - it can be placed at the entry (i.e. can be "hoisted" out) of n, AND
  - it is upwards exposed in node n.

$$Redundant_n = In_n \cap AntGen_n$$

- A hoisting path for an expression e begins at n if $e \in Redundant_n$
- This hoisting path extends against the control flow.

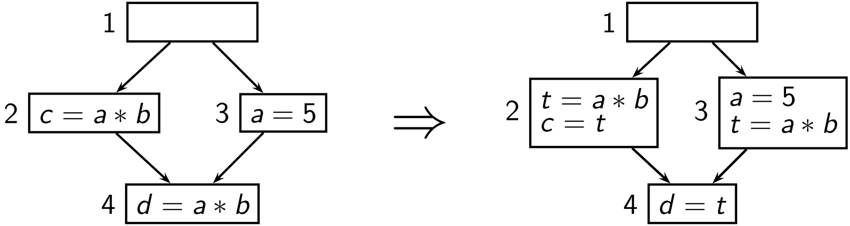
Insertion Criteria in PRE

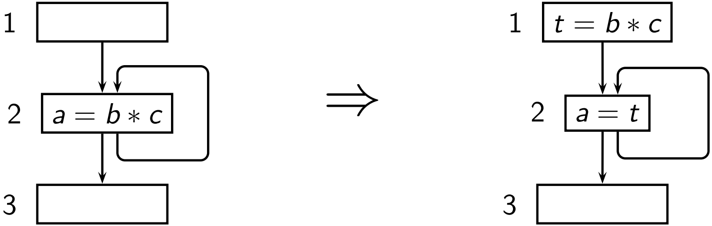
- An expression is inserted at the exit of node n is
  - it can be placed at the exit of n, AND
  - it is not available at the exit of n, AND
  - it cannot be hoisted out of n, OR it is modified in n.

$$Insert_n = Out_n \cap (\neg AvOut_n) \cap (\neg In_n \cup Kill_n)$$

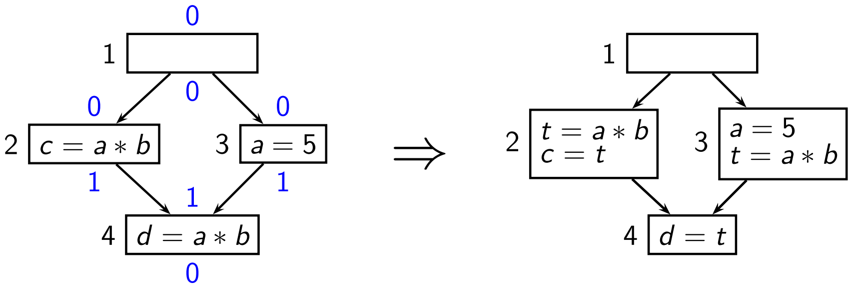
- A hoisting path for an expression e ends at n if $e \in Insert_n$

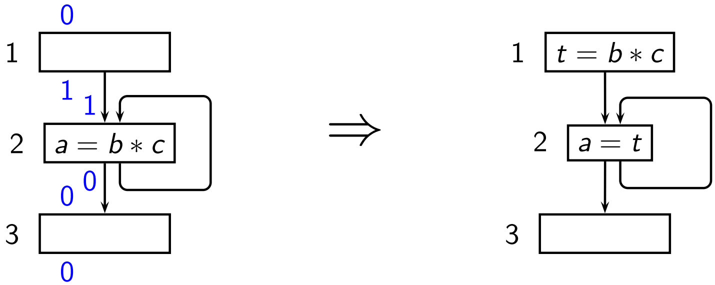
Performing PRE by Computing *In*/*Out*: Simple Cases



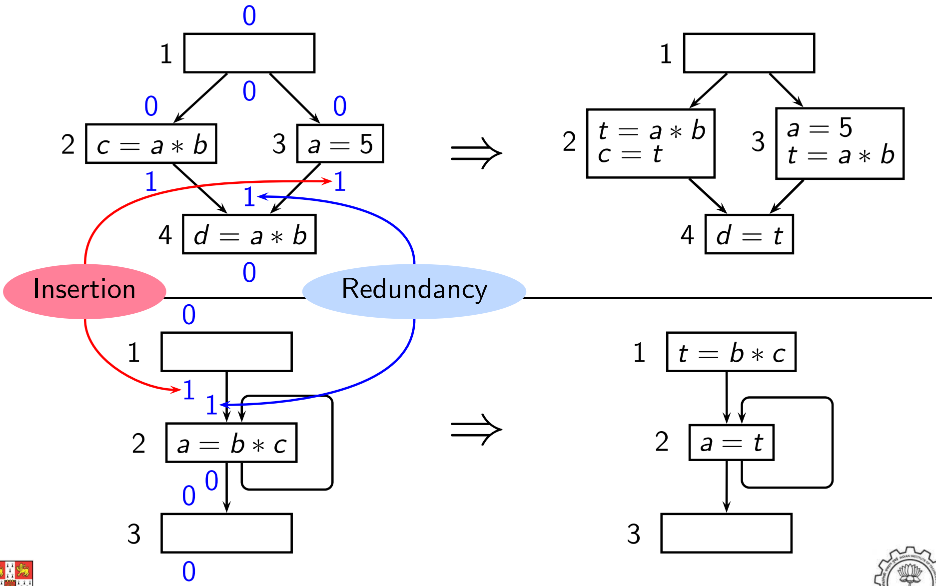


Performing PRE by Computing *In*/*Out*: Simple Cases

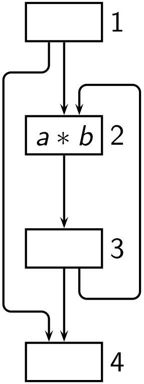




Performing PRE by Computing *In*/*Out*: Simple Cases

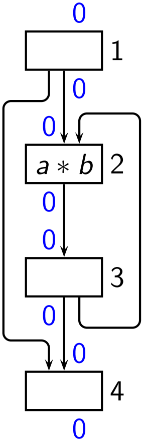


Tutorial Problems for PRE



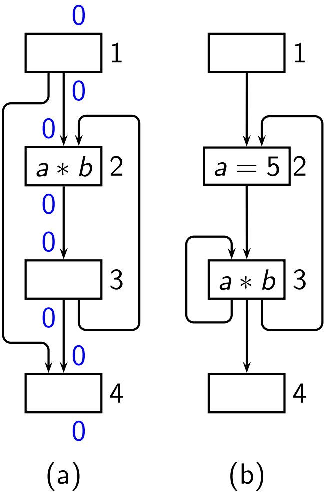
(a)

Tutorial Problems for PRE

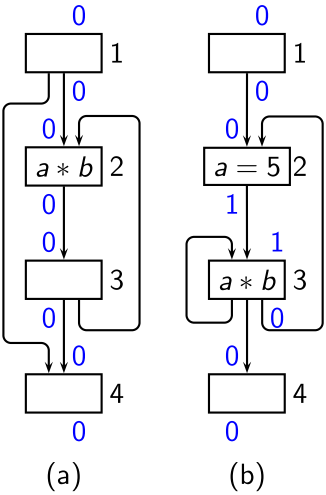


(a)

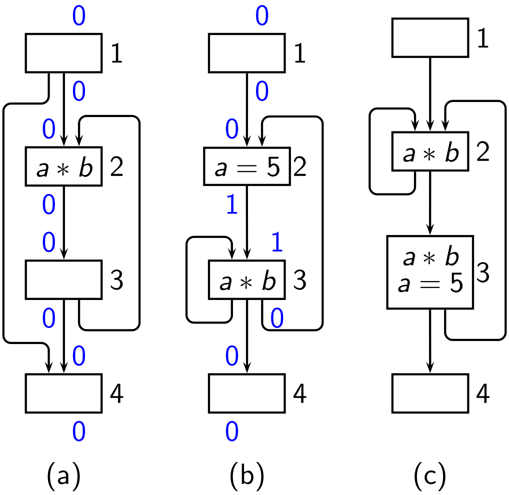
Tutorial Problems for PRE



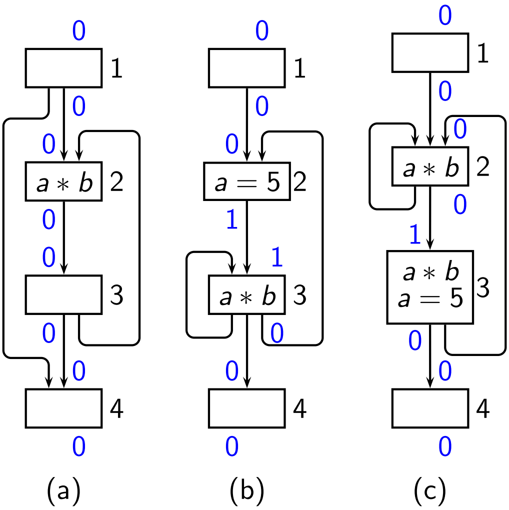
Tutorial Problems for PRE



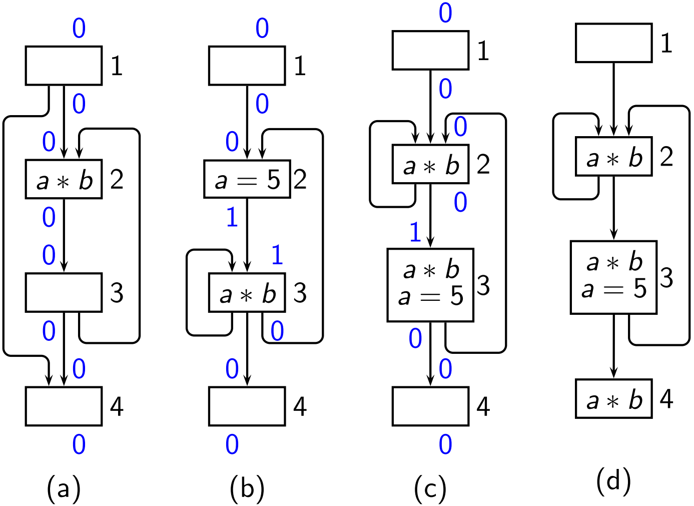
Tutorial Problems for PRE



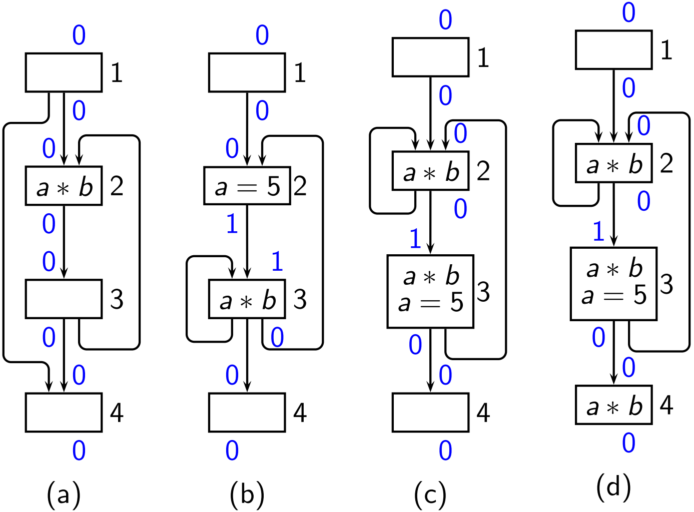
Tutorial Problems for PRE



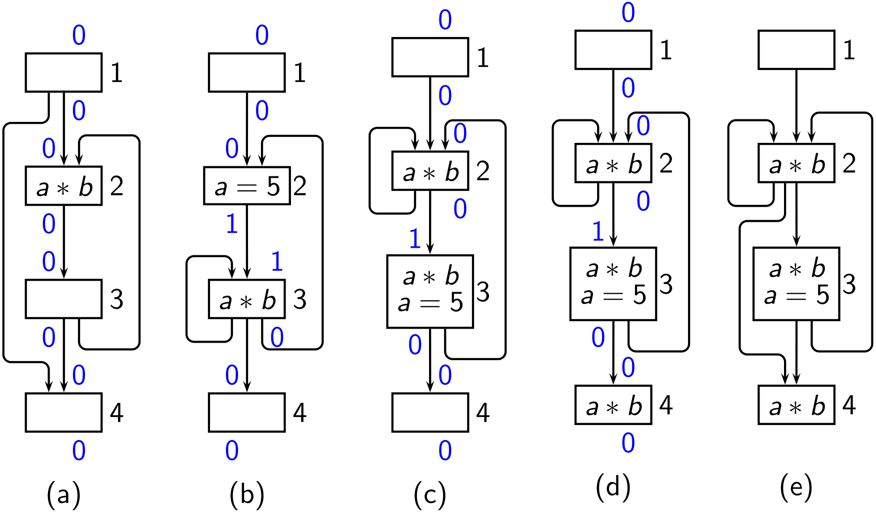
Tutorial Problems for PRE



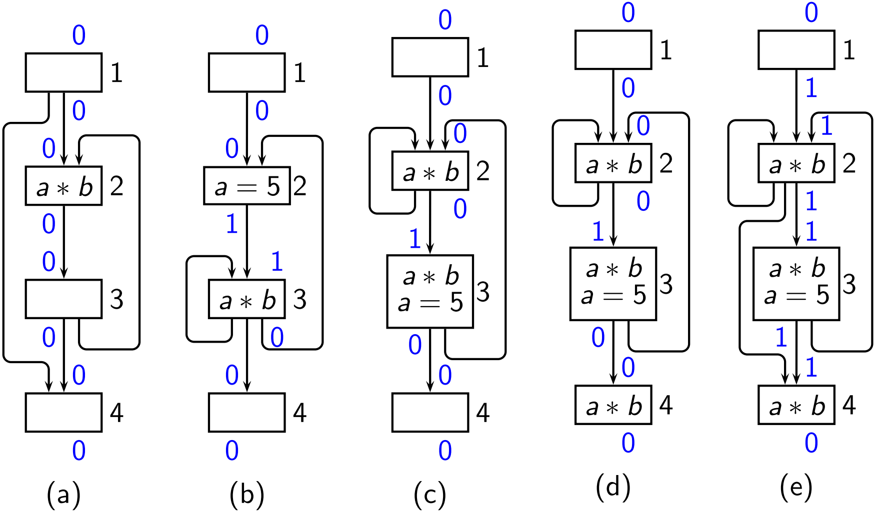
Tutorial Problems for PRE



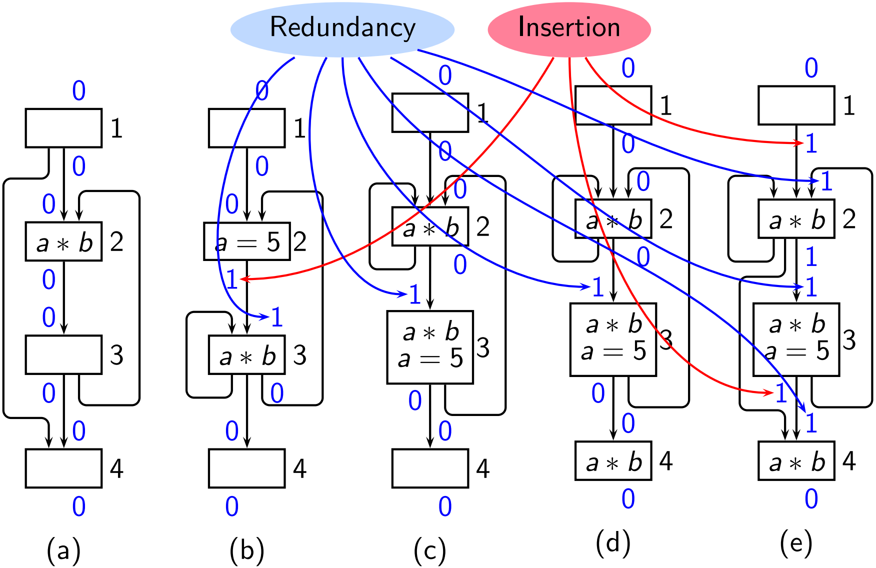
Tutorial Problems for PRE



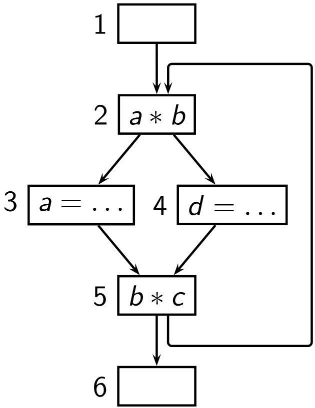
Tutorial Problems for PRE



Tutorial Problems for PRE



Further Tutorial Problem for PRE



Let $\{a * b, b * c\} \equiv$ bit string 11

Node n	$Kill_n$	$AntGen_n$	$PavIn_n$	$AvOut_n$
1	00	00	00	00
2	00	10	11	10
3	10	00	11	00
4	00	00	11	10
5	00	01	11	01
6	00	00	11	01

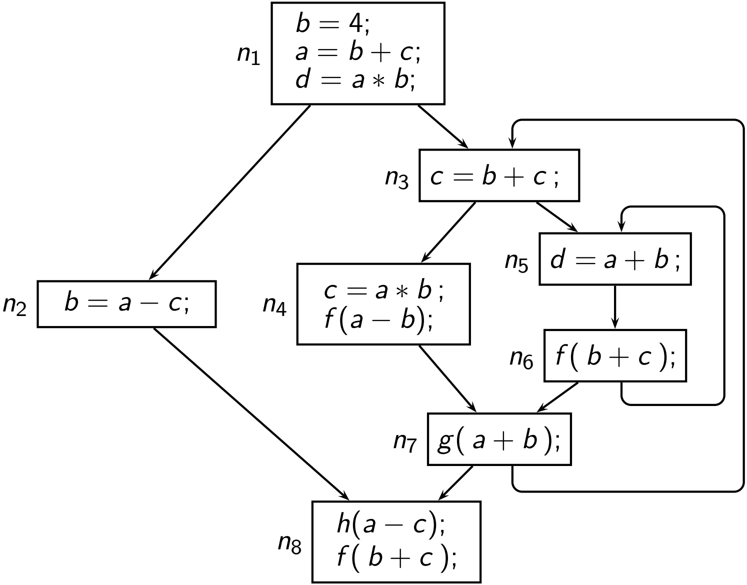
- Compute $In_n/Out_n/Redundant_n/Insert_n$
- Identify hoisting paths

Result of PRE Data Flow Analysis of the Running Example

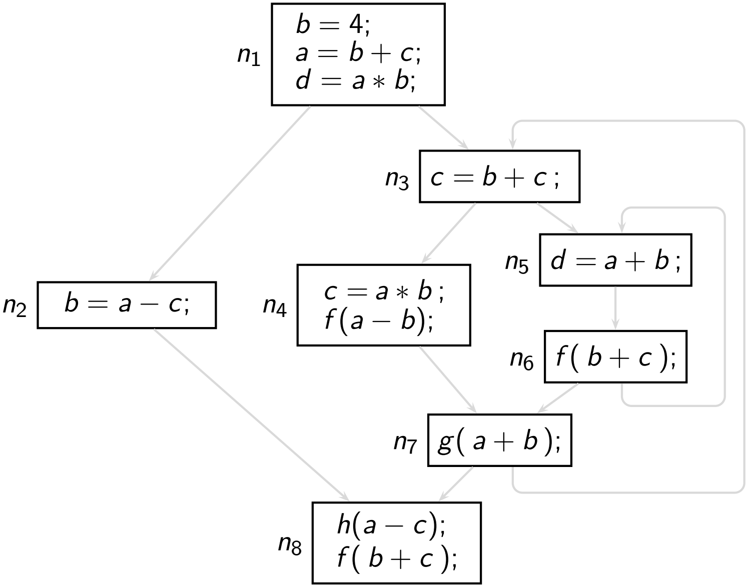
Bit vector	$a * b$	$a + b$	$a - b$	$a - c$	$b + c$

Block	Global Information							
	Constant information		Iteration # 1		Changes in iteration # 2		Changes in iteration # 3	
	$PavIn_n$	$AvOut_n$	Out_n	In_n	Out_n	In_n	Out_n	In_n
n_8	11111	00011	00000	00011				00001
n_7	11101	11000	00011	01001	00001			
n_6	11101	11001	01001	01001			01000	
n_5	11101	11000	01001	01001		01000		
n_4	11100	10100	01001	11100		11000		
n_3	11101	10000	01000	01001		00001		
n_2	10001	00010	00011	00000			00001	
n_1	00000	10001	00000	00000				

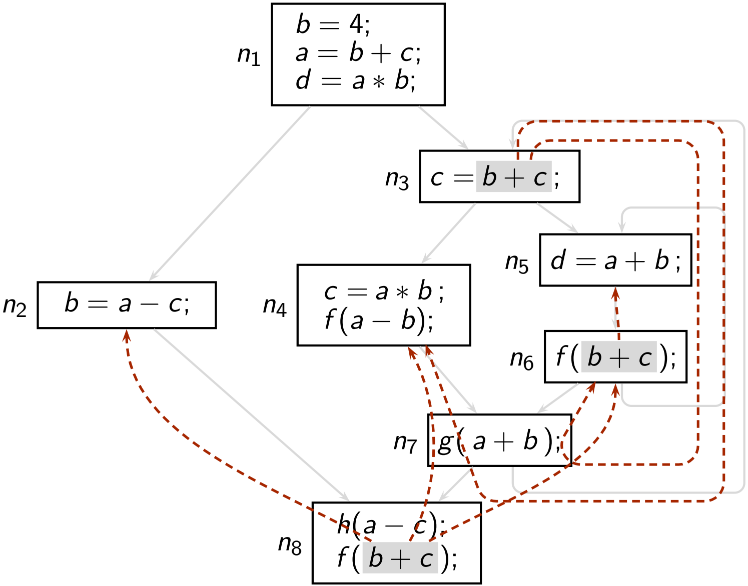
Hoisting Paths for Some Expressions in the Running Example



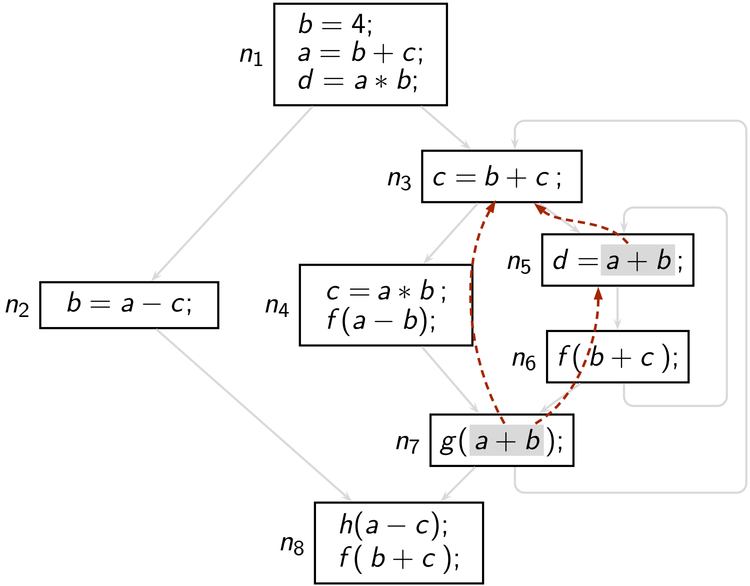
Hoisting Paths for Some Expressions in the Running Example



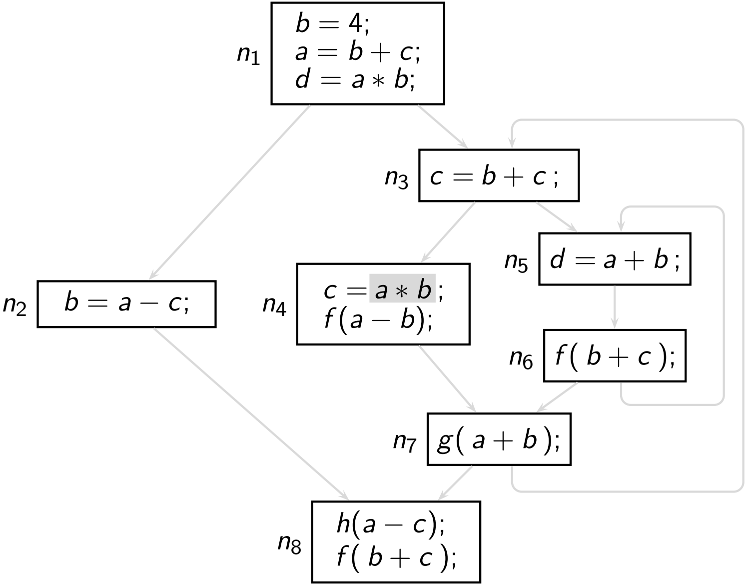
Hoisting Paths for Some Expressions in the Running Example



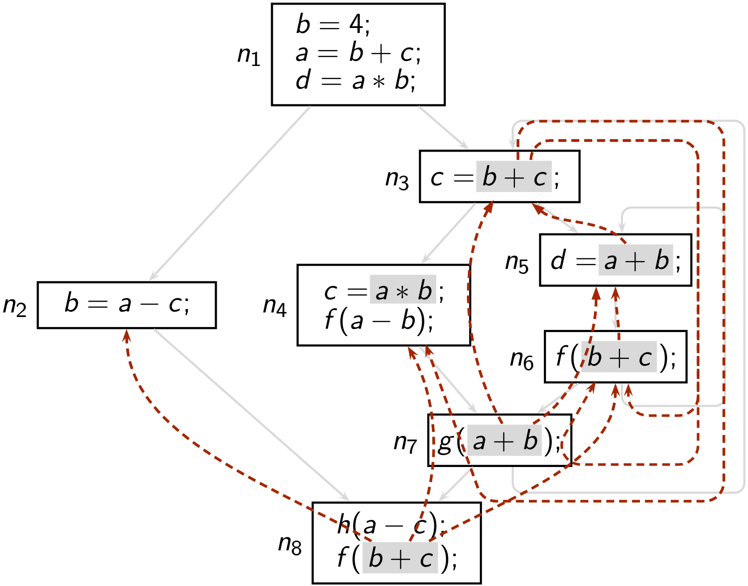
Hoisting Paths for Some Expressions in the Running Example



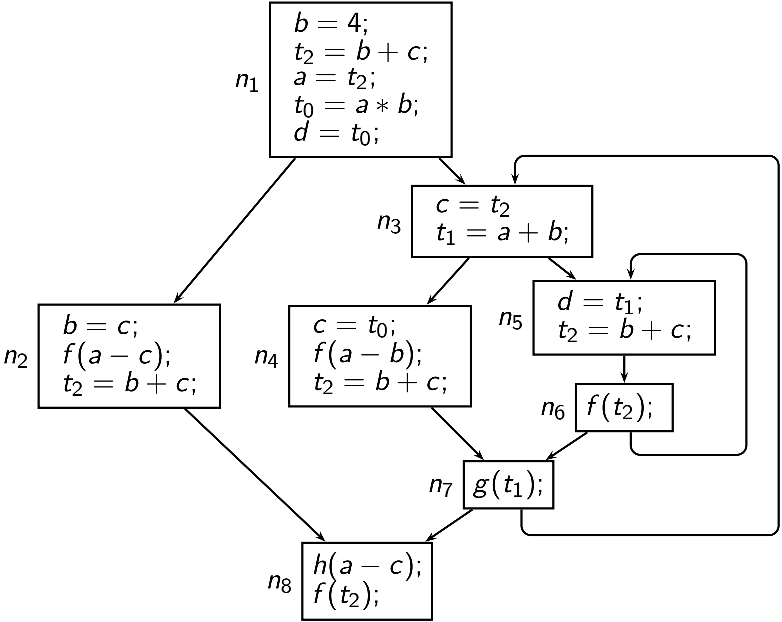
Hoisting Paths for Some Expressions in the Running Example



Hoisting Paths for Some Expressions in the Running Example



Optimized Version of the Running Example



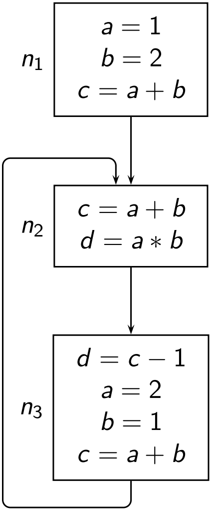
Part 3

The Need for a More General Setting

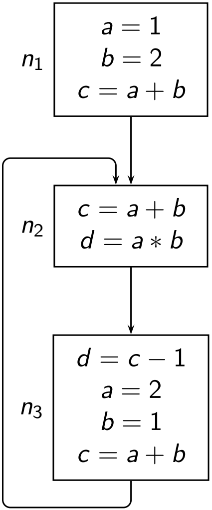
What We Have Seen So Far . . .

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

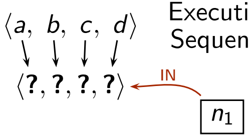
An Introduction to Constant Propagation



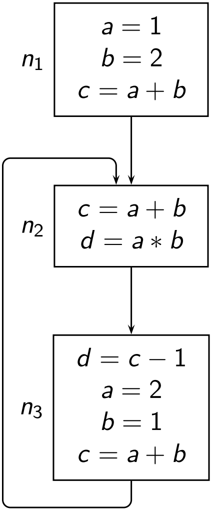
An Introduction to Constant Propagation

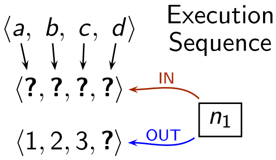


Execution Sequence

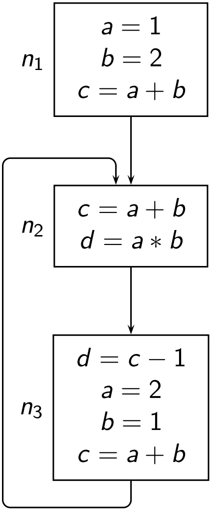


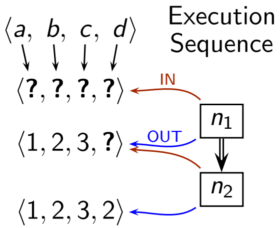
An Introduction to Constant Propagation



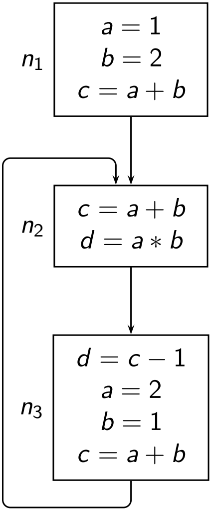


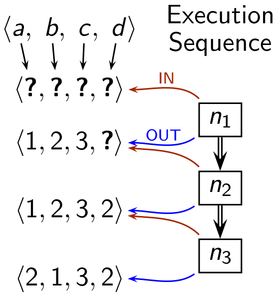
An Introduction to Constant Propagation



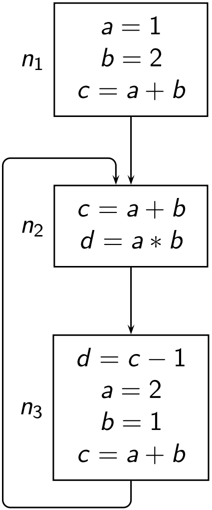


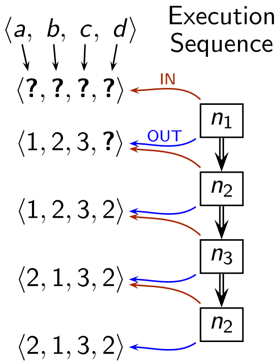
An Introduction to Constant Propagation



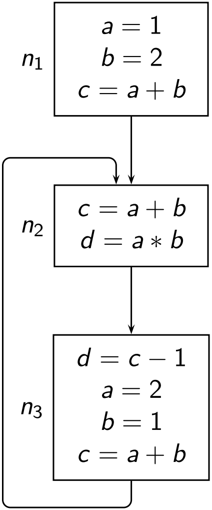


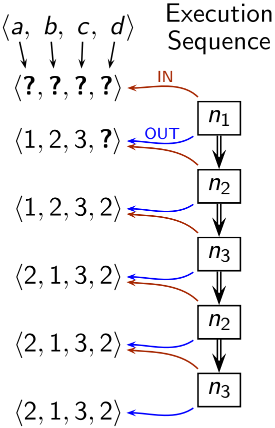
An Introduction to Constant Propagation



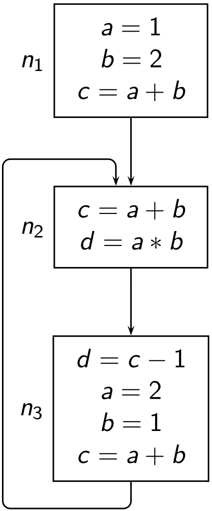


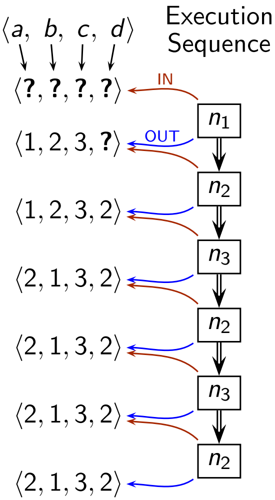
An Introduction to Constant Propagation



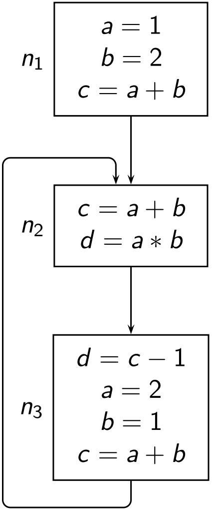


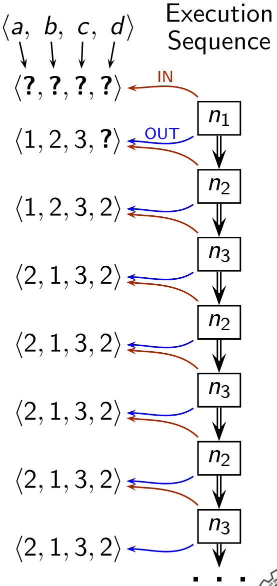
An Introduction to Constant Propagation



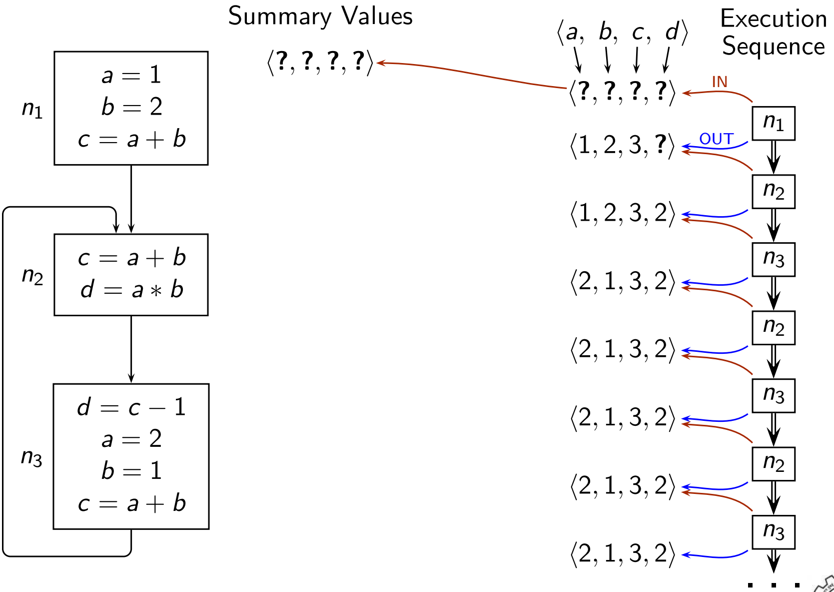


An Introduction to Constant Propagation

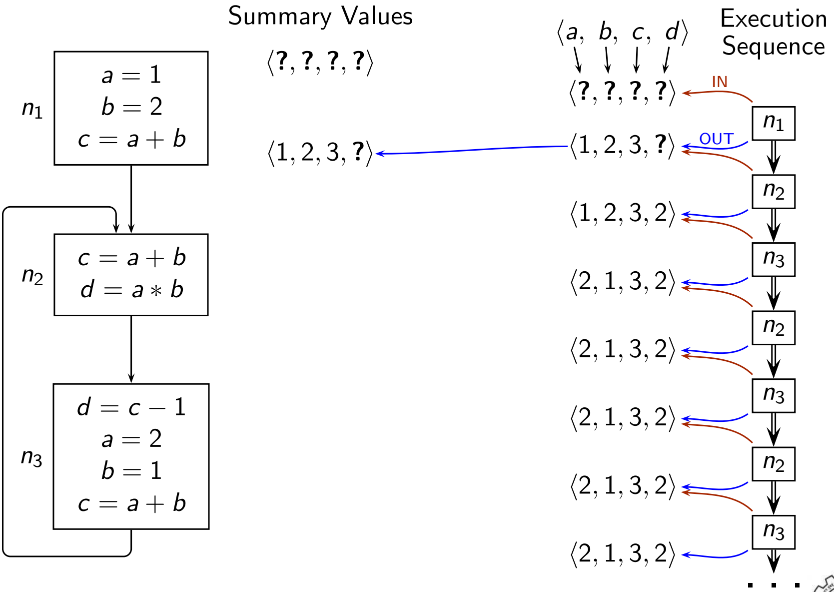




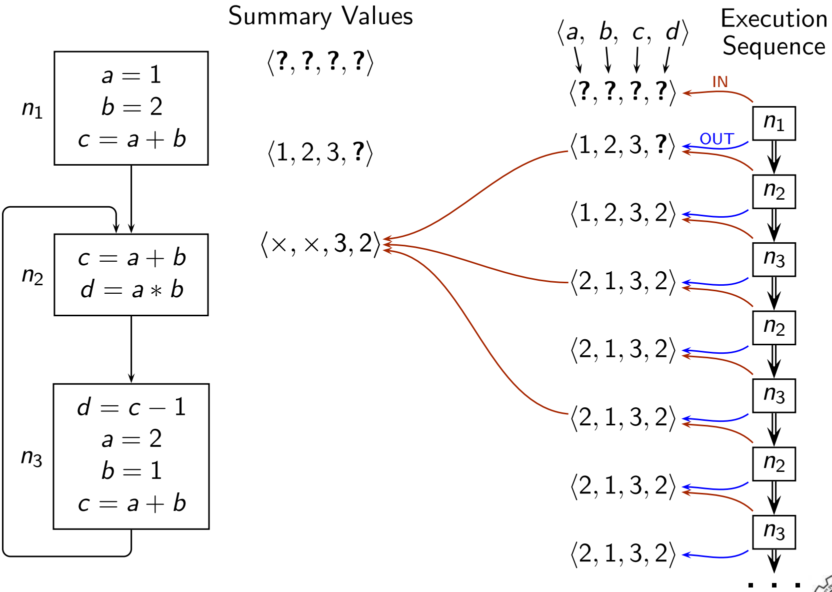
An Introduction to Constant Propagation



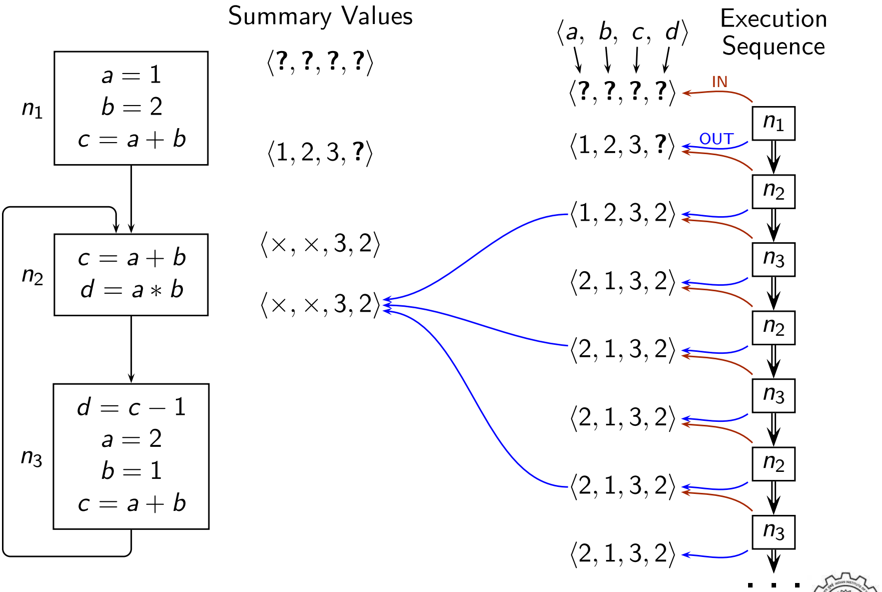
An Introduction to Constant Propagation



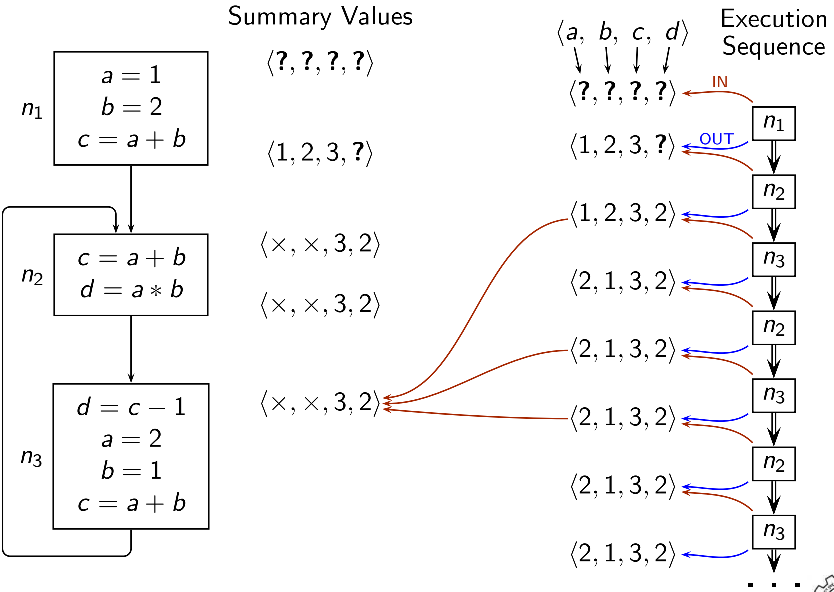
An Introduction to Constant Propagation



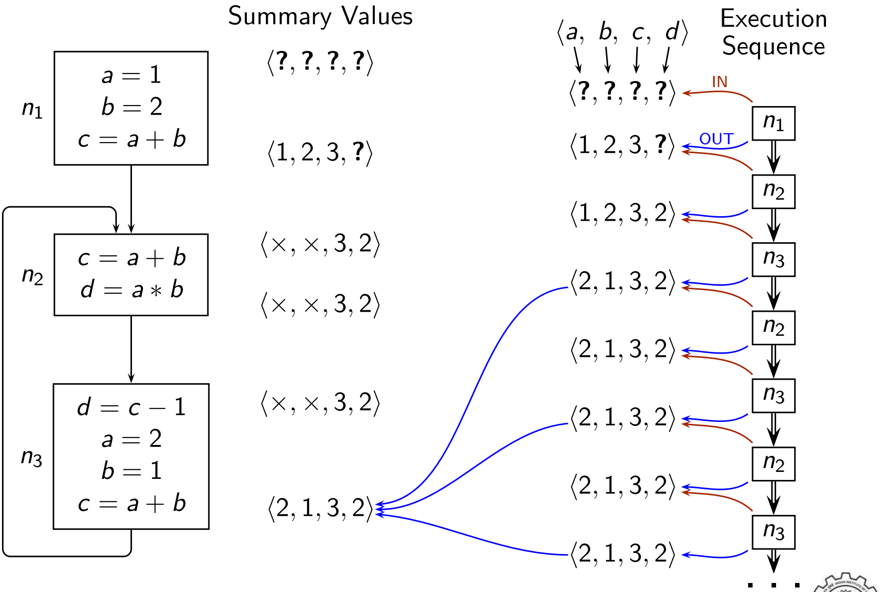
An Introduction to Constant Propagation



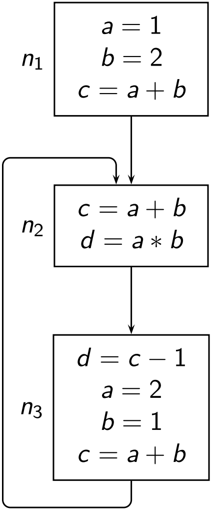
An Introduction to Constant Propagation



An Introduction to Constant Propagation



An Introduction to Constant Propagation



Summary Values

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

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

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

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

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

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

Desired Solution

Data Flow Values for Constant Propagation

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

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

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

Confluence Operation for Constant Propagation

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

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

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

  What are its properties?

Flow Functions for Constant Propagation

- Flow function for $r = a_1 * a_2$

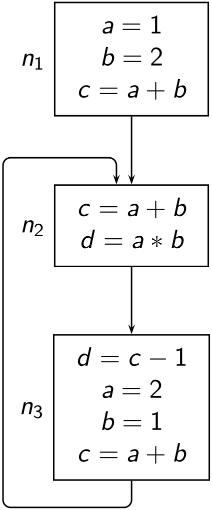
$mult$	$\langle a_1, \mathbf{?} \rangle$	$\langle a_1, \times \rangle$	$\langle a_1, c_1 \rangle$
$\langle a_2, \mathbf{?} \rangle$	$\langle r, \mathbf{?} \rangle$	$\langle r, \times \rangle$	$\langle r, \mathbf{?} \rangle$
$\langle a_2, \times \rangle$	$\langle r, \times \rangle$	$\langle r, \times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

- This cannot be expressed in the form

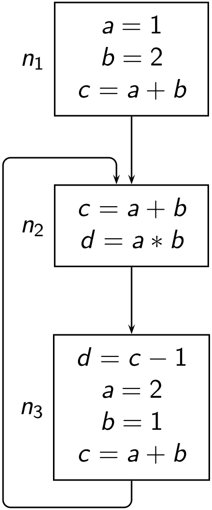
$$f_n(X) = Gen_n \cup (X - Kill_n)$$

where Gen_n and $Kill_n$ are constant effects of block n.

Round Robin Iterative Analysis for Constant Propagation

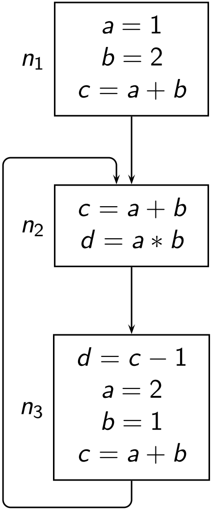


Round Robin Iterative Analysis for Constant Propagation



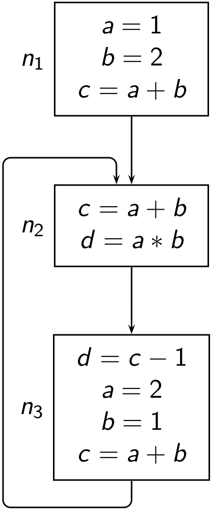
Iteration #1
$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle$
$\langle 1, 2, 3, \mathbf{?} \rangle$
$\langle 1, 2, 3, \mathbf{?} \rangle$
$\langle 1, 2, 3, 2 \rangle$
$\langle 1, 2, 3, 2 \rangle$
$\langle 2, 1, 3, 2 \rangle$

Round Robin Iterative Analysis for Constant Propagation



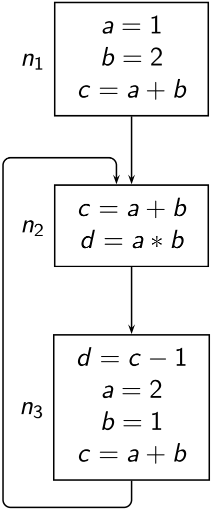
Program point	Iteration #1	Iteration #2
Entry of n_1	$\langle \textbf{?}, \textbf{?}, \textbf{?}, \textbf{?} \rangle$	$\langle \textbf{?}, \textbf{?}, \textbf{?}, \textbf{?} \rangle$
Exit of n_1	$\langle 1, 2, 3, \textbf{?} \rangle$	$\langle 1, 2, 3, \textbf{?} \rangle$
Entry of n_2	$\langle 1, 2, 3, \textbf{?} \rangle$	$\langle \times, \times, 3, 2 \rangle$
Exit of n_2	$\langle 1, 2, 3, 2 \rangle$	$\langle \times, \times, \times, \times \rangle$
Entry of n_3	$\langle 1, 2, 3, 2 \rangle$	$\langle \times, \times, \times, \times \rangle$
Exit of n_3	$\langle 2, 1, 3, 2 \rangle$	$\langle 2, 1, 3, \times \rangle$

Round Robin Iterative Analysis for Constant Propagation



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

Round Robin Iterative Analysis for Constant Propagation

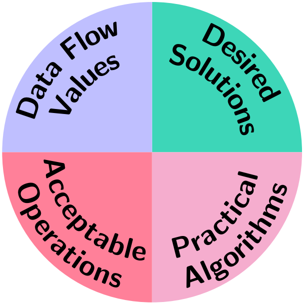


Iteration #1	Iteration #2	Iteration #3	Desired solution
$\langle ?, ?, ?, ? \rangle$	$\langle ?, ?, ?, ? \rangle$	$\langle ?, ?, ?, ? \rangle$	$\langle ?, ?, ?, ? \rangle$
$\langle 1, 2, 3, ? \rangle$	$\langle 1, 2, 3, ? \rangle$	$\langle 1, 2, 3, ? \rangle$	$\langle 1, 2, 3, ? \rangle$
$\langle 1, 2,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

Round Robin Iterative Analysis for Constant Propagation

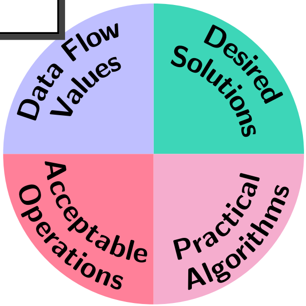
Node	Statements		Iteration #1	Iteration #2	Iteration #3	Desired solution
n_1	$a = 1$ $b = 2$ $c = a + b$	In	$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle$	$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle$	$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle$	$\langle \mathbf{?}, \mathbf{?}, \mathbf{?}, \mathbf{?} \rangle$
		Out	$\langle 1, 2, 3, \mathbf{?} \rangle$	$\langle 1, 2, 3, \mathbf{?} \rangle$	$\langle 1, 2, 3, \mathbf{?} \rangle$	$\langle 1, 2, 3, \mathbf{?} \rangle$
n_2	$c = a + b$ $d = a * b$	In	$\langle 1, 2, 3, \mathbf{?} \rangle$	$\langle \times, \times, 3, 2 \rangle$	$\langle \times, \times, 3, \times \rangle$	$\langle \times, \times, 3, 2 \rangle$
		Out	$\langle 1, 2, 3, 2 \rangle$	$\langle \times, \times, \times, \times \rangle$	$\langle \times, \times, \times, \times \rangle$	$\langle \times, \times, 3, 2 \rangle$
n_3	$d = c - 1$ $a = 2$ $b = 1$ $c = a + b$	In	$\langle 1, 2, 3, 2 \rangle$	$\langle \times, \times, \times, \times \rangle$	$\langle \times, \times, \times, \times \rangle$	$\langle \times, \times, 3, 2 \rangle$
		Out	$\langle 2, 1, 3, 2 \rangle$	$\langle 2, 1, 3, \times \rangle$	$\langle 2, 1, 3, \times \rangle$	$\langle 2, 1, 3, 2 \rangle$

Issues in Data Flow Analysis



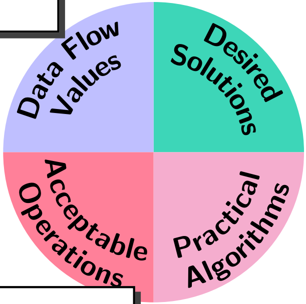
Issues in Data Flow Analysis

- Representation
- Approximation: Partial Order, Lattices



Issues in Data Flow Analysis

- Representation
- Approximation: Partial Order, Lattices

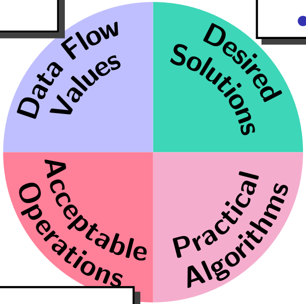


- Merge: Commutativity, Associativity, Idempotence
- Flow Functions: Monotonicity, Distributivity, Boundedness, Separability

Issues in Data Flow Analysis

- Representation
- Approximation: Partial Order, Lattices

- Existence
- Safety (soundness)
- Precision

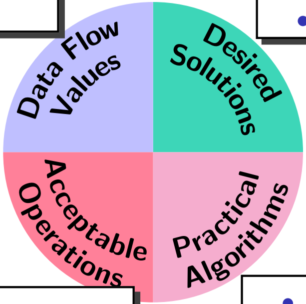


- Merge: Commutativity, Associativity, Idempotence
- Flow Functions: Monotonicity, Distributivity, Boundedness, Separability

Issues in Data Flow Analysis

- Representation
- Approximation: Partial Order, Lattices

- Existence
- Safety (soundness)
- Precision



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

- Complexity, efficiency
- Convergence
- Initialization

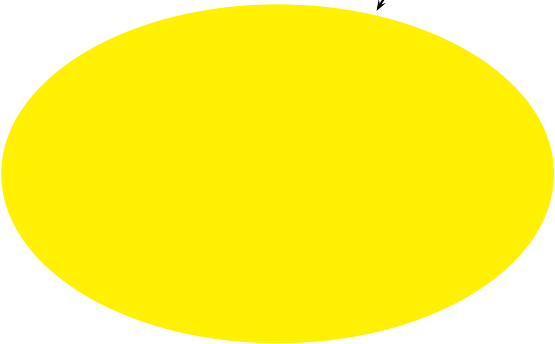
Part 4

Data Flow Values: An Overview

Data Flow Values: An Outline of Our Discussion

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

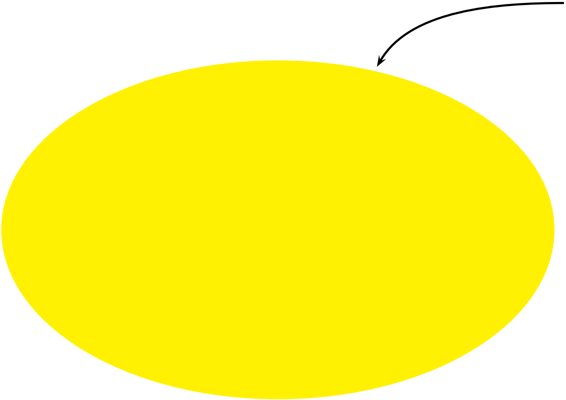
Partially Ordered Sets and Lattices



Partially ordered sets

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

Partially Ordered Sets and Lattices



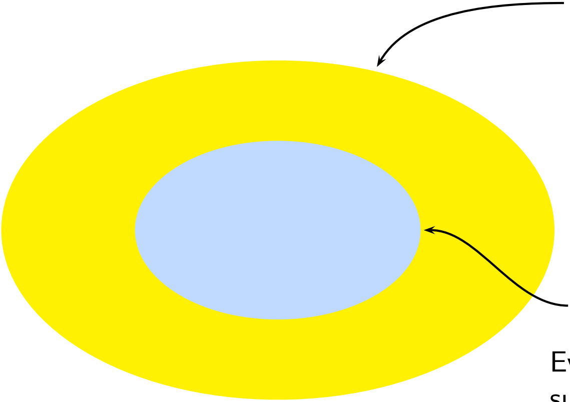
Partially ordered sets

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

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

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

Partially Ordered Sets and Lattices



Partially ordered sets

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

Lattices

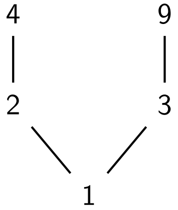
Every non-empty finite subset has a greatest lower bound (glb) and a least upper bound (lub)

Partially Ordered Sets

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

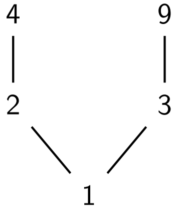
Partially Ordered Sets

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



Partially Ordered Sets

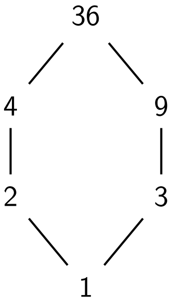
Set $\{1, 2, 3, 4, 9\}$ with $\sqsubseteq$ relation as "divides" (i.e. $a \sqsubseteq b$ iff a divides b)



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

Lattice

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



Complete Lattice

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

  Example:
  Lattice $\mathbb{Z}$ of integers under $\leq$ relation. All finite subsets have a glb and a lub. Infinite subsets do not have a glb or a lub.

Complete Lattice

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

  Example:
  Lattice $\mathbb{Z}$ of integers under $\leq$ relation. All finite subsets have a glb and a lub. Infinite subsets do not have a glb or a lub.

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

Complete Lattice

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

  Example:
  Lattice $\mathbb{Z}$ of integers under $\leq$ relation. All finite subsets have a glb and a lub. Infinite subsets do not have a glb or a lub.

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

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

Complete Lattice

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

  Example:
  Lattice $\mathbb{Z}$ of integers under $\leq$ relation. All finite subsets have a glb and a lub. Infinite subsets do not have a glb or a lub.

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

  Example:
  Lattice $\mathbb{Z}$ of integers under $\leq$ relation with ∞ and $-\infty$.
  - ∞ is the top element denoted $\top$: $\forall i \in \mathbb{Z},\ i \leq \top$.
  - $-\infty$ is the bottom element denoted $\bot$: $\forall i \in \mathbb{Z},\ \bot \leq i$.

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

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

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

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

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

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

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

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

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

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

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

    Every element of $\mathbb{Z} \cup \{\infty, -\infty\}$ is vacuously a lower bound of an element in $\emptyset$ (because there is no element in $\emptyset$).
    The greatest among these lower bounds is $\top$.

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

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

      Every element of $\mathbb{Z} \cup \{\infty, -\infty\}$ is vacuously a lower bound of an element in $\emptyset$ (because there is no element in $\emptyset$).
      The greatest among these lower bounds is $\top$.
    - lub($\emptyset$) is $\perp$

Finite Lattices are Complete

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

Available Expressions Analysis

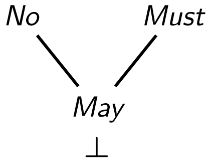
$$\begin{tikzcd}[column sep=small]
 & (\top) & \\
 & \{e_1, e_2, e_3\} \arrow[dl] \arrow[d] \arrow[dr] & \\
\{e_1, e_2\} \arrow[d] \arrow[dr] & \{e_1, e_3\} \arrow[dl] \arrow[dr] & \{e_2, e_3\} \arrow[dl] \arrow[d] \\
\{e_1\} \arrow[dr] & \{e_2\} \arrow[d] & \{e_3\} \arrow[dl] \\
 & \emptyset & \\
 &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

Partially Available Expressions Analysis

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

Lattice for May-Must Analysis

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



Interpreting data flow values

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

- An artificial $\top$ can be added
  However, a lub may not exist for arbitrary sets

Some Variants of Lattices

A poset L is

- A lattice iff each non-empty finite subset of L has a glb and lub.
- A complete lattice iff each subset of L has a glb and lub.
- A meet semilattice iff each non-empty finite subset of L has a glb.
- A join semilattice iff each non-empty finite subset of L has a lub.

Ascending and Descending Chains

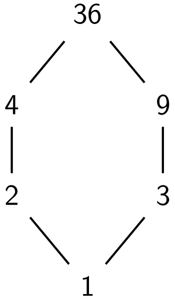
- Strictly ascending chain. $x \sqsubset y \sqsubset \cdots \sqsubset z$
- Strictly descending chain. $x \sqsupset y \sqsupset \cdots \sqsupset z$
- DCC: Descending Chain Condition
  All strictly descending chains are finite.
- ACC: Ascending Chain Condition
  All strictly ascending chains are finite.

Complete Lattice and Ascending and Descending Chains

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

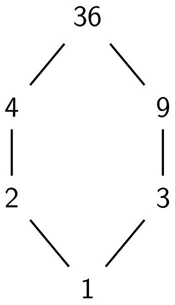
Operations on Lattices

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



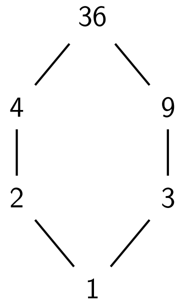
Operations on Lattices

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



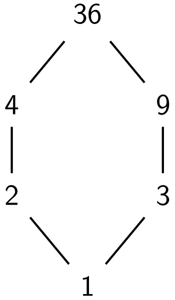
Operations on Lattices

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



Operations on Lattices

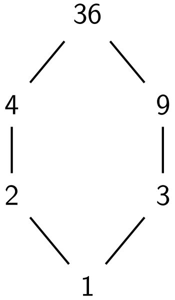
- Meet ($\sqcap$) and Join ($\sqcup$)
  - $x \sqcap y$ computes the glb of x and y.
    $z = x \sqcap y \Rightarrow z \sqsubseteq x \land z \sqsubseteq y$
  - $x \sqcup y$ computes the lub of x and y.
    $z = x \sqcup y \Rightarrow z \sqsupseteq x \land z \sqsupseteq y$
  - $\sqcap$ and $\sqcup$ are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



Operations on Lattices

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

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

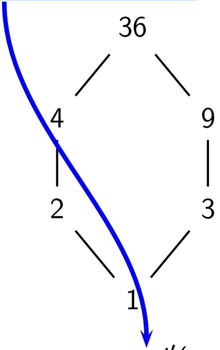


Operations on Lattices

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

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

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



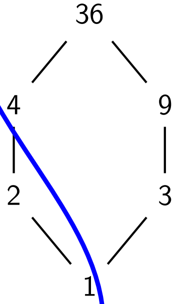
$x \sqcap y = gcd'(x, y)$

Operations on Lattices

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

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

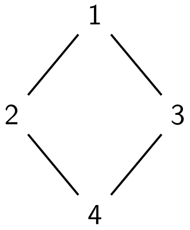
Greatest common divisor (or highest common factor) **in the lattice**



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

Lowest common multiple **in the lattice**

Cartesian Product of Lattice



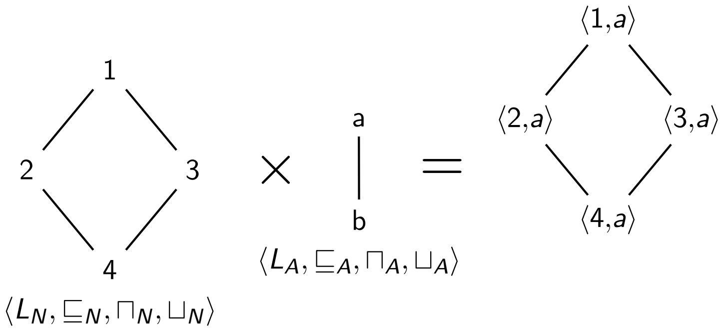
$\langle L_N, \sqsubseteq_N, \sqcap_N, \sqcup_N \rangle$

$\times$

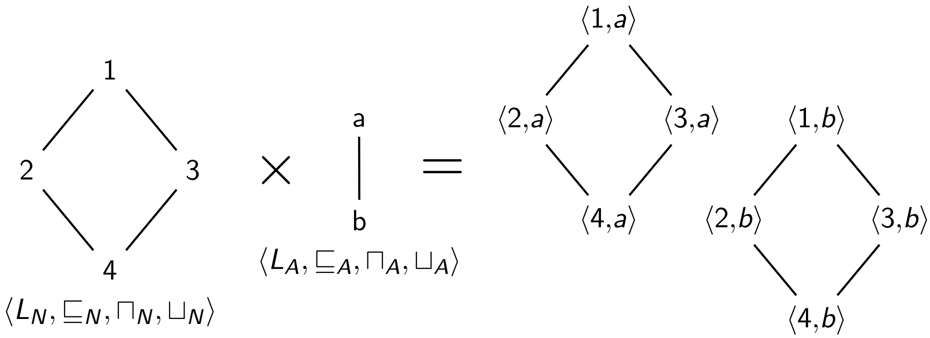
$\langle L_A, \sqsubseteq_A, \sqcap_A, \sqcup_A \rangle$

$=$

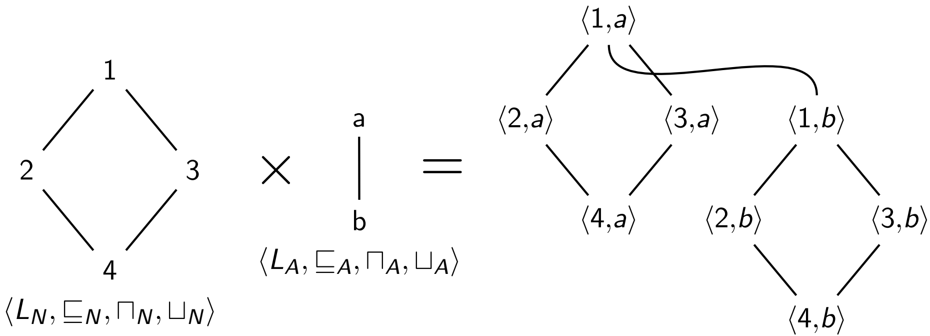
Cartesian Product of Lattice



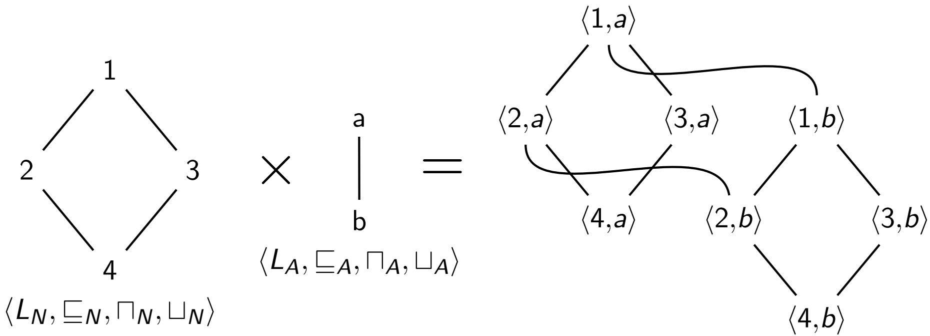
Cartesian Product of Lattice



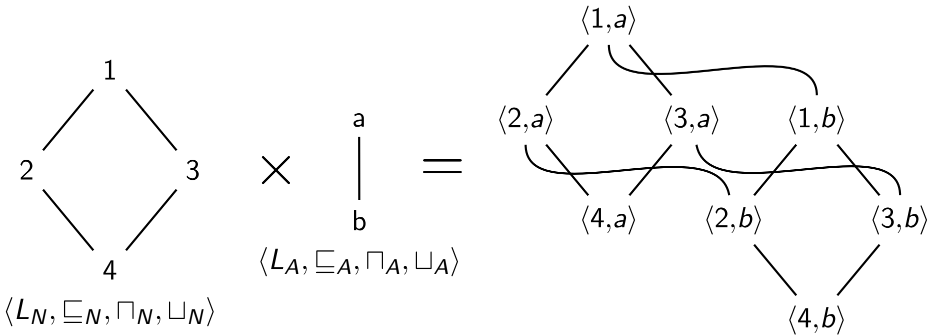
Cartesian Product of Lattice



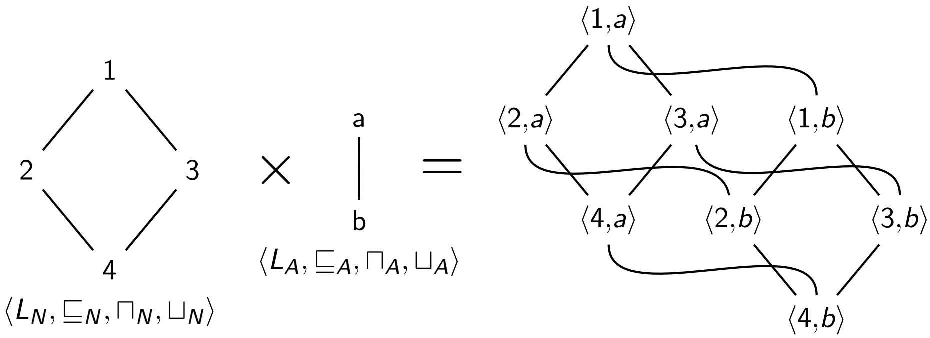
Cartesian Product of Lattice



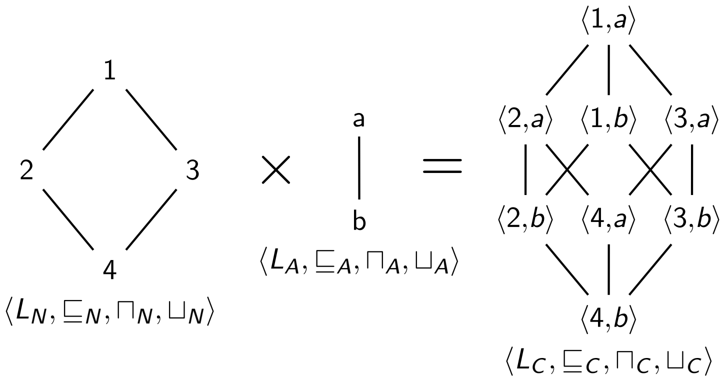
Cartesian Product of Lattice



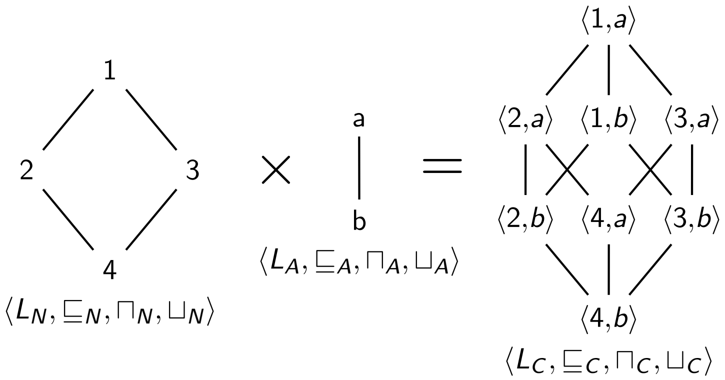
Cartesian Product of Lattice



Cartesian Product of Lattice



Cartesian Product of Lattice



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

The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- glb must exist for all non-empty finite subsets

The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

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

The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- glb must exist for all non-empty finite subsets
- $\perp$ must exist
  What guarantees the presence of $\perp$?
- $\top$ may not exist. Can be added artificially.

The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

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

  What guarantees the presence of $\perp$?

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

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

The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- glb must exist for all non-empty finite subsets
- $\perp$ must exist
  What guarantees the presence of $\perp$?
  - Assume that two maximal descending chains terminate at two incomparable elements x_1 and x_2
  - Since this is a meet semilattice, glb of $\{x_1, x_2\}$ must exist (say z).
- $\top$ may not exist. Can be added artificially.

The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

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

  What guarantees the presence of $\perp$?

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

    $\Rightarrow$ Neither of the chains is maximal.

    Both of them can be extended to include z.

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

The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

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

The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

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

  What guarantees the presence of $\perp$?

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

    $\Rightarrow$ Neither of the chains is maximal.

    Both of them can be extended to include z.
  - Extending this argument to all strictly descending chains, it is easy to see that $\perp$ must exist.
- $\top$ may not exist. Can be added artificially.
  - lub of arbitrary elements may not exist

The Set of Data Flow Values For Available Expressions Analysis

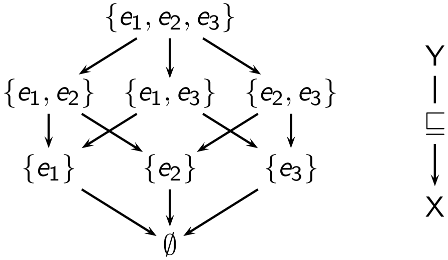
- The powerset of the universal set of expressions
- Partial order is the subset relation

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

Set View of the Lattice

The Set of Data Flow Values For Available Expressions Analysis

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



Set View of the Lattice

The Set of Data Flow Values For Available Expressions Analysis

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

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

Set View of the Lattice

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

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

Bit Vector View

The Concept of Approximation

- x approximates y *iff*

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

- Validity of approximation is context specific

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

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

Two Important Objectives in Data Flow Analysis

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

Two Important Objectives in Data Flow Analysis

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

Two Important Objectives in Data Flow Analysis

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

Context Determines the Validity of Approximations

Analysis	Application	Safe Approximation	Exhaustive Approximation
		May prohibit correct optimization	May enable wrong optimization

Context Determines the Validity of Approximations

May prohibit correct optimization

May enable wrong optimization

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

Context Determines the Validity of Approximations

May prohibit correct optimization

May enable wrong optimization

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

Context Determines the Validity of Approximations

May prohibit correct optimization

May enable wrong optimization

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

Spurious Inclusion

Spurious Exclusion

Partial Order Captures Approximation

- $\sqsubseteq$ captures valid approximations for safety

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

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

Partial Order Captures Approximation

- $\sqsubseteq$ captures valid approximations for safety

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

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

- $\sqsupseteq$ captures valid approximations for exhaustiveness

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

  - The data flow information represented by x contains every value contained in the data flow information represented by y
  - It may be unsafe, though.

Partial Order Captures Approximation

- $\sqsubseteq$ captures valid approximations for safety

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

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

- $\sqsupseteq$ captures valid approximations for exhaustiveness

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

  - The data flow information represented by x contains every value contained in the data flow information represented by y
  - It may be unsafe, though.

We want most exhaustive information which is also safe.

Most Approximate Values in a Complete Lattice

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

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

Most Approximate Values in a Complete Lattice

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

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

Most Approximate Values in a Complete Lattice

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

Most Approximate Values in a Complete Lattice

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

Most Approximate Values in a Complete Lattice

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

Most Approximate Values in a Complete Lattice

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

Appropriate orientation chosen by design.

Setting Up Lattices

Available Expressions Analysis	Live Variables Analysis
$\begin{tikzcd}[column sep=small] & & \{e_1,e_2,e_3\} \arrow[dll] \arrow[d] \arrow[drr] & & \\ \{e_1,e_2\} \arrow[d] \arrow[drr] & & \{e_1,e_3\} \arrow[dll] \arrow[drr] & & \{e_2,e_3\} \arrow[dll] \arrow[d] \\ \{e_1\} \arrow[drr] & & \{e_2\} \arrow[d] & & \{e_3\} \arrow[dll] \\ & & \emptyset & & \end{tikzcd}$	$\begin{tikzcd}[column sep=small] & & \emptyset \arrow[dll] \arrow[d] \arrow[drr] & & \\ \{v_1\} \arrow[d] \arrow[drr] & & \{v_2\} \arrow[dll] \arrow[drr] & & \{v_3\} \arrow[dll] \arrow[d] \\ \{v_1,v_2\} \arrow[drr] & & \{v_1,v_3\} \arrow[d] & & \{v_2,v_3\} \arrow[dll] \\ & & \{v_1,v_2,v_3\} & & \end{tikzcd}$
$\sqsubseteq$ is $\subseteq$	$\sqsubseteq$ is $\supseteq$
$\sqcap$ is $\cap$	$\sqcap$ is $\cup$

Partial Order Relation

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

Partial Order Relation

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

Merging Information

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

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

Merging Information

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

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

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

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

  Idempotent $x \sqcap x = x$

Merging Information

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

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

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

Merging Information

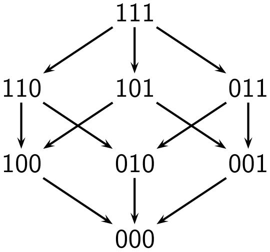
- $x \sqcap y$ computes the *greatest lower bound* of x and y i.e. largest z such that $z \sqsubseteq x$ and $z \sqsubseteq y$

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

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

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

More on Lattices in Data Flow Analysis

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

More on Lattices in Data Flow Analysis

L = Lattice for all expressions

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

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

$$\begin{tikzcd}
\text{(Expression } e \text{ is available)} \\
1 \text{ or } \{e\} \arrow[d] \\
0 \text{ or } \emptyset \\
\text{(Expressions } e \text{ is not available)}
\end{tikzcd}$$

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

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

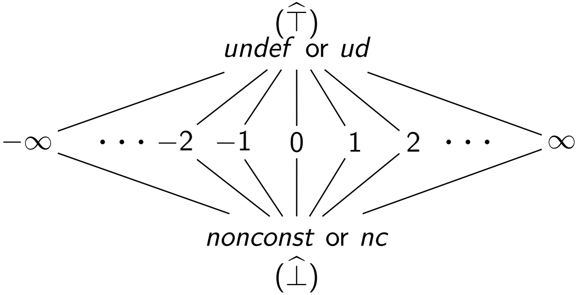
Component Lattice for Data Flow Information Represented By Bit Vectors

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

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

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

Component Lattice for Integer Constant Propagation



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

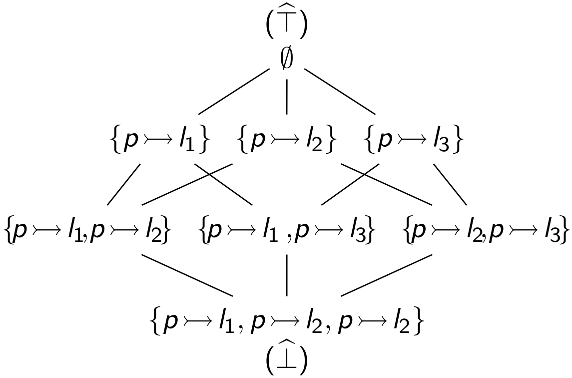
$\widehat{\sqcap}$	$\langle a, ud\rangle$	$\langle a, nc\rangle$	$\langle a, c_1\rangle$
$\langle a, ud\rangle$	$\langle a, ud\rangle$	$\langle a, nc\rangle$	$\langle a, c_1\rangle$
$\langle a, nc\rangle$	$\langle a, nc\rangle$	$\langle a, nc\rangle$	$\langle a, nc\rangle$
$\langle a, c_2\rangle$	$\langle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

Component Lattice for May Points-To Analysis

- Relation between pointer variables and locations in the memory.

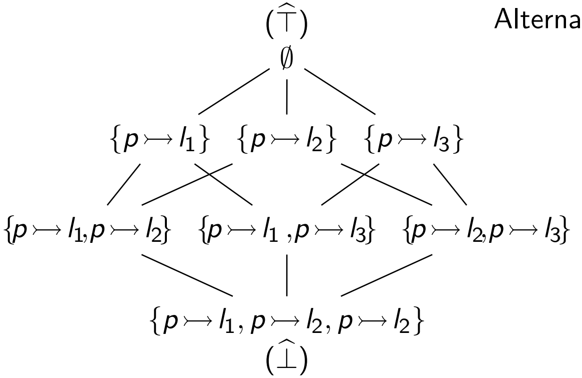
Component Lattice for May Points-To Analysis

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

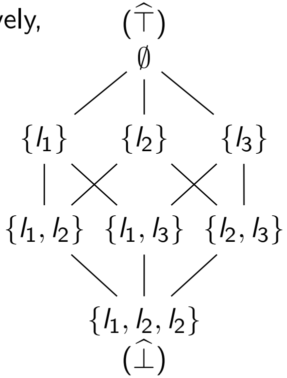


Component Lattice for May Points-To Analysis

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

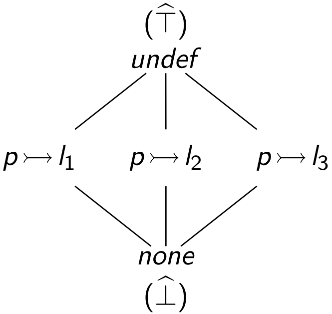


Alternatively,

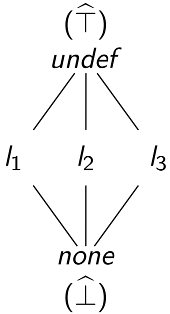


Component Lattice for Must Points-To Analysis

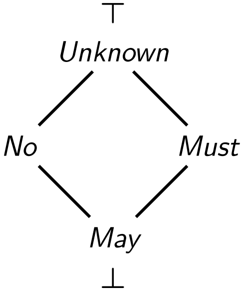
- A pointer can point to at most one location.



Alternatively,



General Lattice for May-Must Analysis



Interpreting data flow values

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

Possible Applications

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