Bit Vector Data Flow Frameworks

Uday Khedker

(www.cse.iitb.ac.in/˜uday)

Department of Computer Science and Engineering,

Indian Institute of Technology, Bombay

Jul 2017

Part 1

About These Slides

Copyright

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

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

  (Indian edition published by Ane Books in 2013)

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

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

Outline

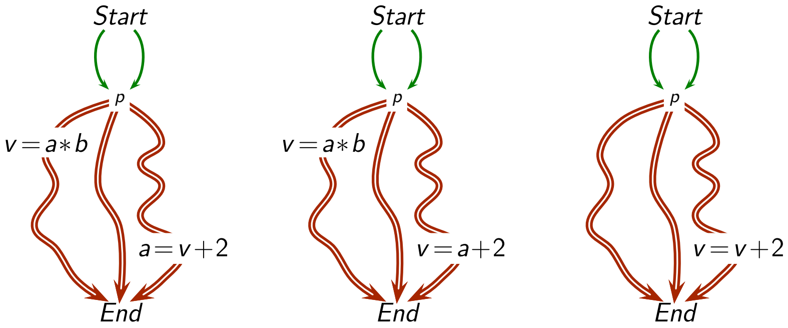
- Live Variables Analysis
- Observations about Data Flow Analysis
- Available Expressions Analysis
- Anticipable Expressions Analysis
- Reaching Definitions Analysis
- Common Features of Bit Vector Frameworks
- Partial Redundancy Elimination

Part 2

Live Variables Analysis

Defining Live Variables Analysis

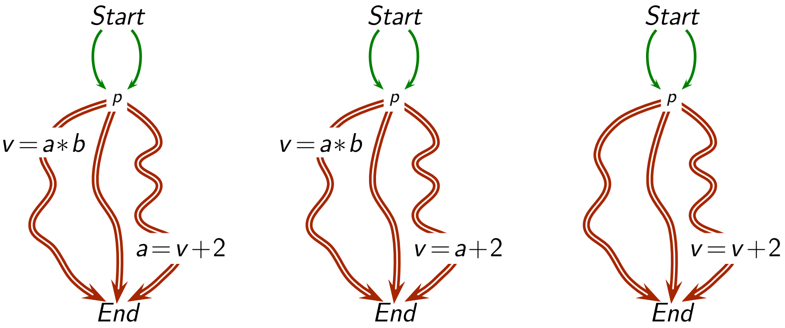
A variable v is live at a program point p, if some path from p to program exit contains an r-value occurrence of v which is not preceded by an l-value occurrence of v.



Defining Live Variables Analysis

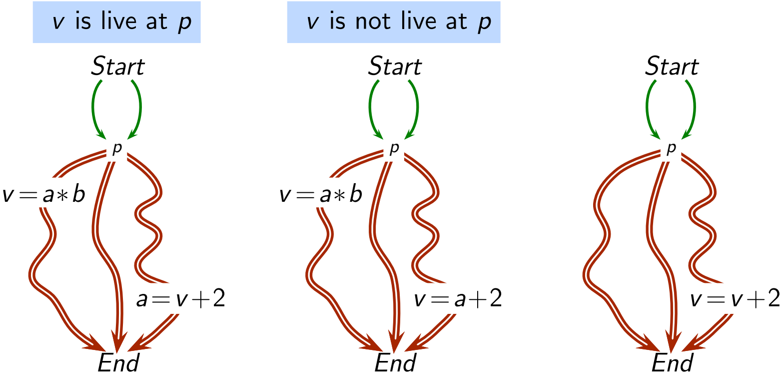
A variable v is live at a program point p, if some path from p to program exit contains an r-value occurrence of v which is not preceded by an l-value occurrence of v.

v is live at p



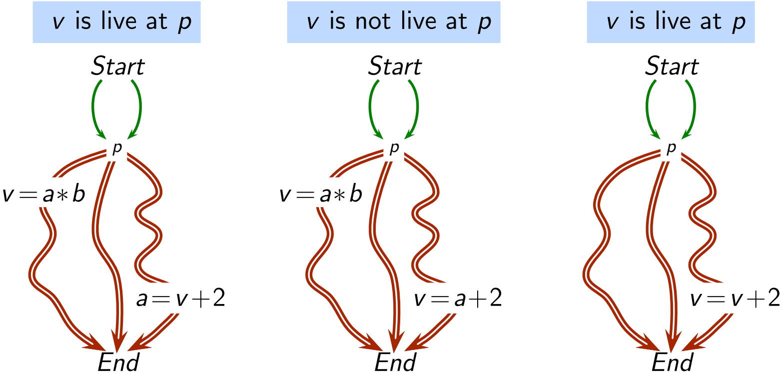
Defining Live Variables Analysis

A variable v is live at a program point p, if some path from p to program exit contains an r-value occurrence of v which is not preceded by an l-value occurrence of v.



Defining Live Variables Analysis

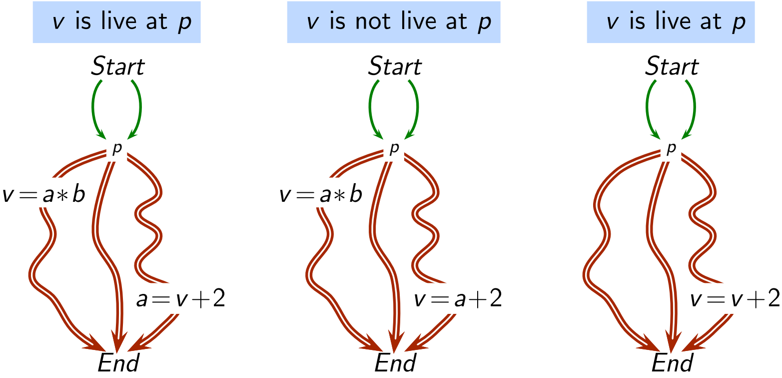
A variable v is live at a program point p, if some path from p to program exit contains an r-value occurrence of v which is not preceded by an l-value occurrence of v.



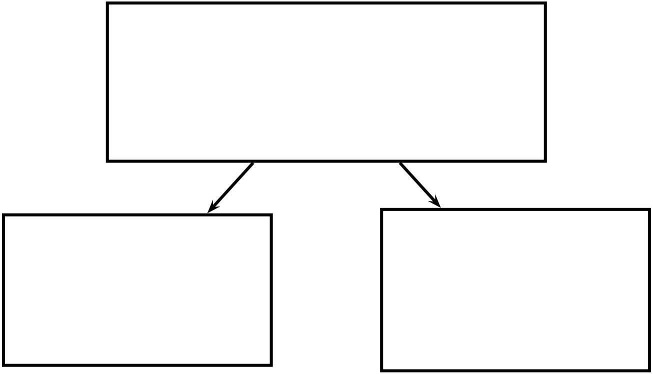
Defining Live Variables Analysis

A variable v is live at a program point p, if some path from p to program exit contains an r-value occurrence of v which is not preceded by an l-value occurrence of v.

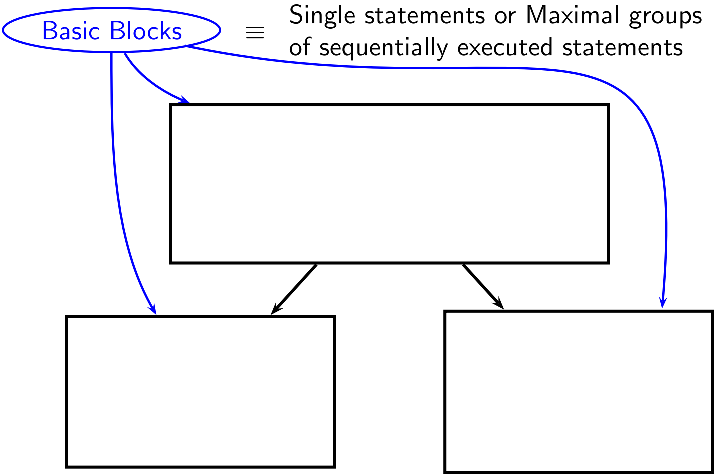
Path based specification



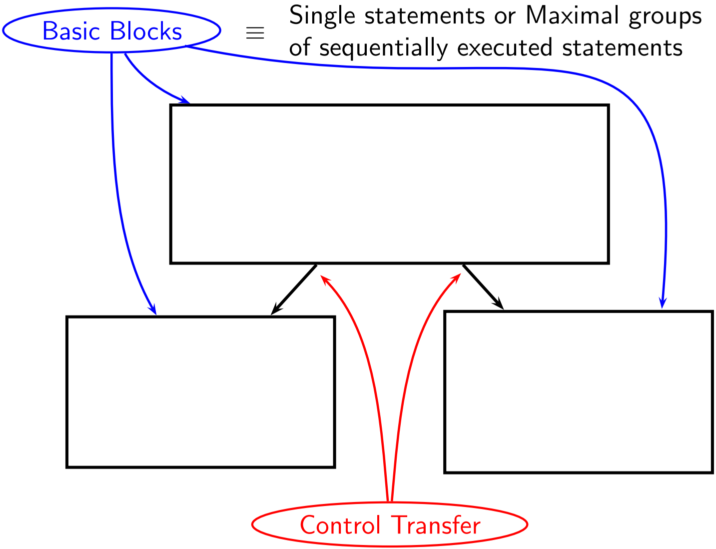
Defining Data Flow Analysis for Live Variables Analysis



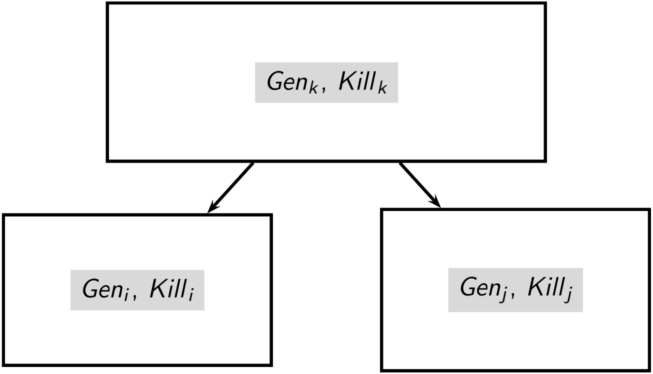
Defining Data Flow Analysis for Live Variables Analysis



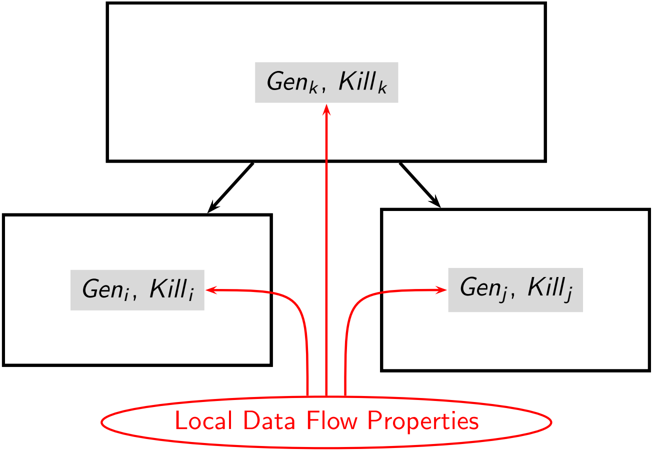
Defining Data Flow Analysis for Live Variables Analysis



Defining Data Flow Analysis for Live Variables Analysis



Defining Data Flow Analysis for Live Variables Analysis



Local Data Flow Properties for Live Variables Analysis

$$
\begin{aligned}
Gen_n &= \{\ v \mid \text{variable } v \text{ is used in basic block } n \text{ and} \\
&\qquad\quad \text{is not preceded by a definition of } v\ \} \\
Kill_n &= \{\ v \mid \text{basic block } n \text{ contains a definition of } v\ \}
\end{aligned}
$$

Local Data Flow Properties for Live Variables Analysis

r-value occurrence

Value is only read, e.g. x,y,z in

x.sum = y.data + z.data

$$Gen_n = \{ v \mid \text{variable } v \text{ is used in basic block } n \text{ and is not preceded by a definition of } v \}$$

$$Kill_n = \{ v \mid \text{basic block } n \text{ contains a definition of } v \}$$

Local Data Flow Properties for Live Variables Analysis

r-value occurrence

Value is only read, e.g. x,y,z in

x.sum = y.data + z.data

l-value occurrence

Value is modified e.g. y in

y = x.lptr

$$Gen_n = \{ v \mid \text{variable } v \text{ is used in basic block } n \text{ and is not preceded by a definition of } v \}$$

$$Kill_n = \{ v \mid \text{basic block } n \text{ contains a definition of } v \}$$

Local Data Flow Properties for Live Variables Analysis

r-value occurrence

Value is only read, e.g. x,y,z in

x.sum = y.data + z.data

l-value occurrence

Value is modified e.g. y in

y = x.lptr

$$Gen_n = \{ v \mid \text{variable } v \text{ is used in basic block } n \text{ and is not preceded by a definition of } v \}$$

$$Kill_n = \{ v \mid \text{basic block } n \text{ contains a definition of } v \}$$

within n

Local Data Flow Properties for Live Variables Analysis

r-value occurrence

Value is only read, e.g. x,y,z in

x.sum = y.data + z.data

l-value occurrence

Value is modified e.g. y in

y = x.lptr

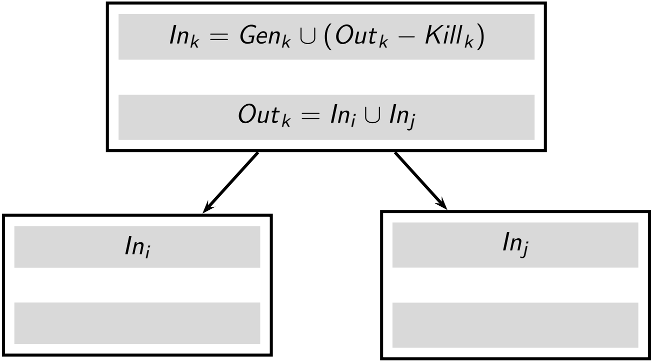
$$Gen_n = \{ v \mid \text{variable } v \text{ is used in basic block } n \text{ and is not preceded by a definition of } v \}$$

$$Kill_n = \{ v \mid \text{basic block } n \text{ contains a definition of } v \}$$

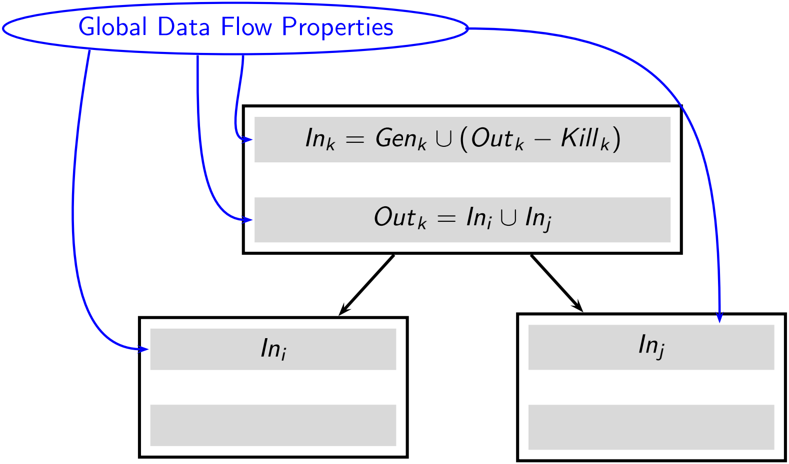
within n

anywhere in n

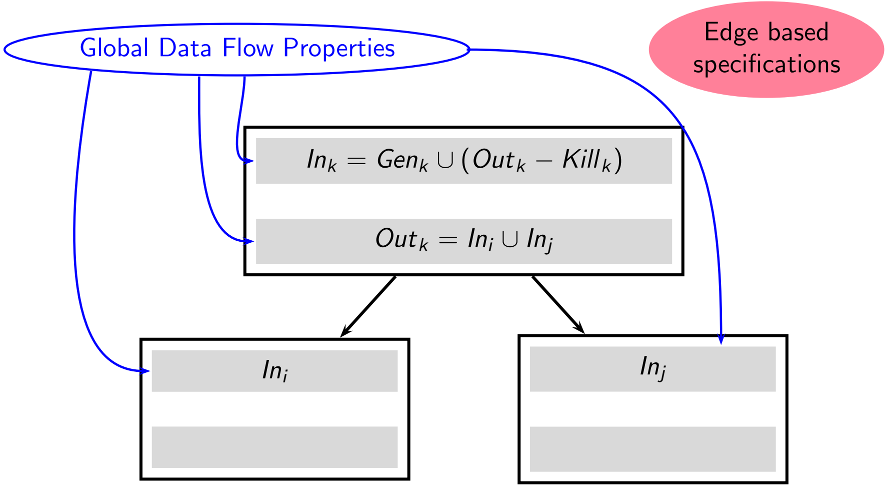
Defining Data Flow Analysis for Live Variables Analysis



Defining Data Flow Analysis for Live Variables Analysis



Defining Data Flow Analysis for Live Variables Analysis



Data Flow Equations For Live Variables Analysis

$$In_n = (Out_n - Kill_n) \cup Gen_n$$

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

Data Flow Equations For Live Variables Analysis

$$
\begin{aligned}
In_n &= (Out_n - Kill_n) \cup Gen_n \\
Out_n &= \begin{cases} BI & n \text{ is } End \text{ block} \\ \bigcup\limits_{s \in succ(n)} In_s & \text{otherwise} \end{cases}
\end{aligned}
$$

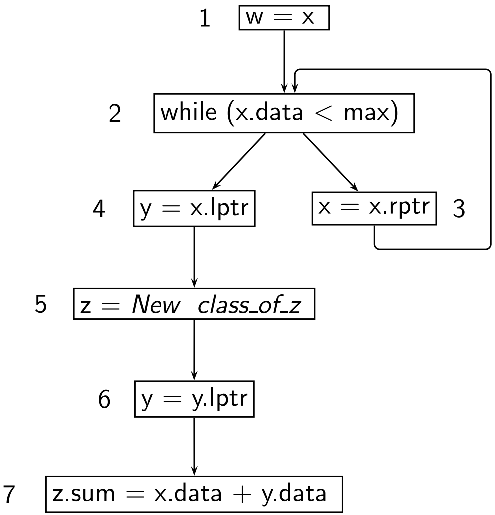
- In_n and Out_n are sets of variables

Data Flow Equations For Live Variables Analysis

$$
\begin{aligned}
In_n &= (Out_n - Kill_n) \cup Gen_n \\
Out_n &= \begin{cases} BI & n \text{ is } End \text{ block} \\ \bigcup\limits_{s \in succ(n)} In_s & \text{otherwise} \end{cases}
\end{aligned}
$$

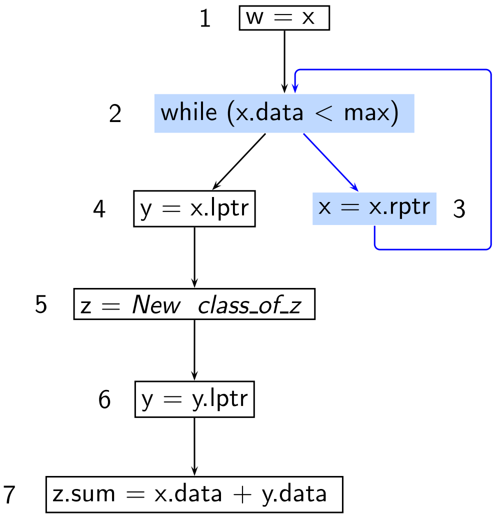
- In_n and Out_n are sets of variables
- BI is boundary information representing the effect of calling contexts
  - $\emptyset$ for local variables except for the values being returned
  - set of global variables used further in any calling context
    (can be safely approximated by the set of all global variables)

Data Flow Equations for Our Example



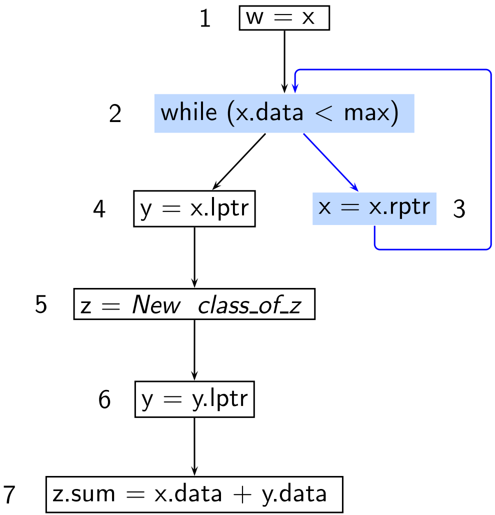
$$
\begin{aligned}
In_1 &= (Out_1 - Kill_1) \cup Gen_1 \\
Out_1 &= In_2 \\
In_2 &= (Out_2 - Kill_2) \cup Gen_2 \\
Out_2 &= In_3 \cup In_4 \\
In_3 &= (Out_3 - Kill_3) \cup Gen_3 \\
Out_3 &= In_2 \\
In_4 &= (Out_4 - Kill_4) \cup Gen_4 \\
Out_4 &= In_5 \\
In_5 &= (Out_5 - Kill_5) \cup Gen_5 \\
Out_5 &= In_6 \\
In_6 &= (Out_6 - Kill_6) \cup Gen_6 \\
Out_6 &= In_7 \\
In_7 &= (Out_7 - Kill_7) \cup Gen_7 \\
Out_7 &= \emptyset
\end{aligned}
$$

Data Flow Equations for Our Example



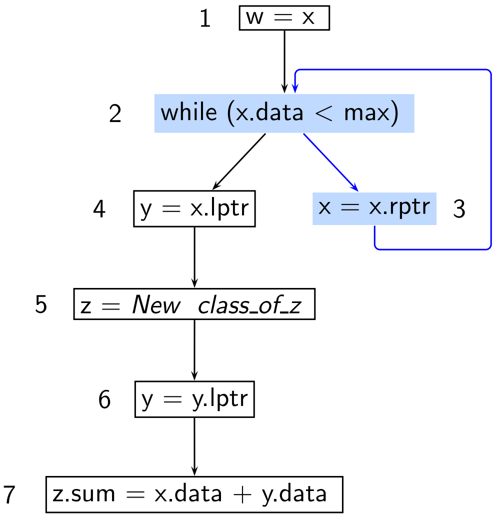
$$In_1 = (Out_1 - Kill_1) \cup Gen_1$$
$$Out_1 = In_2$$
$$In_2 = (Out_2 - Kill_2) \cup Gen_2$$
$$Out_2 = In_3 \cup In_4$$
$$In_3 = (Out_3 - Kill_3) \cup Gen_3$$
$$Out_3 = In_2$$
$$In_4 = (Out_4 - Kill_4) \cup Gen_4$$
$$Out_4 = In_5$$
$$In_5 = (Out_5 - Kill_5) \cup Gen_5$$
$$Out_5 = In_6$$
$$In_6 = (Out_6 - Kill_6) \cup Gen_6$$
$$Out_6 = In_7$$
$$In_7 = (Out_7 - Kill_7) \cup Gen_7$$
$$Out_7 = \emptyset$$

Data Flow Equations for Our Example



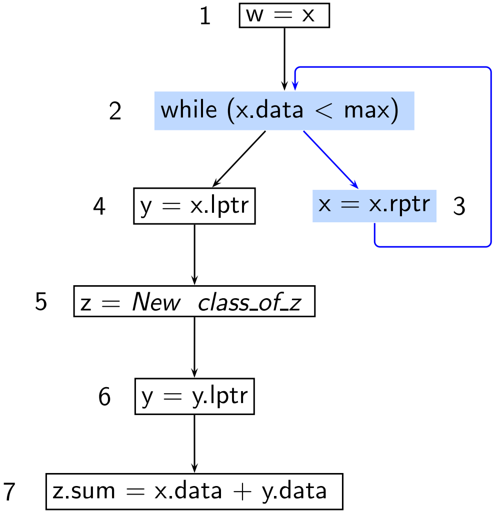
$$
\begin{aligned}
In_1 &= (Out_1 - Kill_1) \cup Gen_1 \\
Out_1 &= In_2 \\
In_2 &= (Out_2 - Kill_2) \cup Gen_2 \\
Out_2 &= In_3 \cup In_4 \\
In_3 &= (Out_3 - Kill_3) \cup Gen_3 \\
Out_3 &= In_2 \\
In_4 &= (Out_4 - Kill_4) \cup Gen_4 \\
Out_4 &= In_5 \\
In_5 &= (Out_5 - Kill_5) \cup Gen_5 \\
Out_5 &= In_6 \\
In_6 &= (Out_6 - Kill_6) \cup Gen_6 \\
Out_6 &= In_7 \\
In_7 &= (Out_7 - Kill_7) \cup Gen_7 \\
Out_7 &= \emptyset
\end{aligned}
$$

Data Flow Equations for Our Example



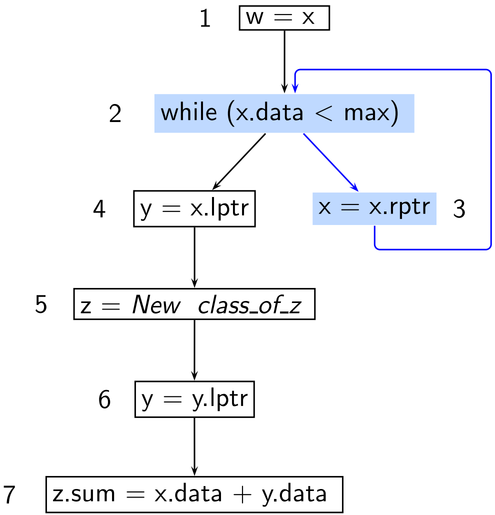
$$\begin{aligned}
In_1 &= (Out_1 - Kill_1) \cup Gen_1 \\
Out_1 &= In_2 \\
In_2 &= (Out_2 - Kill_2) \cup Gen_2 \\
Out_2 &= In_3 \cup In_4 \\
In_3 &= (Out_3 - Kill_3) \cup Gen_3 \\
Out_3 &= In_2 \\
In_4 &= (Out_4 - Kill_4) \cup Gen_4 \\
Out_4 &= In_5 \\
In_5 &= (Out_5 - Kill_5) \cup Gen_5 \\
Out_5 &= In_6 \\
In_6 &= (Out_6 - Kill_6) \cup Gen_6 \\
Out_6 &= In_7 \\
In_7 &= (Out_7 - Kill_7) \cup Gen_7 \\
Out_7 &= \emptyset
\end{aligned}$$

Data Flow Equations for Our Example



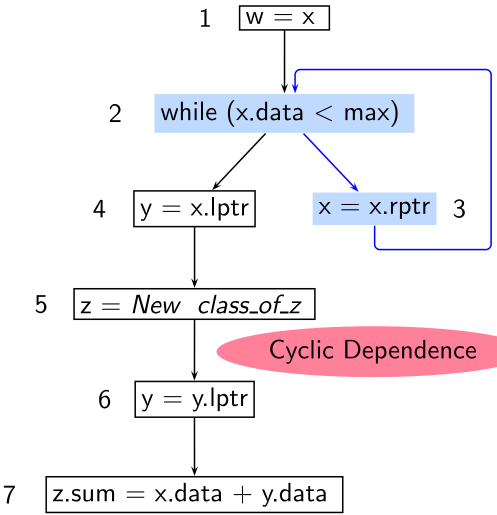
$$
\begin{aligned}
In_1 &= (Out_1 - Kill_1) \cup Gen_1 \\
Out_1 &= In_2 \\
In_2 &= (Out_2 - Kill_2) \cup Gen_2 \\
Out_2 &= In_3 \cup In_4 \\
In_3 &= (Out_3 - Kill_3) \cup Gen_3 \\
Out_3 &= In_2 \\
In_4 &= (Out_4 - Kill_4) \cup Gen_4 \\
Out_4 &= In_5 \\
In_5 &= (Out_5 - Kill_5) \cup Gen_5 \\
Out_5 &= In_6 \\
In_6 &= (Out_6 - Kill_6) \cup Gen_6 \\
Out_6 &= In_7 \\
In_7 &= (Out_7 - Kill_7) \cup Gen_7 \\
Out_7 &= \emptyset
\end{aligned}
$$

Data Flow Equations for Our Example



$$
\begin{aligned}
In_1 &= (Out_1 - Kill_1) \cup Gen_1 \\
Out_1 &= In_2 \\
In_2 &= (Out_2 - Kill_2) \cup Gen_2 \\
Out_2 &= In_3 \cup In_4 \\
In_3 &= (Out_3 - Kill_3) \cup Gen_3 \\
Out_3 &= In_2 \\
In_4 &= (Out_4 - Kill_4) \cup Gen_4 \\
Out_4 &= In_5 \\
In_5 &= (Out_5 - Kill_5) \cup Gen_5 \\
Out_5 &= In_6 \\
In_6 &= (Out_6 - Kill_6) \cup Gen_6 \\
Out_6 &= In_7 \\
In_7 &= (Out_7 - Kill_7) \cup Gen_7 \\
Out_7 &= \emptyset
\end{aligned}
$$

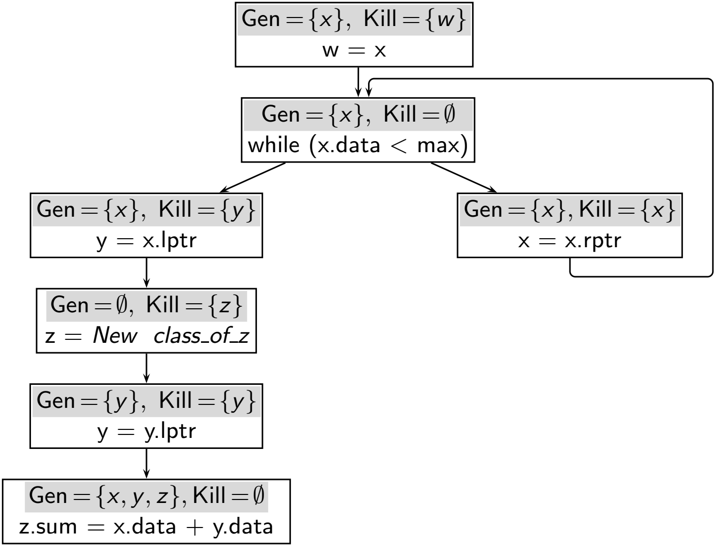
Data Flow Equations for Our Example



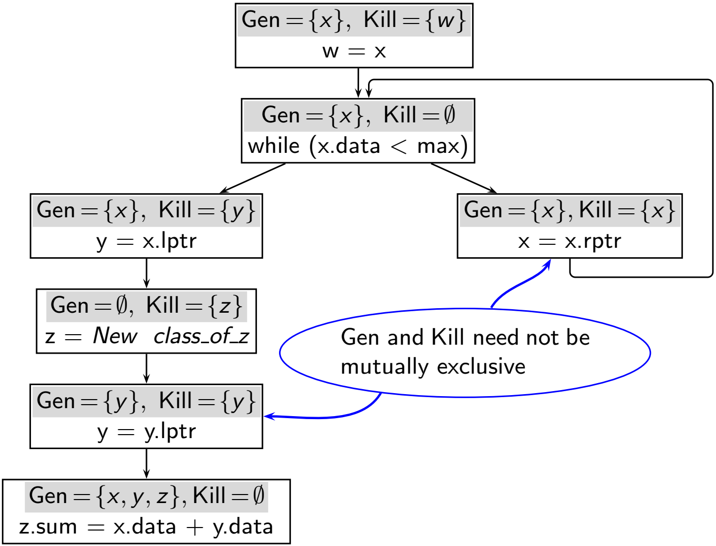
Cyclic Dependence

$$
\begin{aligned}
In_1 &= (Out_1 - Kill_1) \cup Gen_1 \\
Out_1 &= In_2 \\
In_2 &= (Out_2 - Kill_2) \cup Gen_2 \\
Out_2 &= In_3 \cup In_4 \\
In_3 &= (Out_3 - Kill_3) \cup Gen_3 \\
Out_3 &= In_2 \\
In_4 &= (Out_4 - Kill_4) \cup Gen_4 \\
Out_4 &= In_5 \\
In_5 &= (Out_5 - Kill_5) \cup Gen_5 \\
Out_5 &= In_6 \\
In_6 &= (Out_6 - Kill_6) \cup Gen_6 \\
Out_6 &= In_7 \\
In_7 &= (Out_7 - Kill_7) \cup Gen_7 \\
Out_7 &= \emptyset
\end{aligned}
$$

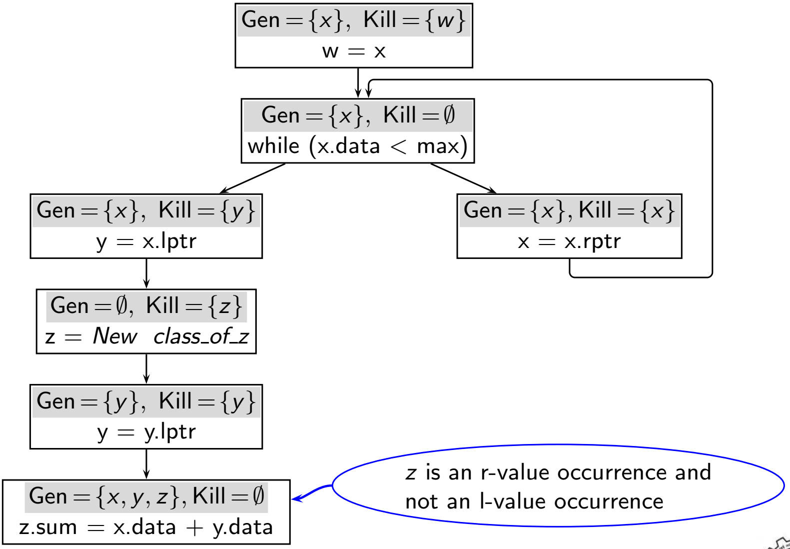
Performing Live Variables Analysis



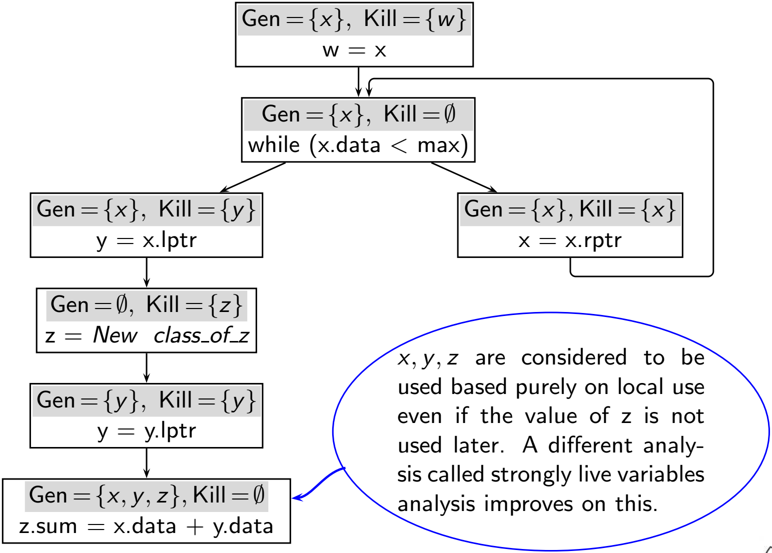
Performing Live Variables Analysis



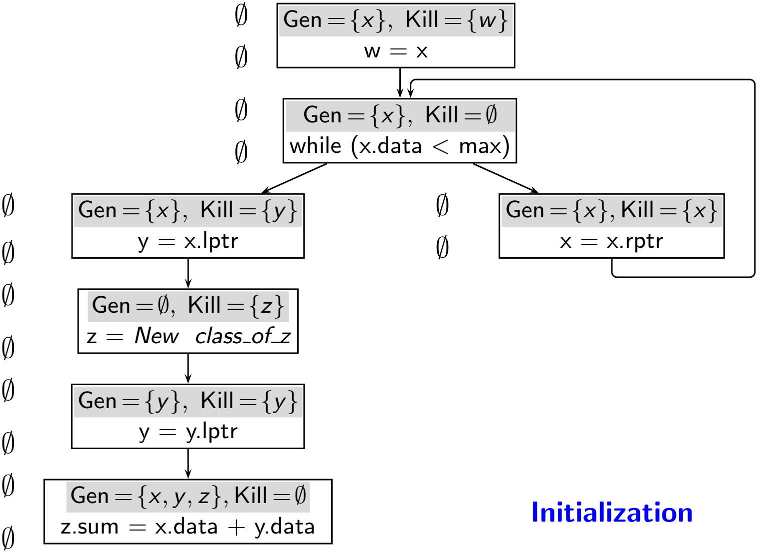
Performing Live Variables Analysis



Performing Live Variables Analysis

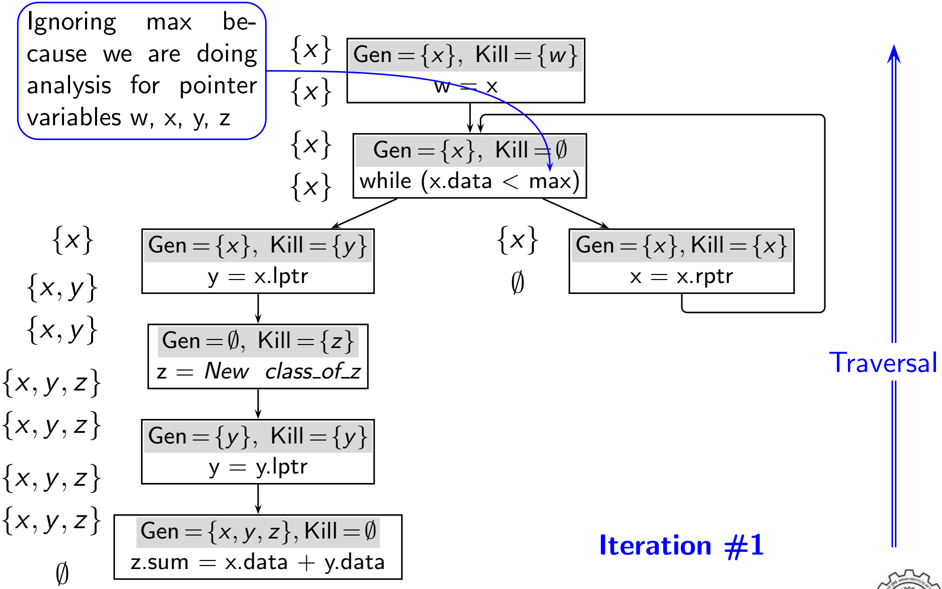


Performing Live Variables Analysis

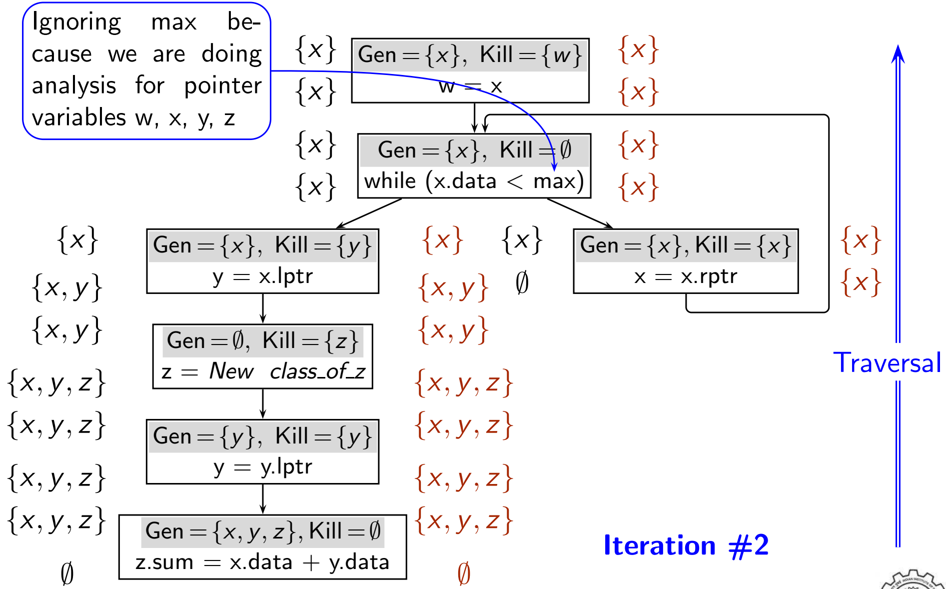


Initialization

Performing Live Variables Analysis

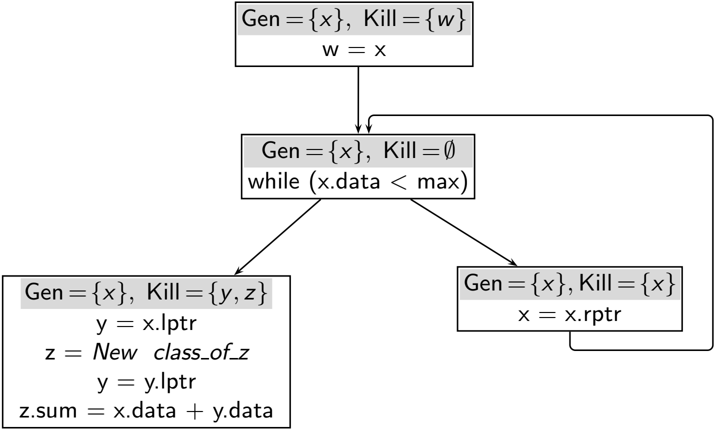


Performing Live Variables Analysis



Performing Live Variables Analysis

Local data flow properties when basic blocks contain multiple statements



Local Data Flow Properties for Live Variables Analysis

$$In_n = Gen_n \cup (Out_n - Kill_n)$$

- Gen_n : Use not preceded by definition
- $Kill_n$: Definition anywhere in a block

Local Data Flow Properties for Live Variables Analysis

$$In_n = Gen_n \cup (Out_n - Kill_n)$$

- Gen_n : Use not preceded by definition

  Upwards exposed use

- $Kill_n$: Definition anywhere in a block

  Stop the effect from being propagated across a block

Local Data Flow Properties for Live Variables Analysis

Case	Local Information		Example	Explanation
1	$v \notin Gen_n$	$v \notin Kill_n$		
2	$v \in Gen_n$	$v \notin Kill_n$		
3	$v \notin Gen_n$	$v \in Kill_n$		
4	$v \in Gen_n$	$v \in Kill_n$		

Local Data Flow Properties for Live Variables Analysis

Case	Local Information		Example	Explanation
1	$v \notin Gen_n$	$v \notin Kill_n$	$a = b + c$ $b = c * d$	liveness of v is unaffected by the basic block
2	$v \in Gen_n$	$v \notin Kill_n$	$a = b + c$ $b = v * d$	v becomes live before the basic block
3	$v \notin Gen_n$	$v \in Kill_n$	$a = b + c$ $v = c * d$	v ceases to be live before the basic block
4	$v \in Gen_n$	$v \in Kill_n$	$a = v + c$ $v = c * d$	liveness of v is killed but v becomes live before the basic block

Using Data Flow Information of Live Variables Analysis

- Used for register allocation

  If variable x is live in a basic block b, it is a potential candidate for register allocation

Using Data Flow Information of Live Variables Analysis

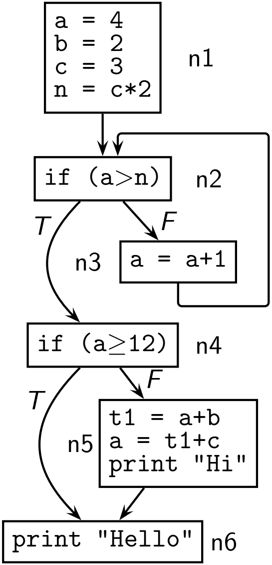
- Used for register allocation

  If variable x is live in a basic block b, it is a potential candidate for register allocation

- Used for dead code elimination

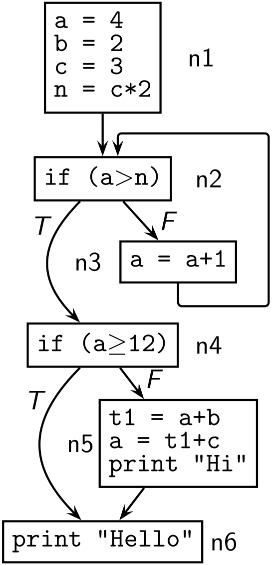
  If variable x is not live after an assignment $x = \ldots$, then the assignment is redundant and can be deleted as dead code

Tutorial Problem 1: Perform Dead Code Elimination



Local Data Flow Information		
	Gen	*Kill*
n1	$\emptyset$	$\{a, b, c, n\}$
n2	$\{a, n\}$	$\emptyset$
n3	$\{a\}$	$\{a\}$
n4	$\{a\}$	$\emptyset$
n5	$\{a, b, c\}$	$\{a, t1\}$
n6	$\emptyset$	$\emptyset$

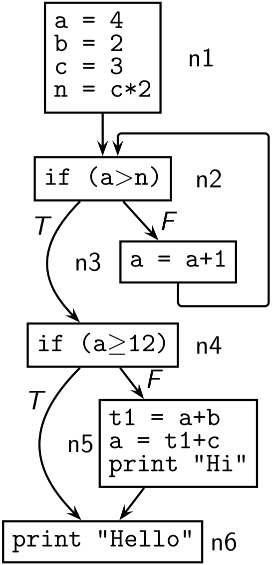
Tutorial Problem 1: Perform Dead Code Elimination



Local Data Flow Information		
	Gen	*Kill*
n1	$\emptyset$	$\{a, b, c, n\}$
n2	$\{a, n\}$	$\emptyset$
n3	$\{a\}$	$\{a\}$
n4	$\{a\}$	$\emptyset$
n5	$\{a, b, c\}$	$\{a, t1\}$
n6	$\emptyset$	$\emptyset$

Global Data Flow Information				
	Iteration #1		Iteration #2	
	Out	*In*	*Out*	*In*
n6	$\emptyset$	$\emptyset$		
n5	$\emptyset$	$\{a, b, c\}$		
n4	$\{a, b, c\}$	$\{a, b, c\}$		
n3	$\emptyset$	$\{a\}$		
n2	$\{a, b, c\}$	$\{a, b, c, n\}$		
n1	$\{a, b, c, n\}$	$\emptyset$		

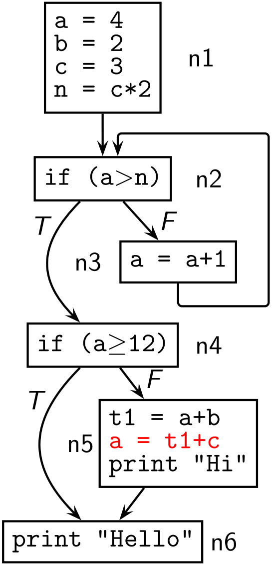
Tutorial Problem 1: Perform Dead Code Elimination



Local Data Flow Information		
	Gen	*Kill*
n1	$\emptyset$	$\{a, b, c, n\}$
n2	$\{a, n\}$	$\emptyset$
n3	$\{a\}$	$\{a\}$
n4	$\{a\}$	$\emptyset$
n5	$\{a, b, c\}$	$\{a, t1\}$
n6	$\emptyset$	$\emptyset$

Global Data Flow Information				
	Iteration #1		Iteration #2	
	Out	*In*	*Out*	*In*
n6	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
n5	$\emptyset$	$\{a, b, c\}$	$\emptyset$	$\{a, b, c\}$
n4	$\{a, b, c\}$	$\{a, b, c\}$	$\{a, b, c\}$	$\{a, b, c\}$
n3	$\emptyset$	$\{a\}$	$\{a, b, c, n\}$	$\{a, b, c, n\}$
n2	$\{a, b, c\}$	$\{a, b, c, n\}$	$\{a, b, c, n\}$	$\{a, b, c, n\}$
n1	$\{a, b, c, n\}$	$\emptyset$	$\{a, b, c, n\}$	$\emptyset$

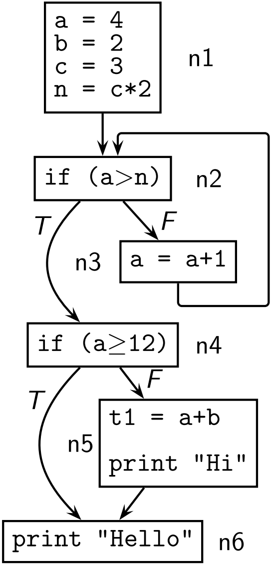
Tutorial Problem 1: Perform Dead Code Elimination



Local Data Flow Information		
	Gen	*Kill*
n1	$\emptyset$	$\{a, b, c, n\}$
n2	$\{a, n\}$	$\emptyset$
n3	$\{a\}$	$\{a\}$
n4	$\{a\}$	$\emptyset$
n5	$\{a, b, c\}$	$\{a, t1\}$
n6	$\emptyset$	$\emptyset$

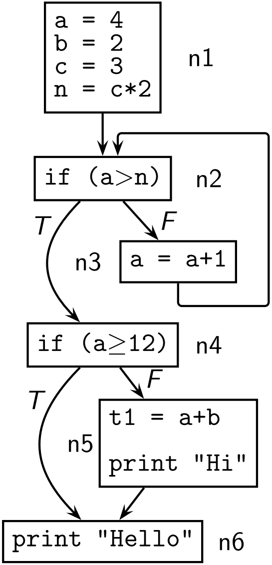
Global Data Flow Information				
	Iteration #1		Iteration #2	
	Out	*In*	*Out*	*In*
n6	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
n5	$\emptyset$	$\{a, b, c\}$	$\emptyset$	$\{a, b, c\}$
n4	$\{a, b, c\}$	$\{a, b, c\}$	$\{a, b, c\}$	$\{a, b, c\}$
n3	$\emptyset$	$\{a\}$	$\{a, b, c, n\}$	$\{a, b, c, n\}$
n2	$\{a, b, c\}$	$\{a, b, c, n\}$	$\{a, b, c, n\}$	$\{a, b, c, n\}$
n1	$\{a, b, c, n\}$	$\emptyset$	$\{a, b, c, n\}$	$\emptyset$

Tutorial Problem 1: Round #2 of Dead Code Elimination



Local Data Flow Information		
	Gen	*Kill*
n1	$\emptyset$	$\{a, b, c, n\}$
n2	$\{a, n\}$	$\emptyset$
n3	$\{a\}$	$\{a\}$
n4	$\{a\}$	$\emptyset$
n5	$\{a, b\}$	$\{t1\}$
n6	$\emptyset$	$\emptyset$

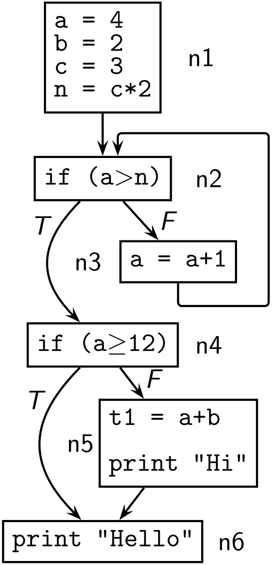
Tutorial Problem 1: Round #2 of Dead Code Elimination



Local Data Flow Information		
	Gen	*Kill*
n1	$\emptyset$	$\{a, b, c, n\}$
n2	$\{a, n\}$	$\emptyset$
n3	$\{a\}$	$\{a\}$
n4	$\{a\}$	$\emptyset$
n5	$\{a, b\}$	$\{t1\}$
n6	$\emptyset$	$\emptyset$

Global Data Flow Information				
	Iteration #1		Iteration #2	
	Out	*In*	*Out*	*In*
n6	$\emptyset$	$\emptyset$		
n5	$\emptyset$	$\{a, b\}$		
n4	$\{a, b\}$	$\{a, b\}$		
n3	$\emptyset$	$\{a\}$		
n2	$\{a, b\}$	$\{a, b, n\}$		
n1	$\{a, b, n\}$	$\emptyset$		

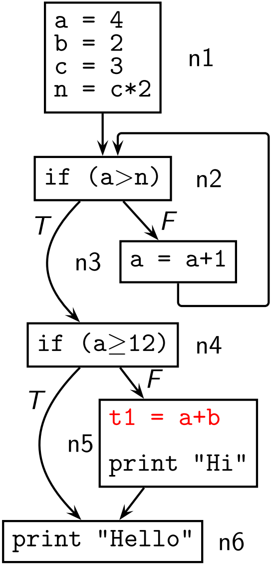
Tutorial Problem 1: Round #2 of Dead Code Elimination



Local Data Flow Information		
	Gen	*Kill*
n1	$\emptyset$	$\{a, b, c, n\}$
n2	$\{a, n\}$	$\emptyset$
n3	$\{a\}$	$\{a\}$
n4	$\{a\}$	$\emptyset$
n5	$\{a, b\}$	$\{t1\}$
n6	$\emptyset$	$\emptyset$

Global Data Flow Information				
	Iteration #1		Iteration #2	
	Out	*In*	*Out*	*In*
n6	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
n5	$\emptyset$	$\{a, b\}$	$\emptyset$	$\{a, b\}$
n4	$\{a, b\}$	$\{a, b\}$	$\{a, b\}$	$\{a, b\}$
n3	$\emptyset$	$\{a\}$	$\{a, b, n\}$	$\{a, b, n\}$
n2	$\{a, b\}$	$\{a, b, n\}$	$\{a, b, n\}$	$\{a, b, n\}$
n1	$\{a, b, n\}$	$\emptyset$	$\{a, b, n\}$	$\emptyset$

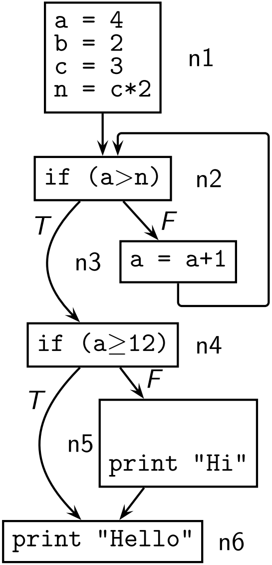
Tutorial Problem 1: Round #2 of Dead Code Elimination



Local Data Flow Information		
	Gen	*Kill*
n1	$\emptyset$	$\{a, b, c, n\}$
n2	$\{a, n\}$	$\emptyset$
n3	$\{a\}$	$\{a\}$
n4	$\{a\}$	$\emptyset$
n5	$\{a, b\}$	$\{t1\}$
n6	$\emptyset$	$\emptyset$

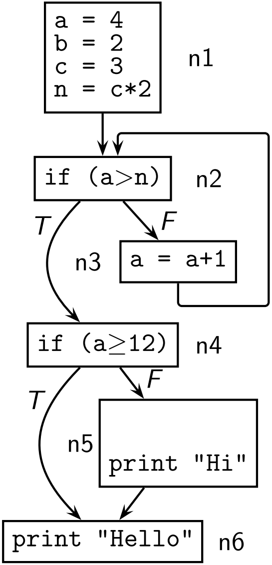
Global Data Flow Information				
	Iteration #1		Iteration #2	
	Out	*In*	*Out*	*In*
n6	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
n5	$\emptyset$	$\{a, b\}$	$\emptyset$	$\{a, b\}$
n4	$\{a, b\}$	$\{a, b\}$	$\{a, b\}$	$\{a, b\}$
n3	$\emptyset$	$\{a\}$	$\{a, b, n\}$	$\{a, b, n\}$
n2	$\{a, b\}$	$\{a, b, n\}$	$\{a, b, n\}$	$\{a, b, n\}$
n1	$\{a, b, n\}$	$\emptyset$	$\{a, b, n\}$	$\emptyset$

Tutorial Problem 1: Round #3 of Dead Code Elimination



Local Data Flow Information		
	Gen	*Kill*
n1	$\emptyset$	$\{a, b, c, n\}$
n2	$\{a, n\}$	$\emptyset$
n3	$\{a\}$	$\{a\}$
n4	$\{a\}$	$\emptyset$
n5	$\emptyset$	$\emptyset$
n6	$\emptyset$	$\emptyset$

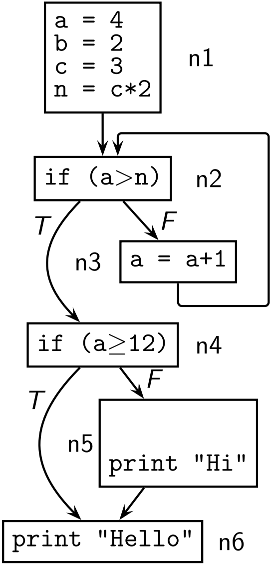
Tutorial Problem 1: Round #3 of Dead Code Elimination



Local Data Flow Information		
	Gen	*Kill*
n1	$\emptyset$	$\{a, b, c, n\}$
n2	$\{a, n\}$	$\emptyset$
n3	$\{a\}$	$\{a\}$
n4	$\{a\}$	$\emptyset$
n5	$\emptyset$	$\emptyset$
n6	$\emptyset$	$\emptyset$

Global Data Flow Information				
	Iteration #1		Iteration #2	
	Out	*In*	*Out*	*In*
n6	$\emptyset$	$\emptyset$		
n5	$\emptyset$	$\emptyset$		
n4	$\emptyset$	$\{a\}$		
n3	$\emptyset$	$\{a\}$		
n2	$\{a\}$	$\{a, n\}$		
n1	$\{a, n\}$	$\emptyset$		

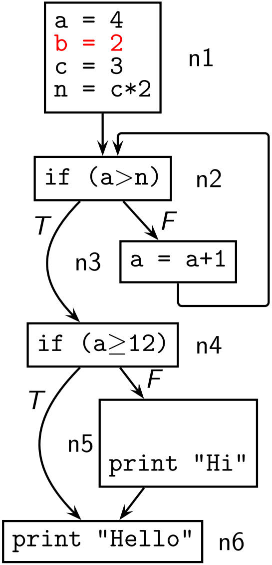
Tutorial Problem 1: Round #3 of Dead Code Elimination



Local Data Flow Information		
	Gen	*Kill*
n1	$\emptyset$	$\{a, b, c, n\}$
n2	$\{a, n\}$	$\emptyset$
n3	$\{a\}$	$\{a\}$
n4	$\{a\}$	$\emptyset$
n5	$\emptyset$	$\emptyset$
n6	$\emptyset$	$\emptyset$

Global Data Flow Information				
	Iteration #1		Iteration #2	
	Out	*In*	*Out*	*In*
n6	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
n5	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
n4	$\emptyset$	$\{a\}$	$\emptyset$	$\{a\}$
n3	$\emptyset$	$\{a\}$	$\{a, n\}$	$\{a, n\}$
n2	$\{a\}$	$\{a, n\}$	$\{a, n\}$	$\{a, n\}$
n1	$\{a, n\}$	$\emptyset$	$\{a, n\}$	$\emptyset$

Tutorial Problem 1: Round #3 of Dead Code Elimination



Local Data Flow Information		
	Gen	*Kill*
n1	$\emptyset$	$\{a, b, c, n\}$
n2	$\{a, n\}$	$\emptyset$
n3	$\{a\}$	$\{a\}$
n4	$\{a\}$	$\emptyset$
n5	$\emptyset$	$\emptyset$
n6	$\emptyset$	$\emptyset$

Global Data Flow Information				
	Iteration #1		Iteration #2	
	Out	*In*	*Out*	*In*
n6	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
n5	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
n4	$\emptyset$	$\{a\}$	$\emptyset$	$\{a\}$
n3	$\emptyset$	$\{a\}$	$\{a, n\}$	$\{a, n\}$
n2	$\{a\}$	$\{a, n\}$	$\{a, n\}$	$\{a, n\}$
n1	$\{a, n\}$	$\emptyset$	$\{a, n\}$	$\emptyset$

Part 3

Some Observations

What Does Data Flow Analysis Involve?

- Defining the analysis.
- Formulating the analysis.
- Performing the analysis.

What Does Data Flow Analysis Involve?

- Defining the analysis. Define the properties of execution paths
- Formulating the analysis.
- Performing the analysis.

What Does Data Flow Analysis Involve?

- Defining the analysis. Define the properties of execution paths
- Formulating the analysis. Define data flow equations
  - Linear simultaneous equations on sets rather than numbers
  - Later we will generalize the domain of values
- Performing the analysis.

What Does Data Flow Analysis Involve?

- Defining the analysis. Define the properties of execution paths
- Formulating the analysis. Define data flow equations
  - Linear simultaneous equations on sets rather than numbers
  - Later we will generalize the domain of values
- Performing the analysis. Solve data flow equations for the given program flow graph

What Does Data Flow Analysis Involve?

- Defining the analysis. Define the properties of execution paths
- Formulating the analysis. Define data flow equations
  - Linear simultaneous equations on sets rather than numbers
  - Later we will generalize the domain of values
- Performing the analysis. Solve data flow equations for the given program flow graph
- Many unanswered questions

  Initial value? Termination? Complexity? Properties of Solutions?

A Digression: Iterative Solution of Linear Simultaneous Equations

- Simultaneous equations represented in the form of the product of a matrix of coefficients (**A**) with the vector of unknowns (**x**)

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

- Start with approximate values
- Compute new values repeatedly from old values
- Two classical methods
    - Gauss-Seidel Method (Gauss: 1823, 1826), (Seidel: 1874)
    - Jacobi Method (Jacobi: 1845)

A Digression: An Example of Iterative Solution of Linear Simultaneous Equations

Equations	Solution
$4w = x + y + 32$ $4x = y + z + 32$ $4y = z + w + 32$ $4z = w + x + 32$	$w = x = y = z = 16$

- Rewrite the equations to define w, x, y, and z

$$\begin{aligned} w &= 0.25x + 0.25y + 8 \\ x &= 0.25y + 0.25z + 8 \\ y &= 0.25z + 0.25w + 8 \\ z &= 0.25w + 0.25x + 8 \end{aligned}$$

- Assume some initial values of w_0, x_0, y_0, and z_0
- Compute w_i, x_i, y_i, and z_i within some margin of error

A Digression: Gauss-Seidel Method

Equations	Initial Values	Error Margin
$w = 0.25x + 0.25y + 8$	$w_0 = 24$	$w_{i+1} - w_i \leq 0.35$
$x = 0.25y + 0.25z + 8$	$x_0 = 24$	$x_{i+1} - x_i \leq 0.35$
$y = 0.25z + 0.25w + 8$	$y_0 = 24$	$y_{i+1} - y_i \leq 0.35$
$z = 0.25w + 0.25x + 8$	$z_0 = 24$	$z_{i+1} - z_i \leq 0.35$

Iteration 1	Iteration 2	Iteration 3

Iteration 4	Iteration 5

A Digression: Gauss-Seidel Method

Equations	Initial Values	Error Margin
$w = 0.25x + 0.25y + 8$	$w_0 = 24$	$w_{i+1} - w_i \leq 0.35$
$x = 0.25y + 0.25z + 8$	$x_0 = 24$	$x_{i+1} - x_i \leq 0.35$
$y = 0.25z + 0.25w + 8$	$y_0 = 24$	$y_{i+1} - y_i \leq 0.35$
$z = 0.25w + 0.25x + 8$	$z_0 = 24$	$z_{i+1} - z_i \leq 0.35$

Iteration 1	Iteration 2	Iteration 3
$w_1 = 6 + 6 + 8 = 20$ $x_1 = 6 + 6 + 8 = 20$ $y_1 = 6 + 6 + 8 = 20$ $z_1 = 6 + 6 + 8 = 20$		

Iteration 4	Iteration 5

A Digression: Gauss-Seidel Method

Equations	Initial Values	Error Margin
$w = 0.25x + 0.25y + 8$	$w_0 = 24$	$w_{i+1} - w_i \leq 0.35$
$x = 0.25y + 0.25z + 8$	$x_0 = 24$	$x_{i+1} - x_i \leq 0.35$
$y = 0.25z + 0.25w + 8$	$y_0 = 24$	$y_{i+1} - y_i \leq 0.35$
$z = 0.25w + 0.25x + 8$	$z_0 = 24$	$z_{i+1} - z_i \leq 0.35$

Iteration 1	Iteration 2	Iteration 3
$w_1 = 6 + 6 + 8 = 20$	$w_2 = 5 + 5 + 8 = 18$	
$x_1 = 6 + 6 + 8 = 20$	$x_2 = 5 + 5 + 8 = 18$	
$y_1 = 6 + 6 + 8 = 20$	$y_2 = 5 + 5 + 8 = 18$	
$z_1 = 6 + 6 + 8 = 20$	$z_2 = 5 + 5 + 8 = 18$	

Iteration 4	Iteration 5

A Digression: Gauss-Seidel Method

Equations	Initial Values	Error Margin
$w = 0.25x + 0.25y + 8$	$w_0 = 24$	$w_{i+1} - w_i \leq 0.35$
$x = 0.25y + 0.25z + 8$	$x_0 = 24$	$x_{i+1} - x_i \leq 0.35$
$y = 0.25z + 0.25w + 8$	$y_0 = 24$	$y_{i+1} - y_i \leq 0.35$
$z = 0.25w + 0.25x + 8$	$z_0 = 24$	$z_{i+1} - z_i \leq 0.35$

Iteration 1	Iteration 2	Iteration 3
$w_1 = 6 + 6 + 8 = 20$	$w_2 = 5 + 5 + 8 = 18$	$w_3 = 4.5 + 4.5 + 8 = 17$
$x_1 = 6 + 6 + 8 = 20$	$x_2 = 5 + 5 + 8 = 18$	$x_3 = 4.5 + 4.5 + 8 = 17$
$y_1 = 6 + 6 + 8 = 20$	$y_2 = 5 + 5 + 8 = 18$	$y_3 = 4.5 + 4.5 + 8 = 17$
$z_1 = 6 + 6 + 8 = 20$	$z_2 = 5 + 5 + 8 = 18$	$z_3 = 4.5 + 4.5 + 8 = 17$

Iteration 4	Iteration 5

A Digression: Gauss-Seidel Method

Equations	Initial Values	Error Margin
$w = 0.25x + 0.25y + 8$	$w_0 = 24$	$w_{i+1} - w_i \leq 0.35$
$x = 0.25y + 0.25z + 8$	$x_0 = 24$	$x_{i+1} - x_i \leq 0.35$
$y = 0.25z + 0.25w + 8$	$y_0 = 24$	$y_{i+1} - y_i \leq 0.35$
$z = 0.25w + 0.25x + 8$	$z_0 = 24$	$z_{i+1} - z_i \leq 0.35$

Iteration 1	Iteration 2	Iteration 3
$w_1 = 6 + 6 + 8 = 20$	$w_2 = 5 + 5 + 8 = 18$	$w_3 = 4.5 + 4.5 + 8 = 17$
$x_1 = 6 + 6 + 8 = 20$	$x_2 = 5 + 5 + 8 = 18$	$x_3 = 4.5 + 4.5 + 8 = 17$
$y_1 = 6 + 6 + 8 = 20$	$y_2 = 5 + 5 + 8 = 18$	$y_3 = 4.5 + 4.5 + 8 = 17$
$z_1 = 6 + 6 + 8 = 20$	$z_2 = 5 + 5 + 8 = 18$	$z_3 = 4.5 + 4.5 + 8 = 17$

Iteration 4	Iteration 5
$w_4 = 4.25 + 4.25 + 8 = 16.5$	
$x_4 = 4.25 + 4.25 + 8 = 16.5$	
$y_4 = 4.25 + 4.25 + 8 = 16.5$	
$z_4 = 4.25 + 4.25 + 8 = 16.5$	

A Digression: Gauss-Seidel Method

Equations	Initial Values	Error Margin
$w = 0.25x + 0.25y + 8$	$w_0 = 24$	$w_{i+1} - w_i \leq 0.35$
$x = 0.25y + 0.25z + 8$	$x_0 = 24$	$x_{i+1} - x_i \leq 0.35$
$y = 0.25z + 0.25w + 8$	$y_0 = 24$	$y_{i+1} - y_i \leq 0.35$
$z = 0.25w + 0.25x + 8$	$z_0 = 24$	$z_{i+1} - z_i \leq 0.35$

Iteration 1	Iteration 2	Iteration 3
$w_1 = 6 + 6 + 8 = 20$	$w_2 = 5 + 5 + 8 = 18$	$w_3 = 4.5 + 4.5 + 8 = 17$
$x_1 = 6 + 6 + 8 = 20$	$x_2 = 5 + 5 + 8 = 18$	$x_3 = 4.5 + 4.5 + 8 = 17$
$y_1 = 6 + 6 + 8 = 20$	$y_2 = 5 + 5 + 8 = 18$	$y_3 = 4.5 + 4.5 + 8 = 17$
$z_1 = 6 + 6 + 8 = 20$	$z_2 = 5 + 5 + 8 = 18$	$z_3 = 4.5 + 4.5 + 8 = 17$

Iteration 4	Iteration 5
$w_4 = 4.25 + 4.25 + 8 = 16.5$	$w_5 = 4.125 + 4.125 + 8 = 16.25$
$x_4 = 4.25 + 4.25 + 8 = 16.5$	$x_5 = 4.125 + 4.125 + 8 = 16.25$
$y_4 = 4.25 + 4.25 + 8 = 16.5$	$y_5 = 4.125 + 4.125 + 8 = 16.25$
$z_4 = 4.25 + 4.25 + 8 = 16.5$	$z_5 = 4.125 + 4.125 + 8 = 16.25$

A Digression: Jacobi Method

Use values from the current iteration wherever possible

Equations	Initial Values	Error Margin
$w = 0.25x + 0.25y + 8$	$w_0 = 24$	$w_{i+1} - w_i \leq 0.35$
$x = 0.25y + 0.25z + 8$	$x_0 = 24$	$x_{i+1} - x_i \leq 0.35$
$y = 0.25z + 0.25w + 8$	$y_0 = 24$	$y_{i+1} - y_i \leq 0.35$
$z = 0.25w + 0.25x + 8$	$z_0 = 24$	$z_{i+1} - z_i \leq 0.35$

Iteration 1	Iteration 2

Iteration 3	Iteration 4

A Digression: Jacobi Method

Use values from the current iteration wherever possible

Equations	Initial Values	Error Margin
$w = 0.25x + 0.25y + 8$	$w_0 = 24$	$w_{i+1} - w_i \leq 0.35$
$x = 0.25y + 0.25z + 8$	$x_0 = 24$	$x_{i+1} - x_i \leq 0.35$
$y = 0.25z + 0.25w + 8$	$y_0 = 24$	$y_{i+1} - y_i \leq 0.35$
$z = 0.25w + 0.25x + 8$	$z_0 = 24$	$z_{i+1} - z_i \leq 0.35$

Iteration 1	Iteration 2
$w_1 = 6 + 6 + 8 = 20$ $x_1 = 6 + 6 + 8 = 20$ $y_1 = 6 + 5 + 8 = 19$ $z_1 = 5 + 5 + 8 = 18$	

Iteration 3	Iteration 4

A Digression: Jacobi Method

Use values from the current iteration wherever possible

Equations	Initial Values	Error Margin
$w = 0.25x + 0.25y + 8$	$w_0 = 24$	$w_{i+1} - w_i \leq 0.35$
$x = 0.25y + 0.25z + 8$	$x_0 = 24$	$x_{i+1} - x_i \leq 0.35$
$y = 0.25z + 0.25w + 8$	$y_0 = 24$	$y_{i+1} - y_i \leq 0.35$
$z = 0.25w + 0.25x + 8$	$z_0 = 24$	$z_{i+1} - z_i \leq 0.35$

Iteration 1	Iteration 2
$w_1 = 6 + 6 + 8 = 20$	$w_2 = 5 + 4.75 + 8 = 17.75$
$x_1 = 6 + 6 + 8 = 20$	$x_2 = 4.75 + 4.5 + 8 = 17.25$
$y_1 = 6 + 5 + 8 = 19$	$y_2 = 4.5 + 4.4375 + 8 = 16.935$
$z_1 = 5 + 5 + 8 = 18$	$z_2 = 4.4375 + 4.375 + 8 = 16.8125$

Iteration 3	Iteration 4

A Digression: Jacobi Method

Use values from the current iteration wherever possible

Equations	Initial Values	Error Margin
$w = 0.25x + 0.25y + 8$	$w_0 = 24$	$w_{i+1} - w_i \leq 0.35$
$x = 0.25y + 0.25z + 8$	$x_0 = 24$	$x_{i+1} - x_i \leq 0.35$
$y = 0.25z + 0.25w + 8$	$y_0 = 24$	$y_{i+1} - y_i \leq 0.35$
$z = 0.25w + 0.25x + 8$	$z_0 = 24$	$z_{i+1} - z_i \leq 0.35$

Iteration 1	Iteration 2
$w_1 = 6 + 6 + 8 = 20$	$w_2 = 5 + 4.75 + 8 = 17.75$
$x_1 = 6 + 6 + 8 = 20$	$x_2 = 4.75 + 4.5 + 8 = 17.25$
$y_1 = 6 + 5 + 8 = 19$	$y_2 = 4.5 + 4.4375 + 8 = 16.935$
$z_1 = 5 + 5 + 8 = 18$	$z_2 = 4.4375 + 4.375 + 8 = 16.8125$

Iteration 3	Iteration 4
$w_3 = 4.3125 + 4.23375 + 8 = 16.54625$	
$x_3 = 4.23375 + 4.23375 + 8 = 16.436875$	
$y_3 = 4.23375 + 4.1365625 + 8 = 16.370$	
$z_3 = 4.1365625 + 4.11 + 8 = 16.34375$	

A Digression: Jacobi Method

Use values from the current iteration wherever possible

Equations	Initial Values	Error Margin
$w = 0.25x + 0.25y + 8$	$w_0 = 24$	$w_{i+1} - w_i \leq 0.35$
$x = 0.25y + 0.25z + 8$	$x_0 = 24$	$x_{i+1} - x_i \leq 0.35$
$y = 0.25z + 0.25w + 8$	$y_0 = 24$	$y_{i+1} - y_i \leq 0.35$
$z = 0.25w + 0.25x + 8$	$z_0 = 24$	$z_{i+1} - z_i \leq 0.35$

Iteration 1	Iteration 2
$w_1 = 6 + 6 + 8 = 20$	$w_2 = 5 + 4.75 + 8 = 17.75$
$x_1 = 6 + 6 + 8 = 20$	$x_2 = 4.75 + 4.5 + 8 = 17.25$
$y_1 = 6 + 5 + 8 = 19$	$y_2 = 4.5 + 4.4375 + 8 = 16.935$
$z_1 = 5 + 5 + 8 = 18$	$z_2 = 4.4375 + 4.375 + 8 = 16.8125$

Iteration 3	Iteration 4
$w_3 = 4.3125 + 4.23375 + 8 = 16.54625$	$w_4 = 16.20172$
$x_3 = 4.23375 + 4.23375 + 8 = 16.436875$	$x_4 = 16.17844$
$y_3 = 4.23375 + 4.1365625 + 8 = 16.370$	$y_4 = 16.13637$
$z_3 = 4.1365625 + 4.11 + 8 = 16.34375$	$z_4 = 16.09504$

Our Method of Performing Data Flow Analysis

- Round robin iteration
- Essentially Jacobi method
- Unknowns are the data flow variables In_i and Out_i
- Domain of values is not numbers
- Computation in a fixed order
  - either forward (reverse post order) traversal, or
  - backward (post order) traversal

  over the control flow graph

Tutorial Problem 2 for Liveness Analysis

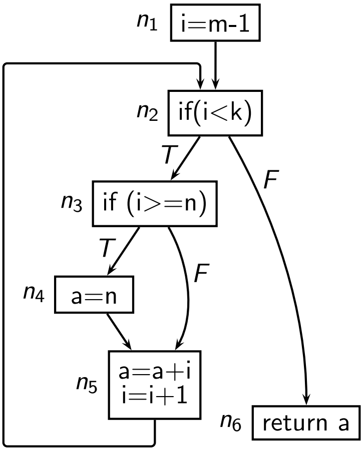
Draw the control flow graph and perform live variables analysis

```
int f(int m, int n, int k)
{
  int a,i;

  for (i=m-1; i<k; i++)
  {   if (i>=n)
          a = n;
      a = a+i;
  }
  return a;
}
```

Tutorial Problem 2 for Liveness Analysis

Draw the control flow graph and perform live variables analysis

```
int f(int m, int n, int k)
{
  int a,i;

  for (i=m-1; i<k; i++)
  {   if (i>=n)
          a = n;
      a = a+i;
  }
  return a;
}
```



The Semantics of Return Statement for Live Variables Analysis

"return a" is modelled by the statement "return_value_in_stack = a"

- If we assume that the statement is executed *within* the block

- If we assume that the statement is executed *outside of* the block and along the edge connecting the procedure to its caller

The Semantics of Return Statement for Live Variables Analysis

"return a" is modelled by the statement "return_value_in_stack = a"

- If we assume that the statement is executed *within* the block
  $\Longrightarrow$ *BI* can be $\emptyset$

- If we assume that the statement is executed *outside of* the block and along the edge connecting the procedure to its caller
  $\Longrightarrow a \in BI$

Solution of Tutorial Problem 2

Block	Local Information		Global Information			
			Iteration # 1		Change in iteration # 2	
	Gen_n	$Kill_n$	Out_n	In_n	Out_n	In_n
n_6	$\{a\}$	$\emptyset$				
n_5	$\{a, i\}$	$\{a, i\}$				
n_4	$\{n\}$	$\{a\}$				
n_3	$\{i, n\}$	$\emptyset$				
n_2	$\{i, k\}$	$\emptyset$				
n_1	$\{m\}$	$\{i\}$				

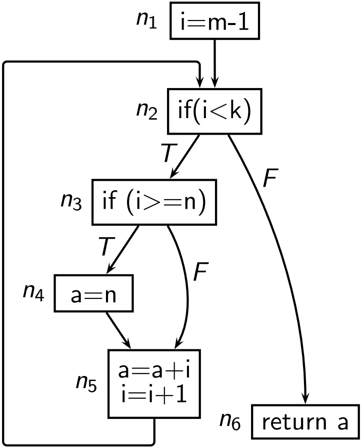
Solution of Tutorial Problem 2

Block	Local Information		Global Information			
			Iteration # 1		Change in iteration # 2	
	Gen_n	$Kill_n$	Out_n	In_n	Out_n	In_n
n_6	$\{a\}$	$\emptyset$	$\emptyset$	$\{a\}$		
n_5	$\{a, i\}$	$\{a, i\}$	$\emptyset$	$\{a, i\}$		
n_4	$\{n\}$	$\{a\}$	$\{a, i\}$	$\{i, n\}$		
n_3	$\{i, n\}$	$\emptyset$	$\{a, i, n\}$	$\{a, i, n\}$		
n_2	$\{i, k\}$	$\emptyset$	$\{a, i, n\}$	$\{a, i, k, n\}$		
n_1	$\{m\}$	$\{i\}$	$\{a, i, k, n\}$	$\{a, k, m, n\}$		

Solution of Tutorial Problem 2

Block	Local Information		Global Information			
			Iteration # 1		Change in iteration # 2	
	Gen_n	$Kill_n$	Out_n	In_n	Out_n	In_n
n_6	$\{a\}$	$\emptyset$	$\emptyset$	$\{a\}$		
n_5	$\{a, i\}$	$\{a, i\}$	$\emptyset$	$\{a, i\}$	$\{a, i, k, n\}$	$\{a, i, k, n\}$
n_4	$\{n\}$	$\{a\}$	$\{a, i\}$	$\{i, n\}$	$\{a, i, k, n\}$	$\{i, k, n\}$
n_3	$\{i, n\}$	$\emptyset$	$\{a, i, n\}$	$\{a, i, n\}$	$\{a, i, k, n\}$	$\{a, i, k, n\}$
n_2	$\{i, k\}$	$\emptyset$	$\{a, i, n\}$	$\{a, i, k, n\}$	$\{a, i, k, n\}$	
n_1	$\{m\}$	$\{i\}$	$\{a, i, k, n\}$	$\{a, k, m, n\}$		

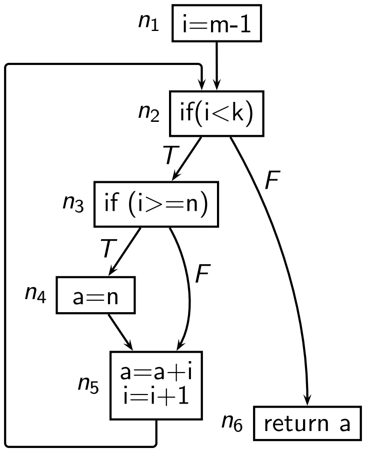
Interpreting the Result of Liveness Analysis for Tutorial Problem 2



- Is a live at the exit of n_5 at the end of iteration 1? Why?

  (We have used post order traversal)

Interpreting the Result of Liveness Analysis for Tutorial Problem 2



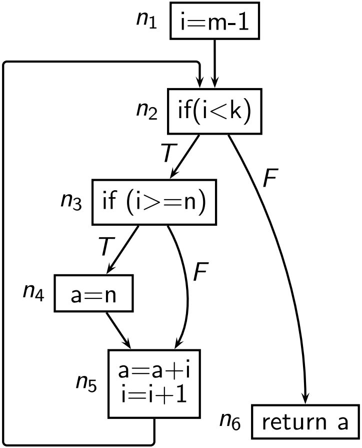
- Is a live at the exit of n_5 at the end of iteration 1? Why?

  (We have used post order traversal)

- Is a live at the exit of n_5 at the end of iteration 2? Why?

  (We have used post order traversal)

Interpreting the Result of Liveness Analysis for Tutorial Problem 2



- Is a live at the exit of n_5 at the end of iteration 1? Why?

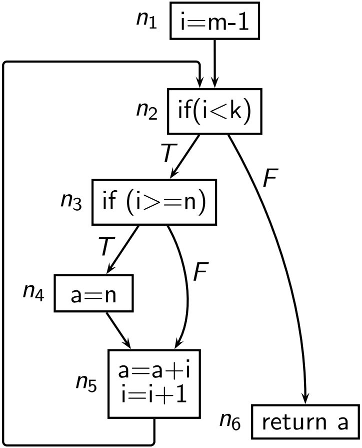
  (We have used post order traversal)

- Is a live at the exit of n_5 at the end of iteration 2? Why?

  (We have used post order traversal)

- Show an execution path along which a is live at the exit of n_5

Interpreting the Result of Liveness Analysis for Tutorial Problem 2



- Is a live at the exit of n_5 at the end of iteration 1? Why?

  (We have used post order traversal)

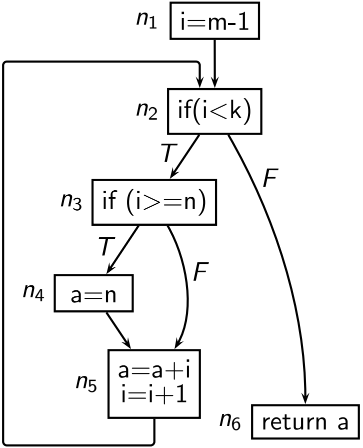
- Is a live at the exit of n_5 at the end of iteration 2? Why?

  (We have used post order traversal)

- Show an execution path along which a is live at the exit of n_5

- Show an execution path along which a is live at the exit of n_3

Interpreting the Result of Liveness Analysis for Tutorial Problem 2



- Is a live at the exit of n_5 at the end of iteration 1? Why?

  (We have used post order traversal)

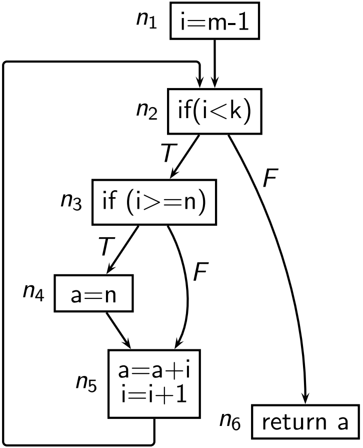
- Is a live at the exit of n_5 at the end of iteration 2? Why?

  (We have used post order traversal)

- Show an execution path along which a is live at the exit of n_5
- Show an execution path along which a is live at the exit of n_3

  $n_1 \rightarrow n_2 \rightarrow n_3 \rightarrow n_5 \rightarrow n_2 \rightarrow \ldots$

Interpreting the Result of Liveness Analysis for Tutorial Problem 2



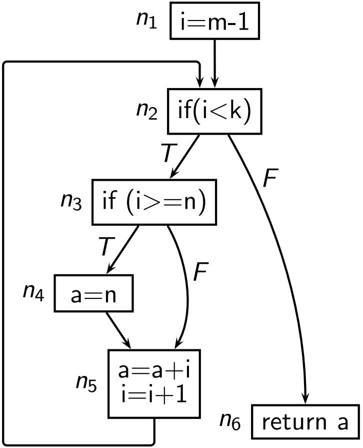
- Is a live at the exit of n_5 at the end of iteration 1? Why?

  (We have used post order traversal)
- Is a live at the exit of n_5 at the end of iteration 2? Why?

  (We have used post order traversal)
- Show an execution path along which a is live at the exit of n_5
- Show an execution path along which a is live at the exit of n_3

  $n_1 \rightarrow n_2 \rightarrow n_3 \rightarrow n_5 \rightarrow n_2 \rightarrow \ldots$
- Show an execution path along which a is not live at the exit of n_3

Interpreting the Result of Liveness Analysis for Tutorial Problem 2



- Is a live at the exit of n_5 at the end of iteration 1? Why?

  (We have used post order traversal)

- Is a live at the exit of n_5 at the end of iteration 2? Why?

  (We have used post order traversal)

- Show an execution path along which a is live at the exit of n_5
- Show an execution path along which a is live at the exit of n_3

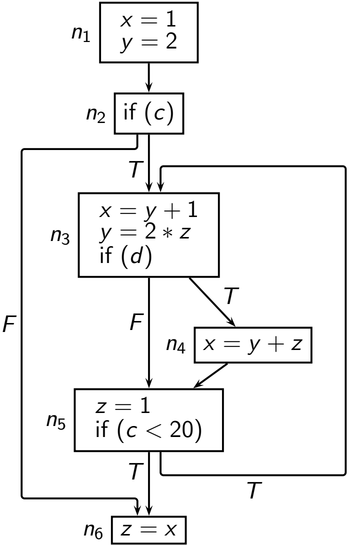
  $n_1 \rightarrow n_2 \rightarrow n_3 \rightarrow n_5 \rightarrow n_2 \rightarrow \ldots$

- Show an execution path along which a is not live at the exit of n_3

  $n_1 \rightarrow n_2 \rightarrow n_3 \rightarrow n_4 \rightarrow n_2 \rightarrow \ldots$

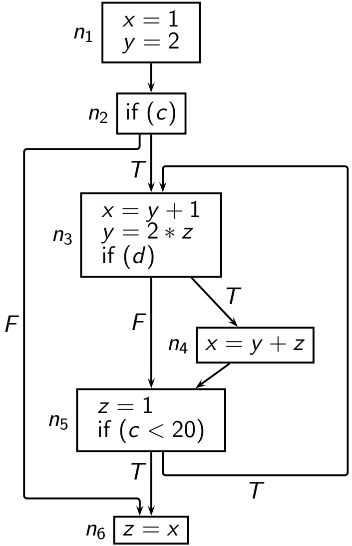
Tutorial Problem 3 for Liveness Analysis

Also write a C program for this CFG without using goto or break



Tutorial Problem 3 for Liveness Analysis

Also write a C program for this CFG without using goto or break




```
void f()
{  int x, y, z;
   int c, d;
   x = 1;
   y = 2;
   if (c)
   {   do
       {   x = y+1;
           y = 2*z;
           if (d)
               x = y+z;
           z = 1;
       } while (c < 20);
   }
   z = x;
}
```

Solution of Tutorial Problem 3

Block	Local Information		Global Information			
			Iteration # 1		Change in iteration # 2	
	Gen_n	$Kill_n$	Out_n	In_n	Out_n	In_n
n_6	$\{x\}$	$\{z\}$				
n_5	$\{c\}$	$\{z\}$				
n_4	$\{y, z\}$	$\{x\}$				
n_3	$\{y, z, d\}$	$\{x, y\}$				
n_2	$\{c\}$	$\emptyset$				
n_1	$\emptyset$	$\{x, y\}$				

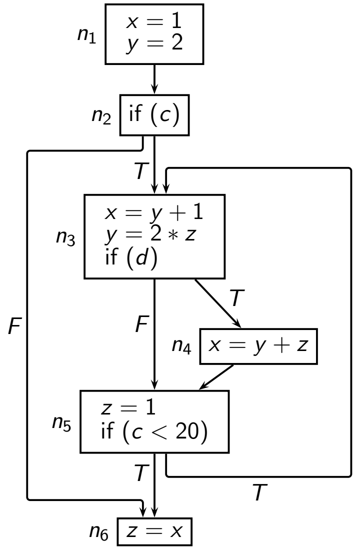
Solution of Tutorial Problem 3

Block	Local Information		Global Information			
			Iteration # 1		Change in iteration # 2	
	Gen_n	$Kill_n$	Out_n	In_n	Out_n	In_n
n_6	$\{x\}$	$\{z\}$	$\emptyset$	$\{x\}$		
n_5	$\{c\}$	$\{z\}$	$\{x\}$	$\{x, c\}$		
n_4	$\{y, z\}$	$\{x\}$	$\{x, c\}$	$\{y, z, c\}$		
n_3	$\{y, z, d\}$	$\{x, y\}$	$\{x, y, z, c\}$	$\{y, z, c, d\}$		
n_2	$\{c\}$	$\emptyset$	$\{x, y, z, c, d\}$	$\{x, y, z, c, d\}$		
n_1	$\emptyset$	$\{x, y\}$	$\{x, y, z, c, d\}$	$\{z, c, d\}$		

Solution of Tutorial Problem 3

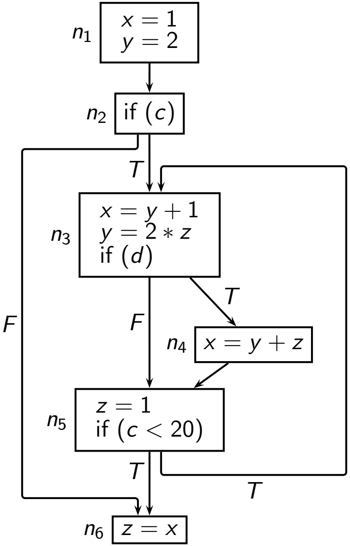
Block	Local Information		Global Information			
			Iteration # 1		Change in iteration # 2	
	Gen_n	$Kill_n$	Out_n	In_n	Out_n	In_n
n_6	$\{x\}$	$\{z\}$	$\emptyset$	$\{x\}$		
n_5	$\{c\}$	$\{z\}$	$\{x\}$	$\{x, c\}$	$\{x, y, z, c, d\}$	$\{x, y, c, d\}$
n_4	$\{y, z\}$	$\{x\}$	$\{x, c\}$	$\{y, z, c\}$	$\{x, y, c, d\}$	$\{y, z, c, d\}$
n_3	$\{y, z, d\}$	$\{x, y\}$	$\{x, y, z, c\}$	$\{y, z, c, d\}$	$\{x, y, z, c, d\}$	
n_2	$\{c\}$	$\emptyset$	$\{x, y, z, c, d\}$	$\{x, y, z, c, d\}$		
n_1	$\emptyset$	$\{x, y\}$	$\{x, y, z, c, d\}$	$\{z, c, d\}$		

Interpreting the Result of Liveness Analysis for Tutorial Problem 3



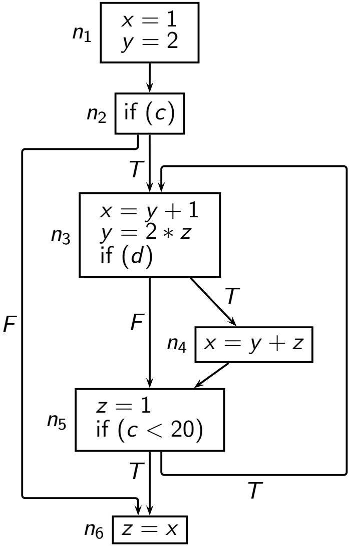
- Why is z live at the exit of n_5?

Interpreting the Result of Liveness Analysis for Tutorial Problem 3



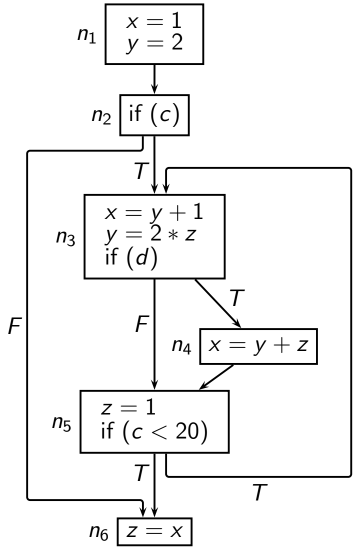
- Why is z live at the exit of n_5?
- Why is z not live at the entry of n_5?

Interpreting the Result of Liveness Analysis for Tutorial Problem 3



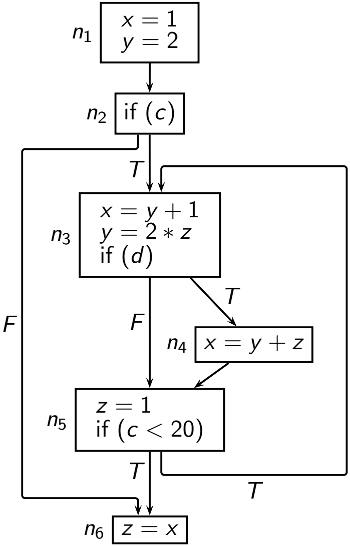
- Why is z live at the exit of n_5?
- Why is z not live at the entry of n_5?
- Why is x live at the exit of n_3 inspite of being killed in n_4?

Interpreting the Result of Liveness Analysis for Tutorial Problem 3



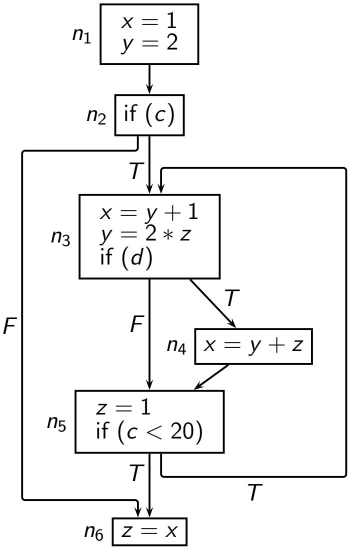
- Why is z live at the exit of n_5?
- Why is z not live at the entry of n_5?
- Why is x live at the exit of n_3 inspite of being killed in n_4?
- Identify the instance of dead code elimination

Interpreting the Result of Liveness Analysis for Tutorial Problem 3



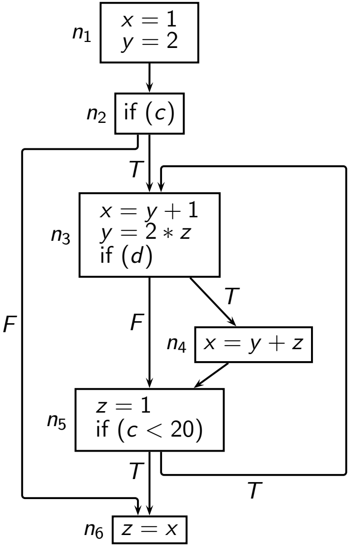
- Why is z live at the exit of n_5?
- Why is z not live at the entry of n_5?
- Why is x live at the exit of n_3 inspite of being killed in n_4?
- Identify the instance of dead code elimination $z = x$ in n_6

Interpreting the Result of Liveness Analysis for Tutorial Problem 3



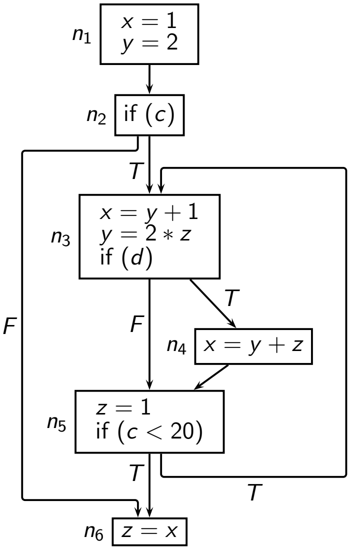
- Why is z live at the exit of n_5?
- Why is z not live at the entry of n_5?
- Why is x live at the exit of n_3 inspite of being killed in n_4?
- Identify the instance of dead code elimination $z = x$ in n_6
- Would the first round of dead code elimination cause liveness information to change?

Interpreting the Result of Liveness Analysis for Tutorial Problem 3



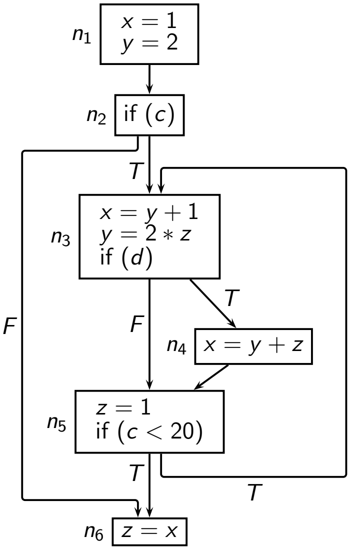
- Why is z live at the exit of n_5?
- Why is z not live at the entry of n_5?
- Why is x live at the exit of n_3 inspite of being killed in n_4?
- Identify the instance of dead code elimination $z = x$ in n_6
- Would the first round of dead code elimination cause liveness information to change? Yes

Interpreting the Result of Liveness Analysis for Tutorial Problem 3



- Why is z live at the exit of n_5?
- Why is z not live at the entry of n_5?
- Why is x live at the exit of n_3 inspite of being killed in n_4?
- Identify the instance of dead code elimination $z = x$ in n_6
- Would the first round of dead code elimination cause liveness information to change? Yes
- Would the second round of liveness analysis lead to further dead code elimination?

Interpreting the Result of Liveness Analysis for Tutorial Problem 3



- Why is z live at the exit of n_5?
- Why is z not live at the entry of n_5?
- Why is x live at the exit of n_3 inspite of being killed in n_4?
- Identify the instance of dead code elimination $z = x$ in n_6
- Would the first round of dead code elimination cause liveness information to change? Yes
- Would the second round of liveness analysis lead to further dead code elimination? Yes

Choice of Initialization

What should be the initial value of internal nodes?

The role of boundary info *BI* explained later in the context of available expressions analysis

Choice of Initialization

What should be the initial value of internal nodes?

- Confluence is $\cup$
- Identity of $\cup$ is $\emptyset$

The role of boundary info *BI* explained later in the context of available expressions analysis

Choice of Initialization

What should be the initial value of internal nodes?

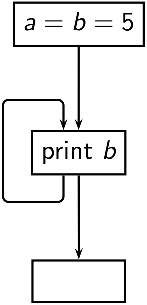
- Confluence is $\cup$
- Identity of $\cup$ is $\emptyset$
- We begin with $\emptyset$ and let the sets at each program point grow

  A revisit to a program point

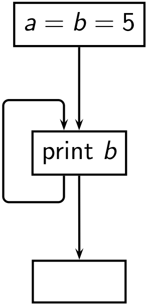
  - may consider a new execution path
  - more variables may be found to be live
  - a variable found to be live earlier does not become dead

The role of boundary info *BI* explained later in the context of available expressions analysis

How Does the Initialization Affect the Solution?



How Does the Initialization Affect the Solution?



Init.

$\emptyset$

$\emptyset$

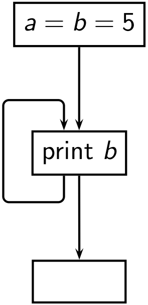
$\emptyset$

$\emptyset$

$\emptyset$

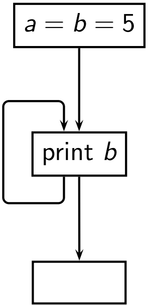
$\emptyset$

How Does the Initialization Affect the Solution?



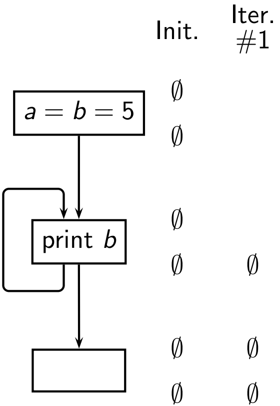
Init.	Iter. #1
$\emptyset$	
$\emptyset$	
$\emptyset$	
$\emptyset$	
$\emptyset$	
$\emptyset$	$\emptyset$

How Does the Initialization Affect the Solution?

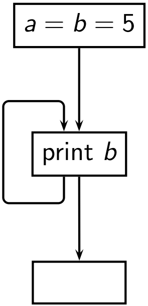


Init.	Iter. #1
$\emptyset$	
$\emptyset$	
$\emptyset$	
$\emptyset$	
$\emptyset$	$\emptyset$
$\emptyset$	$\emptyset$

How Does the Initialization Affect the Solution?

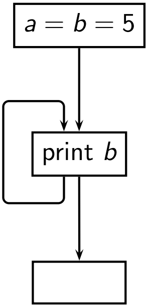


How Does the Initialization Affect the Solution?



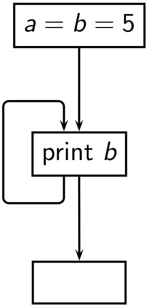
Init.	Iter. #1
$\emptyset$	
$\emptyset$	
$\emptyset$	$\{b\}$
$\emptyset$	$\emptyset$
$\emptyset$	$\emptyset$
$\emptyset$	$\emptyset$

How Does the Initialization Affect the Solution?



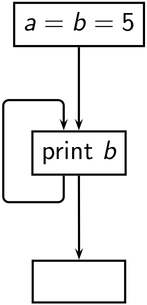
Init.	Iter. #1
$\emptyset$	
$\emptyset$	$\{b\}$
$\emptyset$	$\{b\}$
$\emptyset$	$\emptyset$
$\emptyset$	$\emptyset$
$\emptyset$	$\emptyset$

How Does the Initialization Affect the Solution?



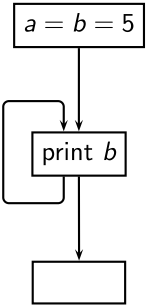
Init.	Iter. #1
$\emptyset$	$\emptyset$
$\emptyset$	$\{b\}$
$\emptyset$	$\{b\}$
$\emptyset$	$\emptyset$
$\emptyset$	$\emptyset$
$\emptyset$	$\emptyset$

How Does the Initialization Affect the Solution?

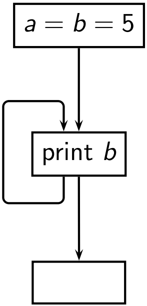


Init.	Iter. #1	Iter. #2
$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	$\{b\}$	$\{b\}$
$\emptyset$	$\{b\}$	$\{b\}$
$\emptyset$	$\emptyset$	$\{b\}$
$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	$\emptyset$	$\emptyset$

How Does the Initialization Affect the Solution?

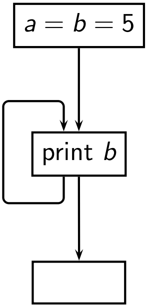


Init.	Iter. #1	Iter. #2
$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	$\{b\}$	$\{b\}$
$\emptyset$	$\{b\}$	$\{b\}$
$\emptyset$	$\emptyset$	$\{b\}$
$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	$\emptyset$	$\emptyset$

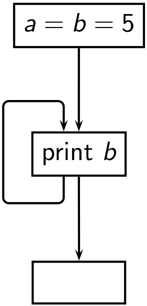


Init.
$\{a, b\}$
$\{a, b\}$
$\{a, b\}$
$\{a, b\}$
$\{a, b\}$
$\emptyset$

How Does the Initialization Affect the Solution?

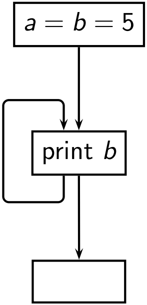


	Init.	Iter. #1	Iter. #2
	$\emptyset$	$\emptyset$	$\emptyset$
	$\emptyset$	$\{b\}$	$\{b\}$
	$\emptyset$	$\{b\}$	$\{b\}$
	$\emptyset$	$\emptyset$	$\{b\}$
	$\emptyset$	$\emptyset$	$\emptyset$
	$\emptyset$	$\emptyset$	$\emptyset$

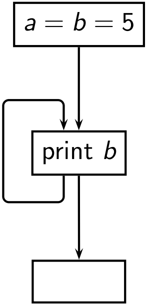


	Init.	Iter. #1
	$\{a, b\}$	
	$\{a, b\}$	
	$\{a, b\}$	
	$\{a, b\}$	
	$\{a, b\}$	
	$\emptyset$	$\emptyset$

How Does the Initialization Affect the Solution?

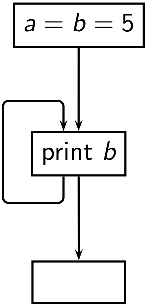


Init.	Iter. #1	Iter. #2
$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	$\{b\}$	$\{b\}$
$\emptyset$	$\{b\}$	$\{b\}$
$\emptyset$	$\emptyset$	$\{b\}$
$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	$\emptyset$	$\emptyset$

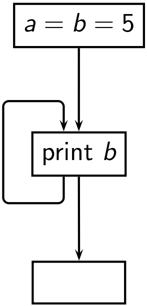


Init.	Iter. #1
$\{a, b\}$	
$\{a, b\}$	
$\{a, b\}$	
$\{a, b\}$	
$\{a, b\}$	$\emptyset$
$\emptyset$	$\emptyset$

How Does the Initialization Affect the Solution?

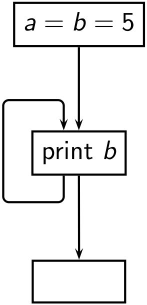


Init.	Iter. #1	Iter. #2
$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	$\{b\}$	$\{b\}$
$\emptyset$	$\{b\}$	$\{b\}$
$\emptyset$	$\emptyset$	$\{b\}$
$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	$\emptyset$	$\emptyset$

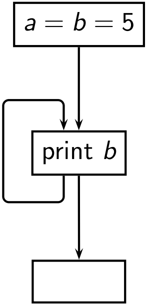


Init.	Iter. #1
$\{a, b\}$	
$\{a, b\}$	
$\{a, b\}$	
$\{a, b\}$	$\{a, b\}$
$\{a, b\}$	$\emptyset$
$\emptyset$	$\emptyset$

How Does the Initialization Affect the Solution?

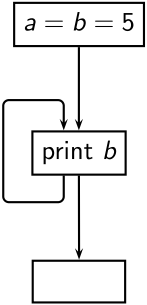


Init.	Iter. #1	Iter. #2
$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	$\{b\}$	$\{b\}$
$\emptyset$	$\{b\}$	$\{b\}$
$\emptyset$	$\emptyset$	$\{b\}$
$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	$\emptyset$	$\emptyset$

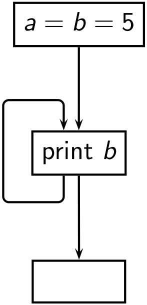


Init.	Iter. #1
$\{a, b\}$	
$\{a, b\}$	
$\{a, b\}$	$\{a, b\}$
$\{a, b\}$	$\{a, b\}$
$\{a, b\}$	$\emptyset$
$\emptyset$	$\emptyset$

How Does the Initialization Affect the Solution?

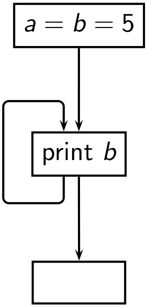


Init.	Iter. #1	Iter. #2
$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	$\{b\}$	$\{b\}$
$\emptyset$	$\{b\}$	$\{b\}$
$\emptyset$	$\emptyset$	$\{b\}$
$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	$\emptyset$	$\emptyset$

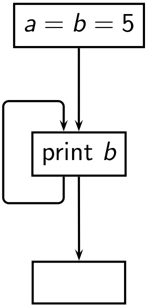


Init.	Iter. #1
$\{a, b\}$	
$\{a, b\}$	$\{a, b\}$
$\{a, b\}$	$\{a, b\}$
$\{a, b\}$	$\{a, b\}$
$\{a, b\}$	$\emptyset$
$\emptyset$	$\emptyset$

How Does the Initialization Affect the Solution?

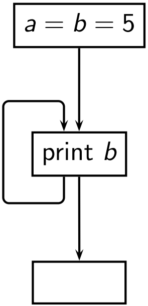


Init.	Iter. #1	Iter. #2
$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	$\{b\}$	$\{b\}$
$\emptyset$	$\{b\}$	$\{b\}$
$\emptyset$	$\emptyset$	$\{b\}$
$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	$\emptyset$	$\emptyset$

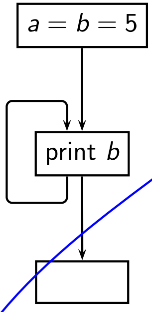


Init.	Iter. #1
$\{a, b\}$	$\emptyset$
$\{a, b\}$	$\{a, b\}$
$\{a, b\}$	$\{a, b\}$
$\{a, b\}$	$\{a, b\}$
$\{a, b\}$	$\emptyset$
$\emptyset$	$\emptyset$

How Does the Initialization Affect the Solution?



Init.	Iter. #1	Iter. #2
$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	$\{b\}$	$\{b\}$
$\emptyset$	$\{b\}$	$\{b\}$
$\emptyset$	$\emptyset$	$\{b\}$
$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	$\emptyset$	$\emptyset$



Init.	Iter. #1
$\{a, b\}$	$\emptyset$
$\{a, b\}$	$\{a, b\}$
$\{a, b\}$	$\{a, b\}$
$\{a, b\}$	$\{a, b\}$
$\{a, b\}$	$\emptyset$
$\emptyset$	$\emptyset$

a is spuriously marked live

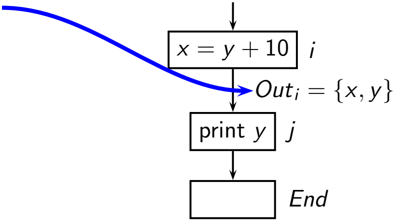
Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

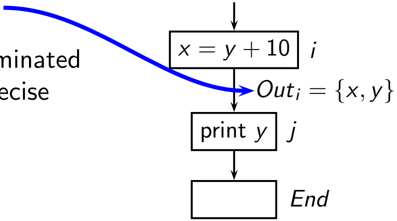
- Spurious inclusion of a non-live variable



Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

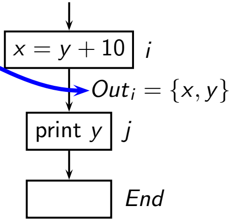
- Spurious inclusion of a non-live variable
  - A dead assignment may not be eliminated
  - Solution is sound but may be imprecise

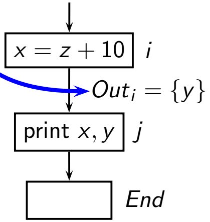


Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

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

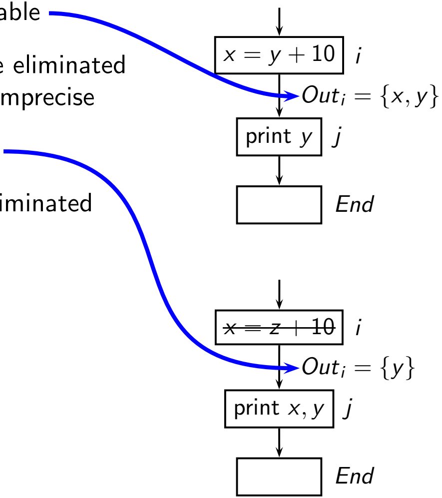




Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

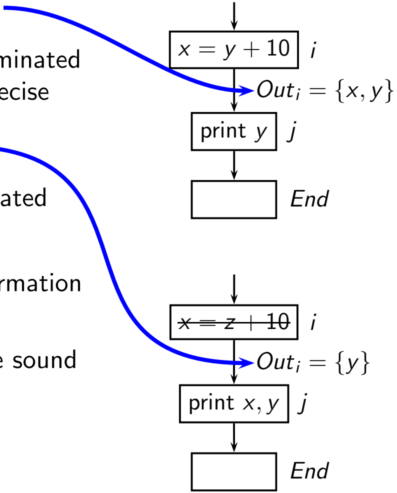
- Spurious inclusion of a non-live variable
  - A dead assignment may not be eliminated
  - Solution is sound but may be imprecise
- Spurious exclusion of a live variable
  - A useful assignment may be eliminated
  - Solution is unsound



Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

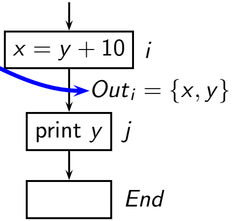
- Spurious inclusion of a non-live variable
  - A dead assignment may not be eliminated
  - Solution is sound but may be imprecise
- Spurious exclusion of a live variable
  - A useful assignment may be eliminated
  - Solution is unsound
- Given $L_2 \supseteq L_1$ representing liveness information
 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

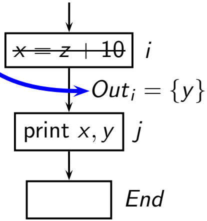


Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

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





Termination, Convergence, and Complexity

- For live variables analysis,
  - The set of all variables is finite, and
  - the confluence operation (i.e. meet) is union, hence
  - the set associated with a data flow variable can only grow

$\Longrightarrow$ Termination is guaranteed

Termination, Convergence, and Complexity

- For live variables analysis,
  - The set of all variables is finite, and
  - the confluence operation (i.e. meet) is union, hence
  - the set associated with a data flow variable can only grow

  $\Longrightarrow$ Termination is guaranteed
- Since initial value is $\emptyset$, live variables analysis converges on the smallest set

Termination, Convergence, and Complexity

- For live variables analysis,
  - The set of all variables is finite, and
  - the confluence operation (i.e. meet) is union, hence
  - the set associated with a data flow variable can only grow

  $\Longrightarrow$ Termination is guaranteed

- Since initial value is $\emptyset$, live variables analysis converges on the smallest set
- How many iterations do we need for reaching the convergence?

Termination, Convergence, and Complexity

- For live variables analysis,
  - The set of all variables is finite, and
  - the confluence operation (i.e. meet) is union, hence
  - the set associated with a data flow variable can only grow

  $\Longrightarrow$ Termination is guaranteed
- Since initial value is $\emptyset$, live variables analysis converges on the smallest set
- How many iterations do we need for reaching the convergence?
- Going beyond live variables analysis
  - Do the sets always grow for other data flow frameworks?
  - What is the complexity of round robin analysis for other analyses?

Termination, Convergence, and Complexity

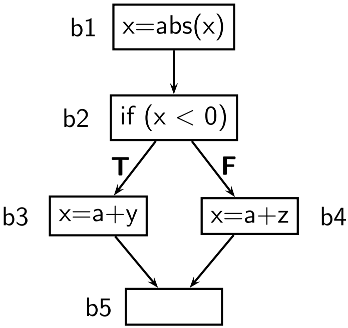
- For live variables analysis,
  - The set of all variables is finite, and
  - the confluence operation (i.e. meet) is union, hence
  - the set associated with a data flow variable can only grow

  $\Longrightarrow$ Termination is guaranteed

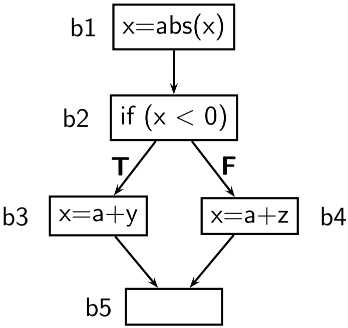
- Since initial value is $\emptyset$, live variables analysis converges on the smallest set
- How many iterations do we need for reaching the convergence?
- Going beyond live variables analysis
  - Do the sets always grow for other data flow frameworks?
  - What is the complexity of round robin analysis for other analyses?

  Answered formally in module 2 (Theoretical Abstractions)

Conservative Nature of Analysis (1)

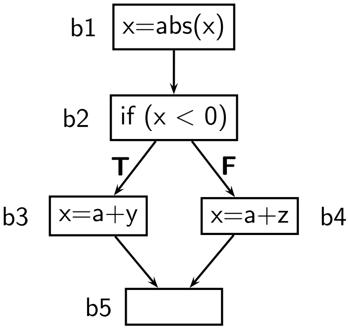


Conservative Nature of Analysis (1)



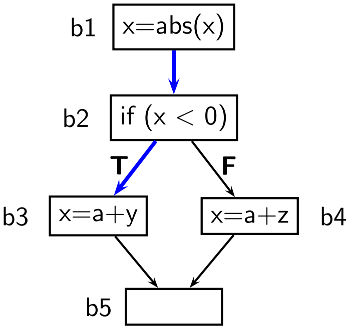
- abs(n) returns the absolute value of n

Conservative Nature of Analysis (1)



- abs(n) returns the absolute value of n
- Is y live on entry to block b2?

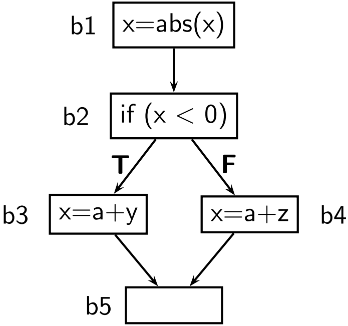
Conservative Nature of Analysis (1)



- abs(n) returns the absolute value of n
- Is y live on entry to block b2?
- By execution semantics, NO

  Path b1→b2→b3 is an infeasible execution path

Conservative Nature of Analysis (1)



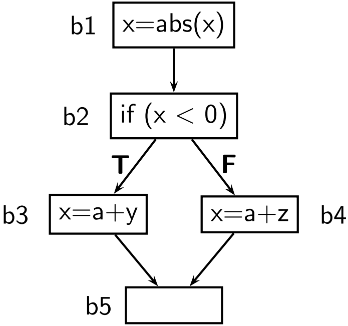
- abs(n) returns the absolute value of n
- Is y live on entry to block b2?
- By execution semantics, NO

  Path b1→b2→b3 is an infeasible execution path
- A compiler makes conservative assumptions:

  All branch outcomes are possible

  ⇒ Consider every path in CFG as a potential execution path

Conservative Nature of Analysis (1)



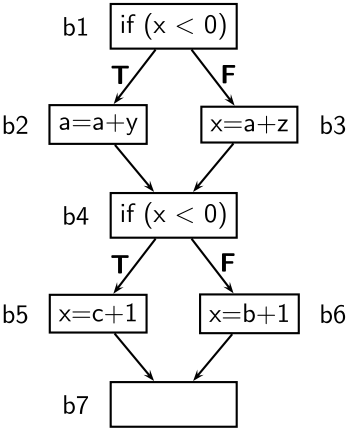
- abs(n) returns the absolute value of n
- Is y live on entry to block b2?
- By execution semantics, NO

  Path b1→b2→b3 is an infeasible execution path
- A compiler makes conservative assumptions:

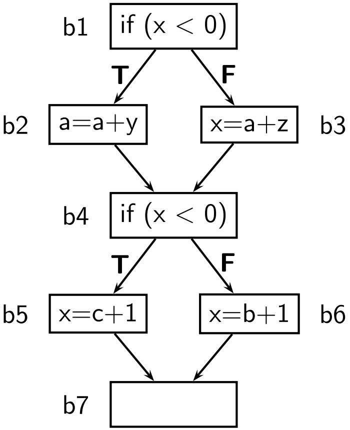
  All branch outcomes are possible

  ⇒ Consider every path in CFG as a potential execution path
- Our analysis concludes that y is live on entry to block b2

Conservative Nature of Analysis (2)

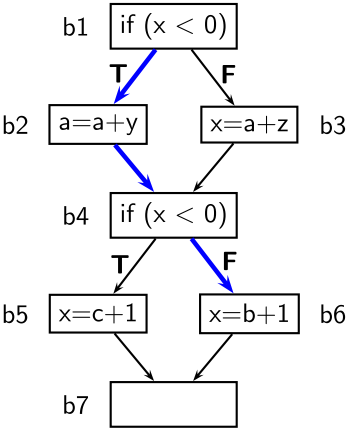


Conservative Nature of Analysis (2)



- Is b live on entry to block b2?

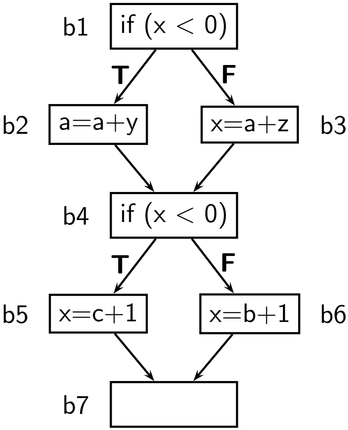
Conservative Nature of Analysis (2)



- Is b live on entry to block b2?
- By execution semantics, NO

  Path b1→b2→b4→b6 is an infeasible execution path

Conservative Nature of Analysis (2)

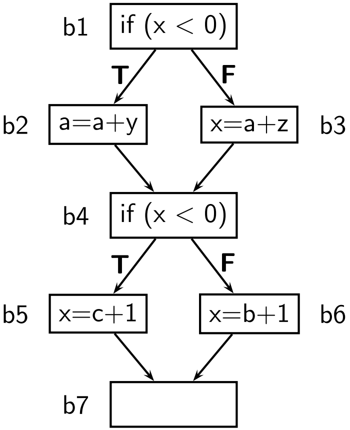


- Is b live on entry to block b2?
- By execution semantics, NO

  Path b1→b2→b4→b6 is an infeasible execution path
- Is c live on entry to block b3?

  Path b1→b3→b4→b6 is a feasible execution path

Conservative Nature of Analysis (2)



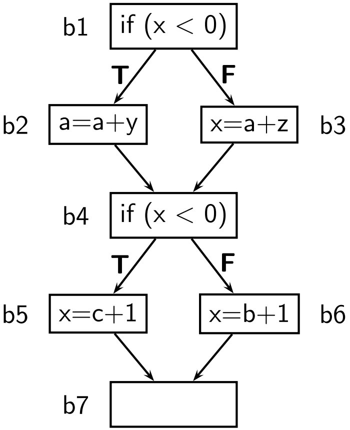
- Is b live on entry to block b2?
- By execution semantics, NO

  Path b1→b2→b4→b6 is an infeasible execution path
- Is c live on entry to block b3?

  Path b1→b3→b4→b6 is a feasible execution path
- A compiler make conservative assumptions
  $\Rightarrow$ our analysis is *path insensitive*

  Note: It is *flow sensitive* (i.e. information is computed for every control flow points)

Conservative Nature of Analysis (2)



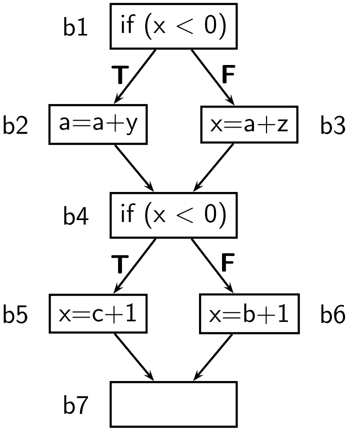
- Is b live on entry to block b2?
- By execution semantics, NO

  Path b1→b2→b4→b6 is an infeasible execution path
- Is c live on entry to block b3?

  Path b1→b3→b4→b6 is a feasible execution path
- A compiler make conservative assumptions ⇒ our analysis is *path insensitive*

  Note: It is *flow sensitive* (i.e. information is computed for every control flow points)
- Our analysis concludes that b is live at the entry of b2

Conservative Nature of Analysis (2)



- Is b live on entry to block b2?
- By execution semantics, NO

  Path b1→b2→b4→b6 is an infeasible execution path
- Is c live on entry to block b3?

  Path b1→b3→b4→b6 is a feasible execution path
- A compiler make conservative assumptions ⇒ our analysis is *path insensitive*

  Note: It is *flow sensitive* (i.e. information is computed for every control flow points)
- Our analysis concludes that b is live at the entry of b2
- Is c live at the entry of b3?

Conservative Nature of Analysis at Intraprocedural Level

- We assume that all paths are potentially executable
- Our analysis is path insensitive
  - The data flow information at a program point p is path insensitive
    - information at p is merged along all paths reaching p
  - The data flow information reaching p is computed path insensitively
    - information is merged at all shared points in paths reaching p
    - may generate spurious information due to non-distributive flow functions

More about it in module 2

Conservative Nature of Analysis at Interprocedural Level

- Context insensitivity
  - Merges of information across all calling contexts
- Flow insensitivity
  - Disregards the control flow

More about it in module 4

What About Soundness of Analysis Results?

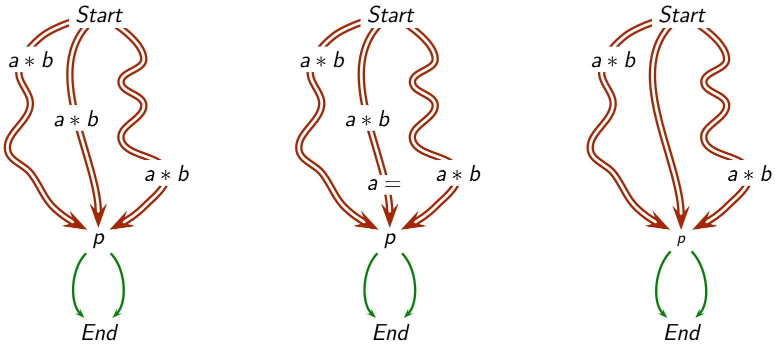
- No compromises
- We will study it in module 2

Part 4

Available Expressions Analysis

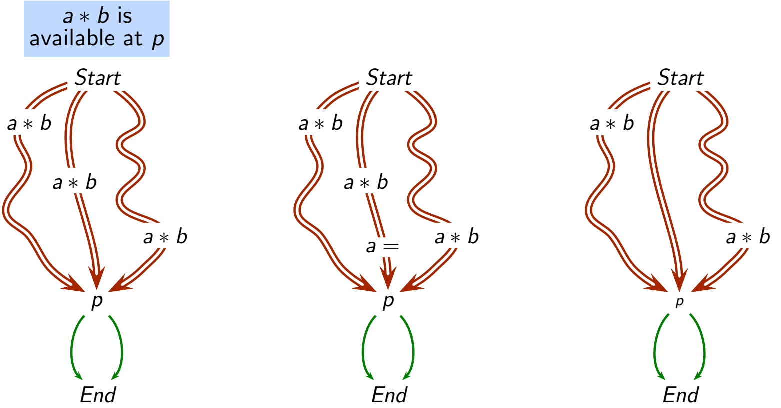
Defining Available Expressions Analysis

An expression *e* is available at a program point *p*, if
every path from program entry to *p* contains an evaluation of *e*
which is not followed by a definition of any operand of *e*.



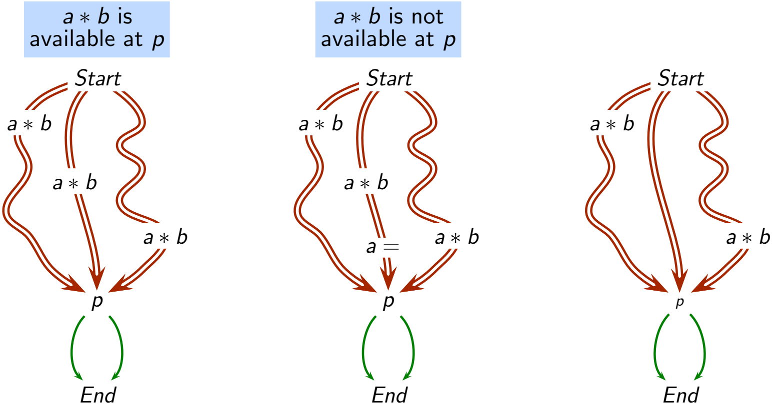
Defining Available Expressions Analysis

An expression e is available at a program point p, if every path from program entry to p contains an evaluation of e which is not followed by a definition of any operand of e.



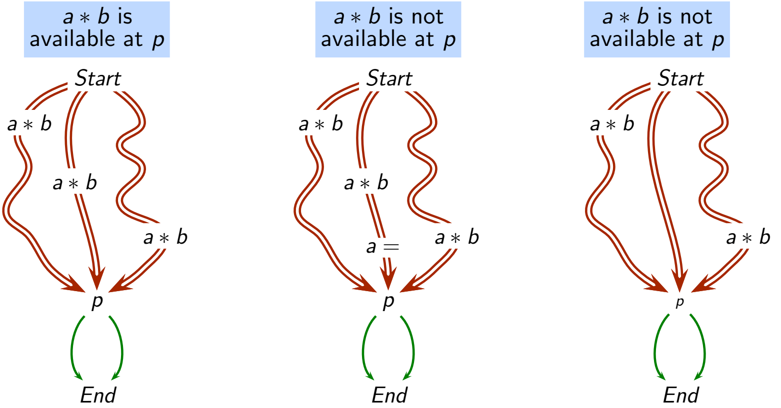
Defining Available Expressions Analysis

An expression e is available at a program point p, if every path from program entry to p contains an evaluation of e which is not followed by a definition of any operand of e.



Defining Available Expressions Analysis

An expression e is available at a program point p, if every path from program entry to p contains an evaluation of e which is not followed by a definition of any operand of e.



Local Data Flow Properties for Available Expressions Analysis

$$Gen_n = \{ e \mid \text{expression } e \text{ is evaluated in basic block } n \text{ and this evaluation is not followed by a definition of any operand of } e\}$$

$$Kill_n = \{ e \mid \text{basic block } n \text{ contains a definition of an operand of } e\}$$

	Entity	Manipulation	Exposition
Gen_n	Expression	Use	Downwards
$Kill_n$	Expression	Modification	Anywhere

Data Flow Equations For Available Expressions Analysis

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

$$Out_n = Gen_n \cup (In_n - Kill_n)$$

Data Flow Equations For Available Expressions Analysis

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

$$Out_n = Gen_n \cup (In_n - Kill_n)$$

Alternatively,

$$Out_n = f_n(In_n), \qquad \text{where}$$

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

Data Flow Equations For Available Expressions Analysis

$$In_n = \begin{cases} BI & n \text{ is } \textit{Start} \text{ block} \\ \displaystyle\bigcap_{p \in pred(n)} Out_p & \text{otherwise} \end{cases}$$

$$Out_n = Gen_n \cup (In_n - Kill_n)$$

Alternatively,

$$Out_n = f_n(In_n), \quad \text{where}$$

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

- In_n and Out_n are sets of expressions

Data Flow Equations For Available Expressions Analysis

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

$$Out_n = Gen_n \cup (In_n - Kill_n)$$

Alternatively,

$$Out_n = f_n(In_n), \quad \text{where}$$

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

- In_n and Out_n are sets of expressions
- BI is $\emptyset$ for expressions involving a local variable

Using Data Flow Information of Available Expressions Analysis

- Common subexpression elimination

Using Data Flow Information of Available Expressions Analysis

- Common subexpression elimination
  - If an expression is available at the entry of a block n (In_n) **and**

Using Data Flow Information of Available Expressions Analysis

- Common subexpression elimination
  - If an expression is available at the entry of a block n (In_n) **and**
  - a computation of the expression exists in n **such that**

Using Data Flow Information of Available Expressions Analysis

- Common subexpression elimination
  - If an expression is available at the entry of a block n (In_n) **and**
  - a computation of the expression exists in n **such that**
  - it is not preceded by a definition of any of its operands ($AntGen_n$)

Using Data Flow Information of Available Expressions Analysis

- Common subexpression elimination
  - If an expression is available at the entry of a block n (In_n) **and**
  - a computation of the expression exists in n **such that**
  - it is not preceded by a definition of any of its operands ($AntGen_n$)

Then the expression is redundant

$$Redundant_n = In_n \cap AntGen_n$$

Using Data Flow Information of Available Expressions Analysis

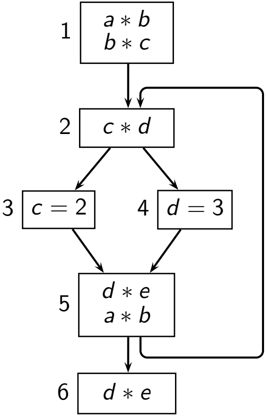
- Common subexpression elimination
  - If an expression is available at the entry of a block n (In_n) **and**
  - a computation of the expression exists in n **such that**
  - it is not preceded by a definition of any of its operands ($AntGen_n$)

  Then the expression is redundant

$$Redundant_n = In_n \cap AntGen_n$$

- A redundant expression is upwards exposed whereas the expressions in Gen_n are downwards exposed

An Example of Available Expressions Analysis

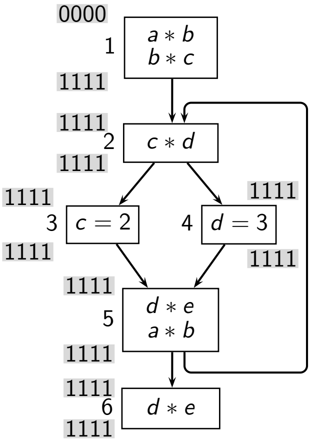


Let $e_1 \equiv a * b$, $e_2 \equiv b * c$, $e_3 \equiv c * d$, $e_4 \equiv d * e$

Node	*Gen*		*Kill*		Available		Redund.	
1	$\{e_1, e_2\}$	1100	$\emptyset$	0000	$\emptyset$	0000	$\emptyset$	0000
2	$\{e_3\}$	0010	$\emptyset$	0000	$\{e_1\}$	1000	$\emptyset$	0000
3	$\emptyset$	0000	$\{e_2, e_3\}$	0110	$\{e_1, e_3\}$	1010	$\emptyset$	0000
4	$\emptyset$	0000	$\{e_3, e_4\}$	0011	$\{e_1, e_3\}$	1010	$\emptyset$	0000
5	$\{e_1, e_4\}$	1001	$\emptyset$	0000	$\{e_1\}$	1000	$\{e_1\}$	1000
6	$\{e_4\}$	0001	$\emptyset$	0000	$\{e_1, e_4\}$	1001	$\{e_4\}$	0001

An Example of Available Expressions Analysis

Initialisation

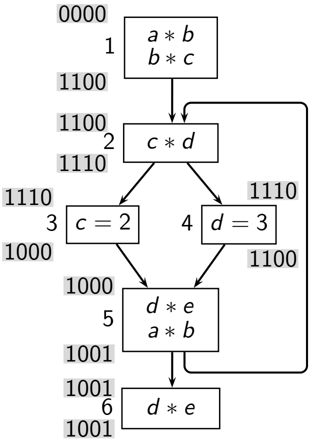


Let $e_1 \equiv a * b$, $e_2 \equiv b * c$, $e_3 \equiv c * d$, $e_4 \equiv d * e$

Node	*Gen*		*Kill*		Available		Redund.	
1	$\{e_1, e_2\}$	1100	$\emptyset$	0000	$\emptyset$	0000	$\emptyset$	0000
2	$\{e_3\}$	0010	$\emptyset$	0000	$\{e_1\}$	1000	$\emptyset$	0000
3	$\emptyset$	0000	$\{e_2, e_3\}$	0110	$\{e_1, e_3\}$	1010	$\emptyset$	0000
4	$\emptyset$	0000	$\{e_3, e_4\}$	0011	$\{e_1, e_3\}$	1010	$\emptyset$	0000
5	$\{e_1, e_4\}$	1001	$\emptyset$	0000	$\{e_1\}$	1000	$\{e_1\}$	1000
6	$\{e_4\}$	0001	$\emptyset$	0000	$\{e_1, e_4\}$	1001	$\{e_4\}$	0001

An Example of Available Expressions Analysis

Iteration #1

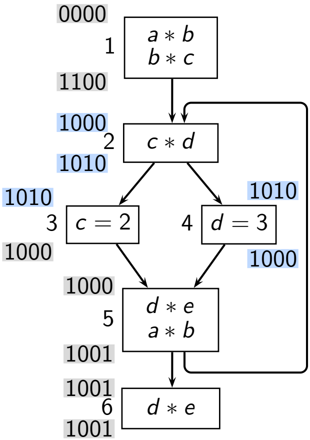


Let $e_1 \equiv a * b$, $e_2 \equiv b * c$, $e_3 \equiv c * d$, $e_4 \equiv d * e$

Node	*Gen*		*Kill*		Available		Redund.	
1	$\{e_1, e_2\}$	1100	$\emptyset$	0000	$\emptyset$	0000	$\emptyset$	0000
2	$\{e_3\}$	0010	$\emptyset$	0000	$\{e_1\}$	1000	$\emptyset$	0000
3	$\emptyset$	0000	$\{e_2, e_3\}$	0110	$\{e_1, e_3\}$	1010	$\emptyset$	0000
4	$\emptyset$	0000	$\{e_3, e_4\}$	0011	$\{e_1, e_3\}$	1010	$\emptyset$	0000
5	$\{e_1, e_4\}$	1001	$\emptyset$	0000	$\{e_1\}$	1000	$\{e_1\}$	1000
6	$\{e_4\}$	0001	$\emptyset$	0000	$\{e_1, e_4\}$	1001	$\{e_4\}$	0001

An Example of Available Expressions Analysis

Iteration #2

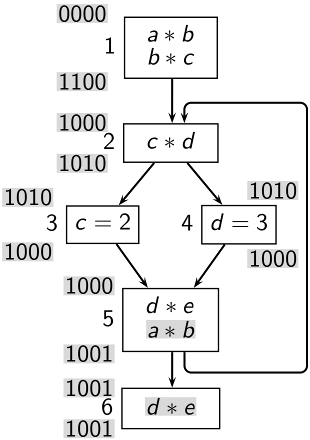


Let $e_1 \equiv a * b$, $e_2 \equiv b * c$, $e_3 \equiv c * d$, $e_4 \equiv d * e$

Node	*Gen*		*Kill*		Available		Redund.	
1	$\{e_1, e_2\}$	1100	$\emptyset$	0000	$\emptyset$	0000	$\emptyset$	0000
2	$\{e_3\}$	0010	$\emptyset$	0000	$\{e_1\}$	1000	$\emptyset$	0000
3	$\emptyset$	0000	$\{e_2, e_3\}$	0110	$\{e_1, e_3\}$	1010	$\emptyset$	0000
4	$\emptyset$	0000	$\{e_3, e_4\}$	0011	$\{e_1, e_3\}$	1010	$\emptyset$	0000
5	$\{e_1, e_4\}$	1001	$\emptyset$	0000	$\{e_1\}$	1000	$\{e_1\}$	1000
6	$\{e_4\}$	0001	$\emptyset$	0000	$\{e_1, e_4\}$	1001	$\{e_4\}$	0001

An Example of Available Expressions Analysis

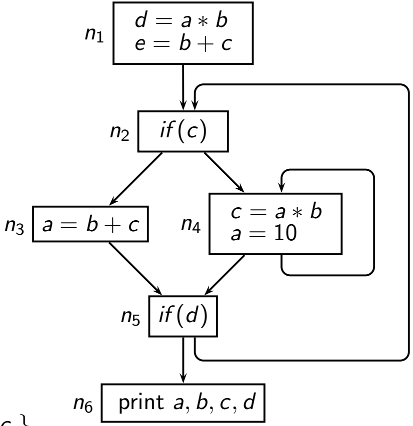
Final Result



Let $e_1 \equiv a * b$, $e_2 \equiv b * c$, $e_3 \equiv c * d$, $e_4 \equiv d * e$

Node	*Gen*		*Kill*		Available		Redund.	
1	$\{e_1, e_2\}$	1100	$\emptyset$	0000	$\emptyset$	0000	$\emptyset$	0000
2	$\{e_3\}$	0010	$\emptyset$	0000	$\{e_1\}$	1000	$\emptyset$	0000
3	$\emptyset$	0000	$\{e_2, e_3\}$	0110	$\{e_1, e_3\}$	1010	$\emptyset$	0000
4	$\emptyset$	0000	$\{e_3, e_4\}$	0011	$\{e_1, e_3\}$	1010	$\emptyset$	0000
5	$\{e_1, e_4\}$	1001	$\emptyset$	0000	$\{e_1\}$	1000	$\{e_1\}$	1000
6	$\{e_4\}$	0001	$\emptyset$	0000	$\{e_1, e_4\}$	1001	$\{e_4\}$	0001

Tutorial Problem 2 for Available Expressions Analysis



$\mathbb{E}\text{xpr} = \{\ a * b,\ b + c\ \}$

Solution of the Tutorial Problem 2

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

Node	Local Information			Global Information				
				Iteration # 1		Changes in iteration # 2		$Redundant_n$
	Gen_n	$Kill_n$	$AntGen_n$	In_n	Out_n	In_n	Out_n	
n_1	11	00	11					
n_2	00	00	00					
n_3	01	10	01					
n_4	00	11	10					
n_5	00	00	00					
n_6	00	00	00					

Solution of the Tutorial Problem 2

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

Node	Local Information			Global Information				
				Iteration # 1		Changes in iteration # 2		$Redundant_n$
	Gen_n	$Kill_n$	$AntGen_n$	In_n	Out_n	In_n	Out_n	
n_1	11	00	11	00	11			
n_2	00	00	00	11	11			
n_3	01	10	01	11	01			
n_4	00	11	10	11	00			
n_5	00	00	00	00	00			
n_6	00	00	00	00	00			

Solution of the Tutorial Problem 2

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

Node	Local Information			Global Information				
				Iteration # 1		Changes in iteration # 2		$Redundant_n$
	Gen_n	$Kill_n$	$AntGen_n$	In_n	Out_n	In_n	Out_n	
n_1	11	00	11	00	11			
n_2	00	00	00	11	11	00	00	
n_3	01	10	01	11	01	00		
n_4	00	11	10	11	00	00		
n_5	00	00	00	00	00			
n_6	00	00	00	00	00			

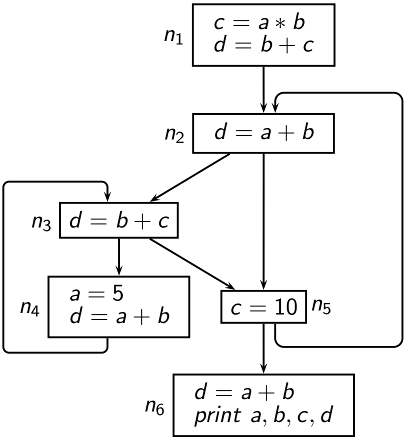
Solution of the Tutorial Problem 2

Bit vector

$a * b$	$b + c$

Node	Local Information			Global Information				
				Iteration # 1		Changes in iteration # 2		$Redundant_n$
	Gen_n	$Kill_n$	$AntGen_n$	In_n	Out_n	In_n	Out_n	
n_1	11	00	11	00	11			00
n_2	00	00	00	11	11	00	00	00
n_3	01	10	01	11	01	00		00
n_4	00	11	10	11	00	00		00
n_5	00	00	00	00	00			00
n_6	00	00	00	00	00			00

Tutorial Problem 3 for Available Expressions Analysis



$\mathbb{E}\text{xpr} = \{\ a * b,\ b + c,\ a + b\ \}$

Solution of the Tutorial Problem 3

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

Node	Local Information			Global Information						
				Iteration # 1		Changes in Iteration # 2		Changes in Iteration # 3		$Redundant_n$
	Gen_n	$Kill_n$	$AntGen_n$	In_n	Out_n	In_n	Out_n	In_n	Out_n	
n_1	110	010	100							
n_2	001	000	001							
n_3	010	000	010							
n_4	001	101	000							
n_5	000	010	000							
n_6	001	000	001							

Solution of the Tutorial Problem 3

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

Node	Local Information			Global Information						
				Iteration # 1		Changes in Iteration # 2		Changes in Iteration # 3		$Redundant_n$
	Gen_n	$Kill_n$	$AntGen_n$	In_n	Out_n	In_n	Out_n	In_n	Out_n	
n_1	110	010	100	000	110					
n_2	001	000	001	110	111					
n_3	010	000	010	111	111					
n_4	001	101	000	111	011					
n_5	000	010	000	111	101					
n_6	001	000	001	101	101					

Solution of the Tutorial Problem 3

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

Node	Local Information			Global Information						
				Iteration # 1		Changes in Iteration # 2		Changes in Iteration # 3		$Redundant_n$
	Gen_n	$Kill_n$	$AntGen_n$	In_n	Out_n	In_n	Out_n	In_n	Out_n	
n_1	110	010	100	000	110					
n_2	001	000	001	110	111	100	101			
n_3	010	000	010	111	111	001	011			
n_4	001	101	000	111	011	011				
n_5	000	010	000	111	101	001	001			
n_6	001	000	001	101	101	001	001			

Solution of the Tutorial Problem 3

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

Node	Local Information			Global Information						
				Iteration # 1		Changes in Iteration # 2		Changes in Iteration # 3		$Redundant_n$
	Gen_n	$Kill_n$	$AntGen_n$	In_n	Out_n	In_n	Out_n	In_n	Out_n	
n_1	110	010	100	000	110					
n_2	001	000	001	110	111	100	101	000	001	
n_3	010	000	010	111	111	001	011			
n_4	001	101	000	111	011	011				
n_5	000	010	000	111	101	001	001			
n_6	001	000	001	101	101	001	001			

Solution of the Tutorial Problem 3

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

Node	Local Information			Global Information						
				Iteration # 1		Changes in Iteration # 2		Changes in Iteration # 3		$Redundant_n$
	Gen_n	$Kill_n$	$AntGen_n$	In_n	Out_n	In_n	Out_n	In_n	Out_n	
n_1	110	010	100	000	110					000
n_2	001	000	001	110	111	100	101	000	001	000
n_3	010	000	010	111	111	001	011			000
n_4	001	101	000	111	011	011				000
n_5	000	010	000	111	101	001	001			000
n_6	001	000	001	101	101	001	001			001

Solution of the Tutorial Problem 3

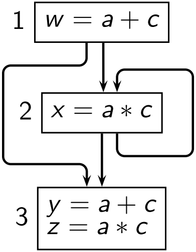
Bit vector

$a * b$	$b + c$	$a + b$

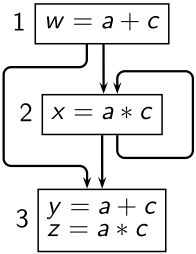
Node	Local Information			Global Information						
				Iteration # 1		Changes in Iteration # 2		Changes in Iteration # 3		$Redundant_n$
	Gen_n	$Kill_n$	$AntGen_n$	In_n	Out_n	In_n	Out_n	In_n	Out_n	
n_1	110	010	100	000	110					000
n_2	001	000	001	110	111	100	101	000	001	000
n_3	010	000	010	111	111	001	011			000
n_4	001	101	000	111	011	011				000
n_5	000	010	000	111	101	001	001			000
n_6	001	000	001	101	101	001	001			001

Why do we need 3 iterations as against 2 for previous problems?

The Effect of BI and Initialization on a Solution



The Effect of *BI* and Initialization on a Solution

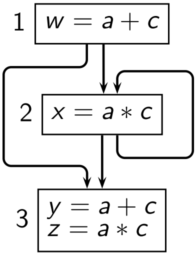


Bit Vector

$a + c$	$a * c$

BI	Node	Initialization $\mathbb{U}$		Initialization $\emptyset$	
		In_n	Out_n	In_n	Out_n
$\emptyset$	1				
	2				
	3				
$\mathbb{U}$	1				
	2				
	3				

The Effect of BI and Initialization on a Solution

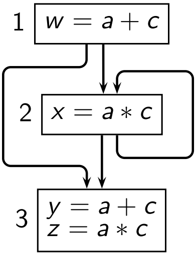


Bit Vector

$a + c$	$a * c$

BI	Node	Initialization $\mathbb{U}$		Initialization $\emptyset$	
		In_n	Out_n	In_n	Out_n
$\emptyset$	1	00	10		
	2	10	11		
	3	10	11		
$\mathbb{U}$	1				
	2				
	3				

The Effect of *BI* and Initialization on a Solution

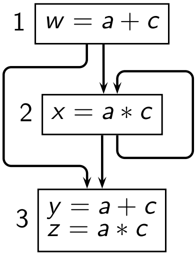


Bit Vector

$a+c$	$a*c$

BI	Node	Initialization $\mathbb{U}$		Initialization $\emptyset$	
		In_n	Out_n	In_n	Out_n
$\emptyset$	1	00	10	00	10
	2	10	11	00	01
	3	10	11	01	11
$\mathbb{U}$	1				
	2				
	3				

The Effect of BI and Initialization on a Solution



Bit Vector

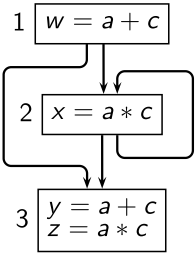
$a + c$	$a * c$

BI	Node	Initialization $\mathbb{U}$		Initialization $\emptyset$	
		In_n	Out_n	In_n	Out_n
$\emptyset$	1	00	10	00	10
	2	10	11	00	01
	3	10	11	01	11
$\mathbb{U}$	1	11	11		
	2	11	11		
	3	11	11		

The Effect of BI and Initialization on a Solution

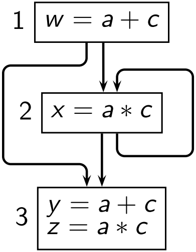
Bit Vector

$a+c$	$a*c$



BI	Node	Initialization $\mathbb{U}$		Initialization $\emptyset$	
		In_n	Out_n	In_n	Out_n
$\emptyset$	1	00	10	00	10
	2	10	11	00	01
	3	10	11	01	11
$\mathbb{U}$	1	11	11	11	11
	2	11	11	00	01
	3	11	11	01	11

The Effect of *BI* and Initialization on a Solution



Bit Vector

$a + c$	$a * c$

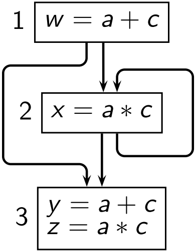
BI	Node	Initialization $\mathbb{U}$		Initialization $\emptyset$	
		In_n	Out_n	In_n	Out_n
$\emptyset$	1	00	10	00	10
	2	10	11	00	01
	3	10	11	01	11
$\mathbb{U}$	1	11	11	11	11
	2	11	11	00	01
	3	11	11	01	11

This represents the expected availability information leading to elimination of $a + c$ in node 3 ($a * c$ is not redundant in node 3)

The Effect of *BI* and Initialization on a Solution

Bit Vector

$a + c$	$a * c$



BI	Node	Initialization $\mathbb{U}$		Initialization $\emptyset$	
		In_n	Out_n	In_n	Out_n
$\emptyset$	1	00	10	00	10
	2	10	11	00	01
	3	10	11	01	11
$\mathbb{U}$	1	11	11	11	11
	2	11	11	00	01
	3	11	11	01	11

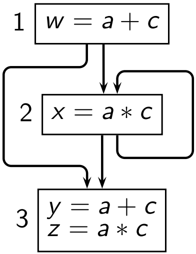
This misses the availability of $a + c$ in node 3

The Effect of BI and Initialization on a Solution

This makes $a * c$ available in node 3 although its computation in node 3 is not redundant

Bit Vector

$a + c$	$a * c$



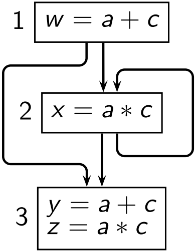
BI	Node	Initialization $\mathbb{U}$		Initialization $\emptyset$	
		In_n	Out_n	In_n	Out_n
$\emptyset$	1	00	10	00	10
	2	10	11	00	01
	3	10	11	01	11
$\mathbb{U}$	1	11	11	11	11
	2	11	11	00	01
	3	11	11	01	11

The Effect of *BI* and Initialization on a Solution

This make $a * c$ available in node 3 and but misses the availability of $a + c$ in node 3

Bit Vector

$a + c$	$a * c$

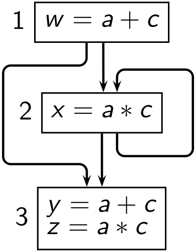


BI	Node	Initialization $\mathbb{U}$		Initialization $\emptyset$	
		In_n	Out_n	In_n	Out_n
$\emptyset$	1	00	10	00	10
	2	10	11	00	01
	3	10	11	01	11
$\mathbb{U}$	1	11	11	11	11
	2	11	11	00	01
	3	11	11	01	11

The Effect of *BI* and Initialization on a Solution

Bit Vector

$a + c$	$a * c$



BI	Node	Initialization $\mathbb{U}$		Initialization $\emptyset$	
		In_n	Out_n	In_n	Out_n
$\emptyset$	1	Sound & Precise		Sound & Imprecise	
	2				
	3				
$\mathbb{U}$	1	Unsound		Unsound	
	2				
	3				

Some Observations

- Data flow equations do not require a particular order of computation
  - Specification. Data flow equations define what needs to be computed and not how it is to be computed
  - Implementation. Round robin iterations perform the actual computation
  - Specification and implementation are distinct
- Initialization governs the quality of solution found
  - Only precision is affected, soundness is guaranteed
  - Associated with "internal" nodes
- *BI* depends on the semantics of the calling context
  - May cause unsoundness
  - Associated with "boundary" node (specified by data flow equations) Does not vary with the method or order of traversal

Still More Tutorial Problems ☺

A New Data Flow Framework: Partially available expressions analysis

- Expressions that are computed and remain unmodified along some path reaching p
- The data flow equations are same as that of available expressions analysis except that the confluence is changed to $\cup$

Perform partially available expressions analysis for the example program used for available expressions analysis

Solution of the Tutorial Problem 2 for Partial Availability Analysis

Bit vector

$a * b$	$b + c$

Node	Local Information			Global Information		
				Iteration # 1		$ParRedund_n$
	Gen_n	$Kill_n$	$AntGen_n$	$PavIn_n$	$PavOut_n$	
n_1	11	00	11			
n_2	00	00	00			
n_3	01	10	01			
n_4	00	11	10			
n_5	00	00	00			
n_6	00	00	00			

Solution of the Tutorial Problem 2 for Partial Availability Analysis

Bit vector

$a * b$	$b + c$

Node	Local Information			Global Information		
				Iteration # 1		$ParRedund_n$
	Gen_n	$Kill_n$	$AntGen_n$	$PavIn_n$	$PavOut_n$	
n_1	11	00	11	00	11	
n_2	00	00	00	11	11	
n_3	01	10	01	11	01	
n_4	00	11	10	11	00	
n_5	00	00	00	01	01	
n_6	00	00	00	01	01	

Solution of the Tutorial Problem 2 for Partial Availability Analysis

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

Node	Local Information			Global Information		
				Iteration # 1		$ParRedund_n$
	Gen_n	$Kill_n$	$AntGen_n$	$PavIn_n$	$PavOut_n$	
n_1	11	00	11	00	11	00
n_2	00	00	00	11	11	00
n_3	01	10	01	11	01	01
n_4	00	11	10	11	00	10
n_5	00	00	00	01	01	00
n_6	00	00	00	01	01	00

Solution of the Tutorial Problem 3 for Partial Availability Analysis

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

Node	Local Information			Global Information				
				Iteration # 1		Changes in iteration # 2		$ParRedund_n$
	Gen_n	$Kill_n$	$AntGen_n$	$PavIn_n$	$PavOut_n$	In_n	Out_n	
n_1	110	010	100					
n_2	001	000	001					
n_3	010	000	010					
n_4	001	101	000					
n_5	000	010	000					
n_6	001	000	001					

Solution of the Tutorial Problem 3 for Partial Availability Analysis

Bit vector

$a * b$	$b + c$	$a + b$

Node	Local Information			Global Information				
				Iteration # 1		Changes in iteration # 2		$ParRedund_n$
	Gen_n	$Kill_n$	$AntGen_n$	$PavIn_n$	$PavOut_n$	In_n	Out_n	
n_1	110	010	100	000	110			
n_2	001	000	001	110	111			
n_3	010	000	010	111	111			
n_4	001	101	000	111	011			
n_5	000	010	000	111	101			
n_6	001	000	001	101	101			

Solution of the Tutorial Problem 3 for Partial Availability Analysis

Bit vector

$a * b$	$b + c$	$a + b$

Node	Local Information			Global Information				
				Iteration # 1		Changes in iteration # 2		$ParRedund_n$
	Gen_n	$Kill_n$	$AntGen_n$	$PavIn_n$	$PavOut_n$	In_n	Out_n	
n_1	110	010	100	000	110			
n_2	001	000	001	110	111	111		
n_3	010	000	010	111	111			
n_4	001	101	000	111	011			
n_5	000	010	000	111	101			
n_6	001	000	001	101	101			

Solution of the Tutorial Problem 3 for Partial Availability Analysis

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

Node	Local Information			Global Information				
				Iteration # 1		Changes in iteration # 2		$ParRedund_n$
	Gen_n	$Kill_n$	$AntGen_n$	$PavIn_n$	$PavOut_n$	In_n	Out_n	
n_1	110	010	100	000	110			000
n_2	001	000	001	110	111	111		001
n_3	010	000	010	111	111			010
n_4	001	101	000	111	011			000
n_5	000	010	000	111	101			000
n_6	001	000	001	101	101			001

Part 5

Reaching Definitions Analysis

Defining Reaching Definitions Analysis

- A definition $d_x : x = e$ reaches a program point p if it appears (without a redefinition of x) on some path from program entry to p

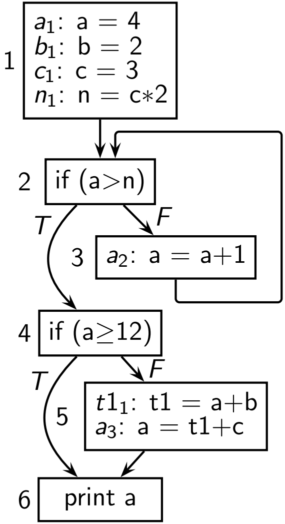
  (x is a variable and e is an expression)

- Application : Copy Propagation

  A use of a variable x at a program point p can be replaced by y if $d_x : x = y$ is the only definition which reaches p and y is not modified between the point of d_x and p.

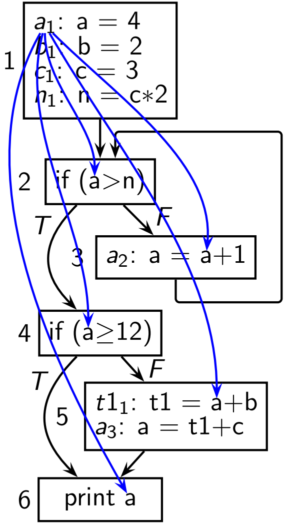
Using Reaching Definitions for Def-Use and Use-Def Chains

Def-Use Chains



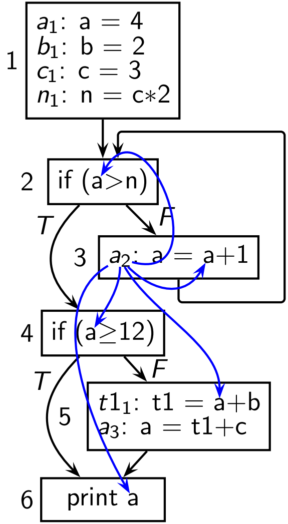
Using Reaching Definitions for Def-Use and Use-Def Chains

Def-Use Chains



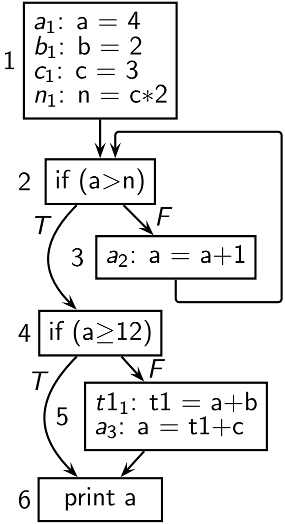
Using Reaching Definitions for Def-Use and Use-Def Chains

Def-Use Chains

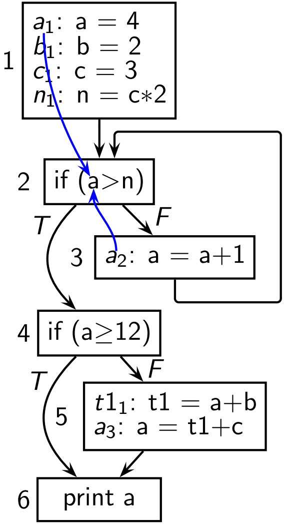


Using Reaching Definitions for Def-Use and Use-Def Chains

Def-Use Chains

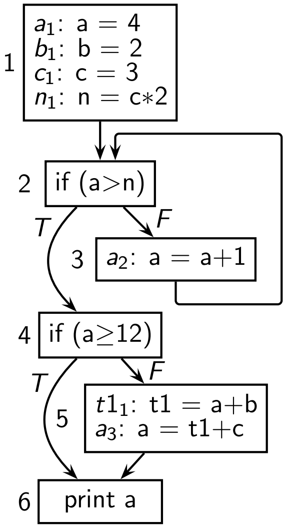


Use-Def Chains

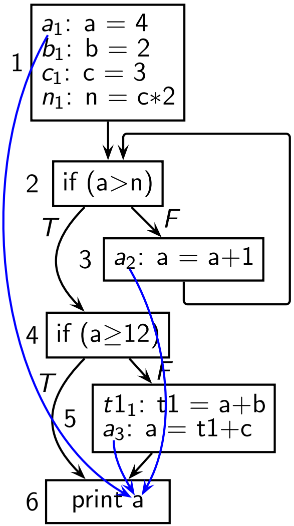


Using Reaching Definitions for Def-Use and Use-Def Chains

Def-Use Chains

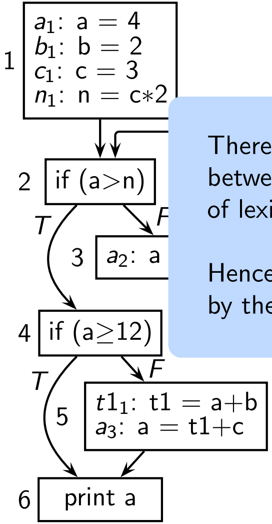


Use-Def Chains

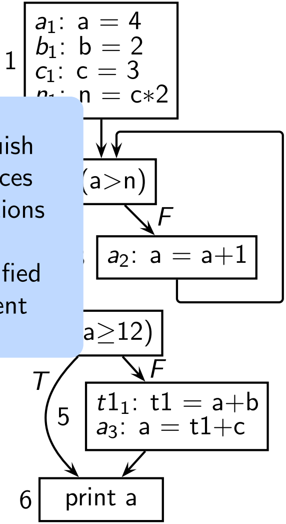


Using Reaching Definitions for Def-Use and Use-Def Chains

Def-Use Chains



Use-Def Chains



There is a need to distinguish between different occurrences of lexically identical definitions

Hence a definition is identified by the label of the statement

Defining Data Flow Analysis for Reaching Definitions Analysis

Let d_v be a definition of variable v

$$Gen_n = \{ d_v \mid \text{variable } v \text{ is defined in basic block } n \text{ and this definition is not followed (within } n\text{) by a definition of } v\}$$

$$Kill_n = \{ d_v \mid \text{basic block } n \text{ contains a definition of } v\}$$

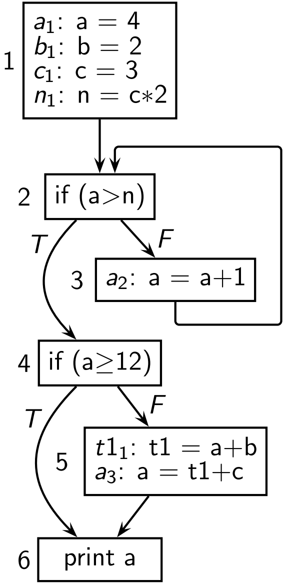
	Entity	Manipulation	Exposition
Gen_n	Definition	Occurrence	Downwards
$Kill_n$	Definition	Occurrence	Anywhere

Data Flow Equations for Reaching Definitions Analysis

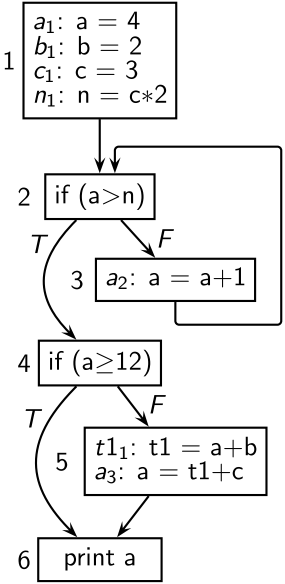
$$
\begin{aligned}
In_n &= \begin{cases} BI & n \text{ is } \mathit{Start} \text{ block} \\ \bigcup\limits_{p \in pred(n)} Out_p & \text{otherwise} \end{cases} \\
Out_n &= Gen_n \cup (In_n - Kill_n) \\
BI &= \{d_x : x = undef \mid x \in \mathbb{V}\text{ar}\}
\end{aligned}
$$

In_n and Out_n are sets of definitions

Tutorial Problem for Copy Propagation

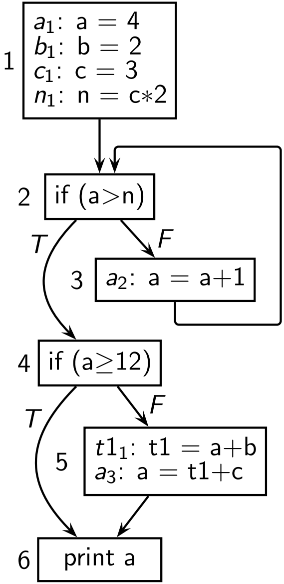


Tutorial Problem for Copy Propagation

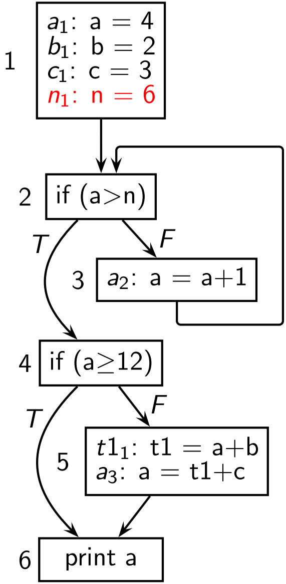


Local copy propagation and constant folding

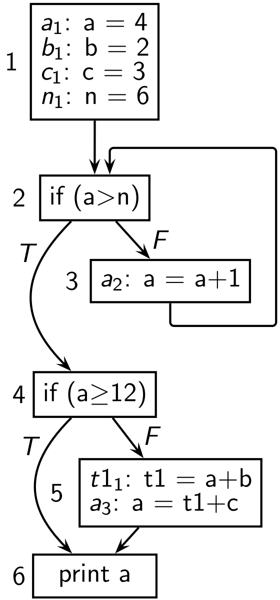
Tutorial Problem for Copy Propagation



Local copy propagation and constant folding

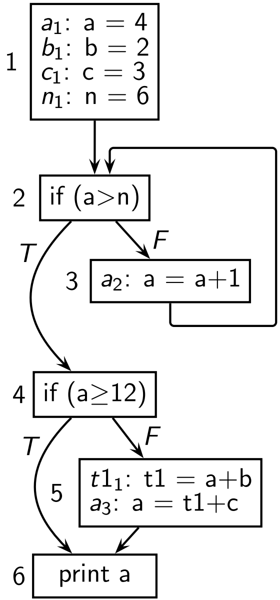


Tutorial Problem for Copy Propagation



	Gen	*Kill*
n1	$\{a_1, b_1, c_1, n_1\}$	$\{a_0, a_1, a_2, a_3, b_0, b_1, c_0, c_1, n_0, n_1\}$
n2	$\emptyset$	$\emptyset$
n3	$\{a_2\}$	$\{a_0, a_1, a_2, a_3\}$
n4	$\emptyset$	$\emptyset$
n5	$\{a_3\}$	$\{a_0, a_1, a_2, a_3\}$
n6	$\emptyset$	$\emptyset$

Tutorial Problem for Copy Propagation

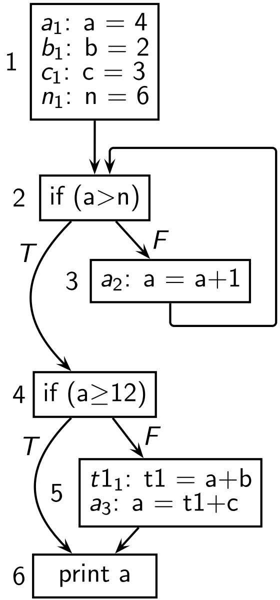


	Gen	*Kill*
n1	$\{a_1, b_1, c_1, n_1\}$	$\{a_0, a_1, a_2, a_3, b_0, b_1, c_0, c_1, n_0, n_1\}$
n2	$\emptyset$	$\emptyset$
n3	$\{a_2\}$	$\{a_0, a_1, a_2, a_3\}$
n4	$\emptyset$	$\emptyset$
n5	$\{a_3\}$	$\{a_0, a_1, a_2, a_3\}$
n6	$\emptyset$	$\emptyset$

- Temporary variable t1 is ignored
- For variable v, v_0 denotes the definition $v = ?$

  This is used for defining *BI*

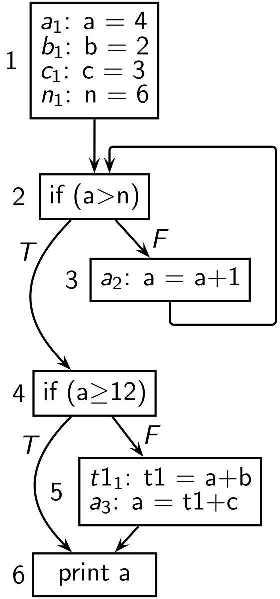
Tutorial Problem for Copy Propagation



	Gen	*Kill*
n1	$\{a_1, b_1, c_1, n_1\}$	$\{a_0, a_1, a_2, a_3, b_0, b_1, c_0, c_1, n_0, n_1\}$
n2	$\emptyset$	$\emptyset$
n3	$\{a_2\}$	$\{a_0, a_1, a_2, a_3\}$
n4	$\emptyset$	$\emptyset$
n5	$\{a_3\}$	$\{a_0, a_1, a_2, a_3\}$
n6	$\emptyset$	$\emptyset$

	Iteration #1	
	In	*Out*
n1	$\{a_0, b_0, c_0, n_0\}$	$\{a_1, b_1, c_1, n_1\}$
n2	$\{a_1, b_1, c_1, n_1\}$	$\{a_1, b_1, c_1, n_1\}$
n3	$\{a_1, b_1, c_1, n_1\}$	$\{a_2, b_1, c_1, n_1\}$
n4	$\{a_1, b_1, c_1, n_1\}$	$\{a_1, b_1, c_1, n_1\}$
n5	$\{a_1, b_1, c_1, n_1\}$	$\{a_3, b_1, c_1, n_1\}$
n6	$\{a_1, a_3, b_1, c_1, n_1\}$	$\{a_1, a_3, b_1, c_1, n_1\}$

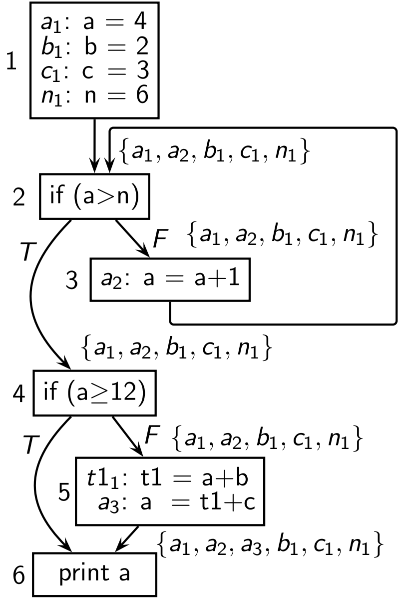
Tutorial Problem for Copy Propagation



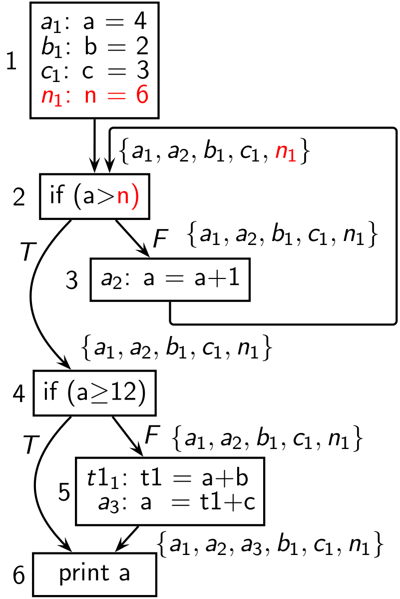
	Gen	*Kill*
n1	$\{a_1, b_1, c_1, n_1\}$	$\{a_0, a_1, a_2, a_3, b_0, b_1, c_0, c_1, n_0, n_1\}$
n2	$\emptyset$	$\emptyset$
n3	$\{a_2\}$	$\{a_0, a_1, a_2, a_3\}$
n4	$\emptyset$	$\emptyset$
n5	$\{a_3\}$	$\{a_0, a_1, a_2, a_3\}$
n6	$\emptyset$	$\emptyset$

	Iteration #2	
	In	*Out*
n1	$\{a_0, b_0, c_0, n_0\}$	$\{a_1, b_1, c_1, n_1\}$
n2	$\{a_1, a_2, b_1, c_1, n_1\}$	$\{a_1, a_2, b_1, c_1, n_1\}$
n3	$\{a_1, a_2, b_1, c_1, n_1\}$	$\{a_2, b_1, c_1, n_1\}$
n4	$\{a_1, a_2, b_1, c_1, n_1\}$	$\{a_1, a_2, b_1, c_1, n_1\}$
n5	$\{a_1, a_2, b_1, c_1, n_1\}$	$\{a_3, b_1, c_1, n_1\}$
n6	$\{a_1, a_2, a_3, b_1, c_1, n_1\}$	$\{a_1, a_2, a_3, b_1, c_1, n_1\}$

Tutorial Problem for Copy Propagation

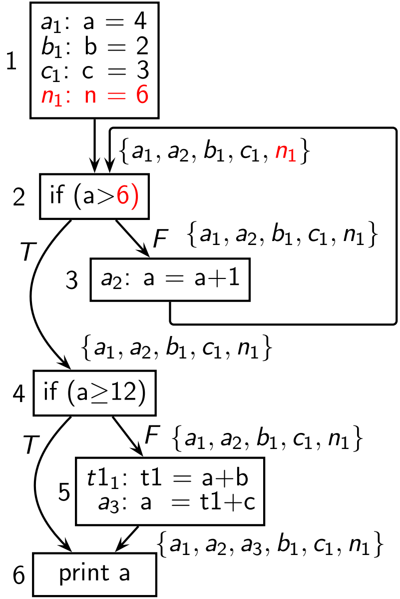


Tutorial Problem for Copy Propagation



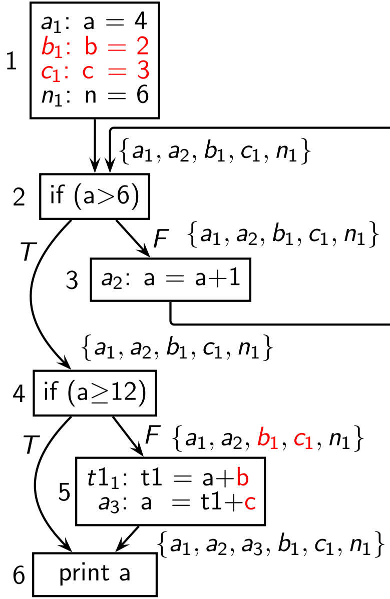
- RHS of n_1 is constant and hence cannot change

Tutorial Problem for Copy Propagation



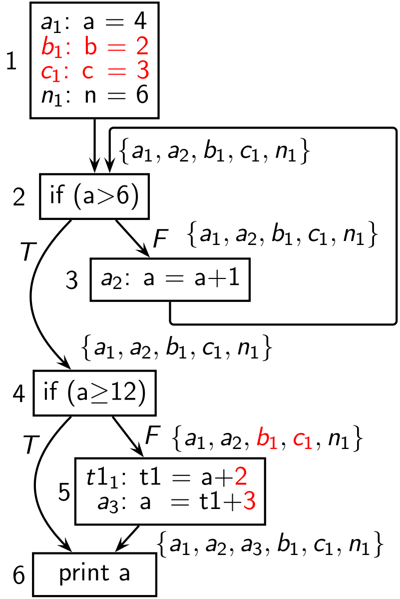
- RHS of n_1 is constant and hence cannot change
- In block 2, n can be replaced by 6

Tutorial Problem for Copy Propagation



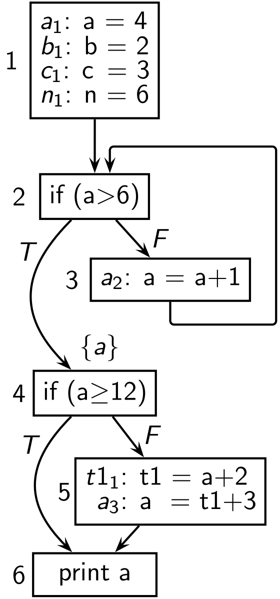
- RHS of n_1 is constant and hence cannot change
- In block 2, n can be replaced by 6
- RHS of b_1 and c_1 are constant and hence cannot change

Tutorial Problem for Copy Propagation

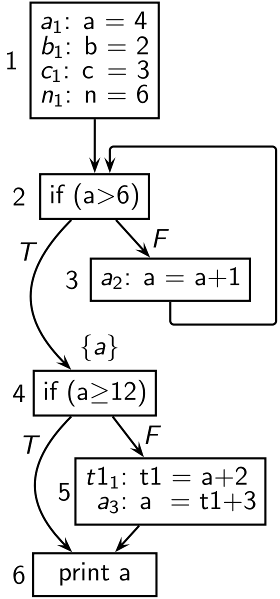


- RHS of n_1 is constant and hence cannot change
- In block 2, n can be replaced by 6
- RHS of b_1 and c_1 are constant and hence cannot change
- In block 5, b can be replaced by 2 and c can be replaced by 3

Tutorial Problem for Copy Propagation

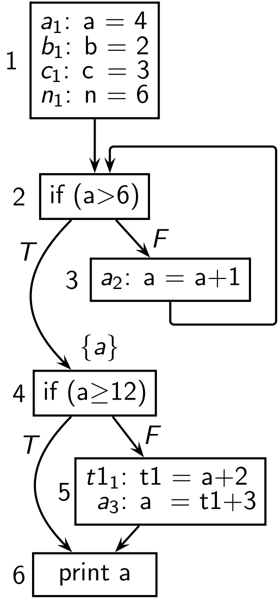


Tutorial Problem for Copy Propagation



So what is the advantage?

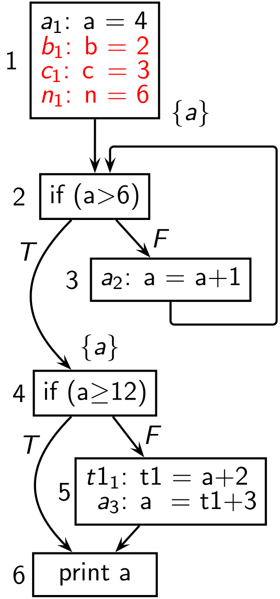
Tutorial Problem for Copy Propagation



So what is the advantage?

Dead Code Elimination

Tutorial Problem for Copy Propagation

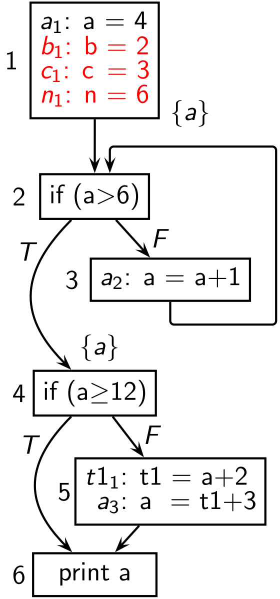


So what is the advantage?

Dead Code Elimination

- Only a is live at the exit of 1

Tutorial Problem for Copy Propagation

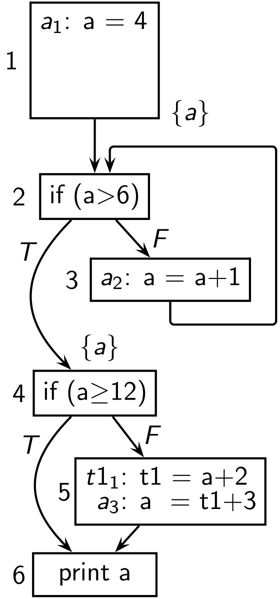


So what is the advantage?

Dead Code Elimination

- Only a is live at the exit of 1
- Assignments of b, c, and n are dead code

Tutorial Problem for Copy Propagation



So what is the advantage?

Dead Code Elimination

- Only a is live at the exit of 1
- Assignments of b, c, and n are dead code
- Can be deleted

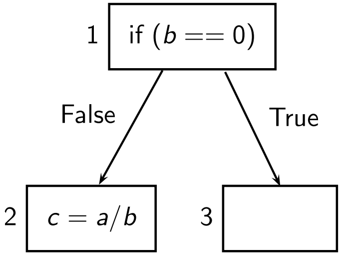
Part 6

Anticipable Expressions Analysis

Defining Anticipable Expressions Analysis

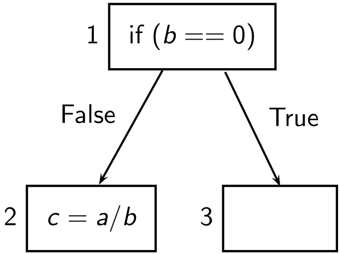
- An expression e is anticipable at a program point p, if every path from p to the program exit contains an evaluation of e which is not preceded by a redefinition of any operand of e.
- Application : Safety of Code Placement

Safety of Code Placement

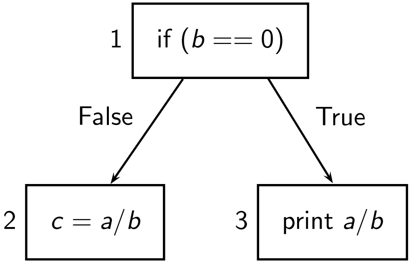


Placing a/b at the exit of 1 is unsafe ($\equiv$ can change the behaviour of the optimized program)

Safety of Code Placement

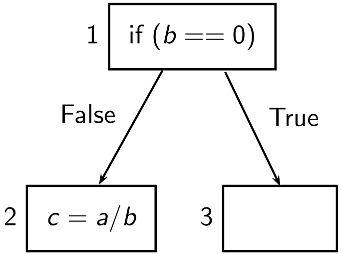


Placing a/b at the exit of 1 is unsafe
($\equiv$ can change the behaviour of the optimized program)

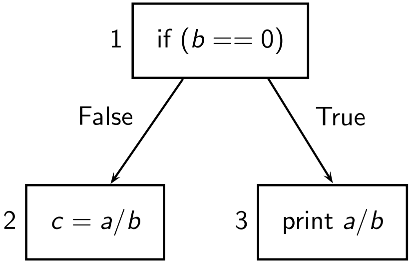


??

Safety of Code Placement



Placing a/b at the exit of 1 is unsafe
($\equiv$ can change the behaviour of the optimized program)



??

A guarded computation of an expression should not be converted to an unguarded computation

Defining Data Flow Analysis for Anticipable Expressions Analysis

$$Gen_n = \{ e \mid \text{expression } e \text{ is evaluated in basic block } n \text{ and this evaluation is not preceded (within } n\text{) by a definition of any operand of } e\}$$

$$Kill_n = \{ e \mid \text{basic block } n \text{ contains a definition of an operand of } e\}$$

	Entity	Manipulation	Exposition
Gen_n	Expression	Use	Upwards
$Kill_n$	Expression	Modification	Anywhere

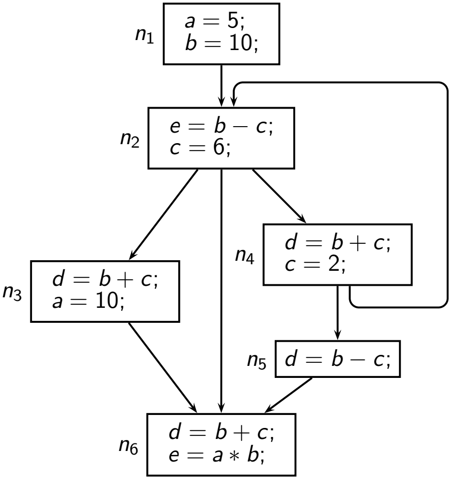
Data Flow Equations for Anticipable Expressions Analysis

$$In_n = Gen_n \cup (Out_n - Kill_n)$$

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

In_n and Out_n are sets of expressions

Tutorial Problem 1 for Anticipable Expressions Analysis



$\mathbb{E}\text{xpr} = \{\ a * b,\ b + c, b - c\ \}$

Solution of Tutorial Problem 1

Block	Local Information		Global Information			
			Iteration # 1		Change in iteration # 2	
	Gen_n	$Kill_n$	Out_n	In_n	Out_n	In_n
n_6	110	000				
n_5	001	000				
n_4	010	011				
n_3	010	100				
n_2	001	011				
n_1	000	111				

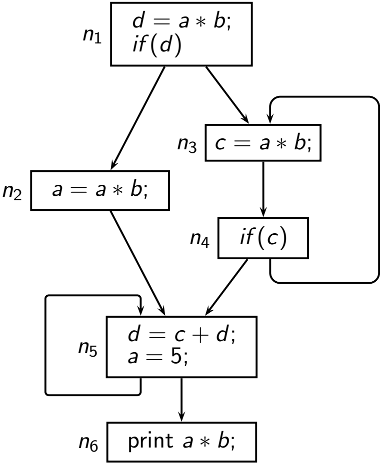
Solution of Tutorial Problem 1

Block	Local Information		Global Information			
			Iteration # 1		Change in iteration # 2	
	Gen_n	$Kill_n$	Out_n	In_n	Out_n	In_n
n_6	110	000	000	110		
n_5	001	000	110	111		
n_4	010	011	111	110		
n_3	010	100	110	010		
n_2	001	011	010	001		
n_1	000	111	001	000		

Solution of Tutorial Problem 1

Block	Local Information		Global Information			
			Iteration # 1		Change in iteration # 2	
	Gen_n	$Kill_n$	Out_n	In_n	Out_n	In_n
n_6	110	000	000	110		
n_5	001	000	110	111		
n_4	010	011	111	110	001	010
n_3	010	100	110	010		
n_2	001	011	010	001		
n_1	000	111	001	000		

Tutorial Problem 2 for Anticipable Expressions Analysis



$\mathbb{E}\text{xpr} = \{\ a * b,\ c + d\ \}$

Solution of Tutorial Problem 2

Block	Local Information		Global Information			
			Iteration # 1		Change in iteration # 2	
	Gen_n	$Kill_n$	Out_n	In_n	Out_n	In_n
n_6	10	00				
n_5	01	11				
n_4	00	00				
n_3	10	01				
n_2	10	10				
n_1	10	01				

Solution of Tutorial Problem 2

Block	Local Information		Global Information			
			Iteration # 1		Change in iteration # 2	
	Gen_n	$Kill_n$	Out_n	In_n	Out_n	In_n
n_6	10	00	00	10		
n_5	01	11	10	01		
n_4	00	00	01	01		
n_3	10	01	01	10		
n_2	10	10	01	11		
n_1	10	01	10	10		

Solution of Tutorial Problem 2

Block	Local Information		Global Information			
			Iteration # 1		Change in iteration # 2	
	Gen_n	$Kill_n$	Out_n	In_n	Out_n	In_n
n_6	10	00	00	10		
n_5	01	11	10	01	00	
n_4	00	00	01	01	00	00
n_3	10	01	01	10	00	
n_2	10	10	01	11		
n_1	10	01	10	10		

Part 7

Common Features of Bit Vector Data Flow Frameworks

Defining Local Data Flow Properties

- Live variables analysis

	Entity	Manipulation	Exposition
Gen_n	Variable	Use	Upwards
$Kill_n$	Variable	Modification	Anywhere

- Analysis of expressions

	Entity	Manipulation	Exposition	
			Availability	Anticipability
Gen_n	Expression	Use	Downwards	Upwards
$Kill_n$	Expression	Modification	Anywhere	Anywhere

Common Form of Data Flow Equations

$$
\begin{aligned}
X_i &= f(Y_i) \\
Y_i &= \bigsqcap X_j
\end{aligned}
$$

Common Form of Data Flow Equations

Data Flow Information

So far we have seen sets (or bit vectors).
Could be entities other than sets.

$$X_i = f(Y_i)$$
$$Y_i = \bigsqcap X_j$$

Common Form of Data Flow Equations

Data Flow Information

So far we have seen sets (or bit vectors). Could be entities other than sets.

Flow Function

$$X_i = f(Y_i)$$
$$Y_i = \prod X_j$$

So far we have seen constant *Gen* and *Kill*. Could be dependent *Gen* and *Kill*.

Common Form of Data Flow Equations

Data Flow Information: So far we have seen sets (or bit vectors). Could be entities other than sets.

Flow Function: So far we have seen constant *Gen* and *Kill*. Could be dependent *Gen* and *Kill*.

$$X_i = f(Y_i)$$

$$Y_i = \bigsqcap X_j$$

Confluence: So far we have seen $\cup$ and $\cap$. Could be other operations.

A Taxonomy of Bit Vector Data Flow Frameworks

	Confluence	
	Union	Intersection
Forward	Reaching Definitions	Available Expressions
Backward	Live Variables	Anticipable Expressions
Bidirectional (limited)		Partial Redundancy Elimination (Original M-R Formulation)

A Taxonomy of Bit Vector Data Flow Frameworks

Any Path → Union

	Confluence	
	Union	Intersection
Forward	Reaching Definitions	Available Expressions
Backward	Live Variables	Anticipable Expressions
Bidirectional (limited)		Partial Redundancy Elimination (Original M-R Formulation)

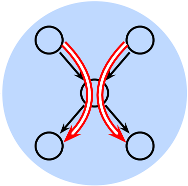
A Taxonomy of Bit Vector Data Flow Frameworks

Any Path → Union

All Paths → Intersection

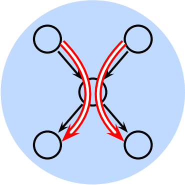
	Confluence	
	Union	Intersection
Forward	Reaching Definitions	Available Expressions
Backward	Live Variables	Anticipable Expressions
Bidirectional (limited)		Partial Redundancy Elimination (Original M-R Formulation)

A Taxonomy of Bit Vector Data Flow Frameworks



	Confluence	
	Union	Intersection
Forward	Reaching Definitions	Available Expressions
Backward	Live Variables	Anticipable Expressions
Bidirectional (limited)		Partial Redundancy Elimination (Original M-R Formulation)

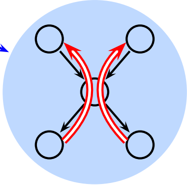
A Taxonomy of Bit Vector Data Flow Frameworks



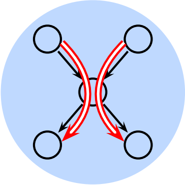
Any Path

All Paths

	Confluence	
	Union	Intersection
Forward	Reaching Definitions	Available Expressions
Backward	Live Variables	Anticipable Expressions
Bidirectional (limited)		Partial Redundancy Elimination (Original M-R Formulation)



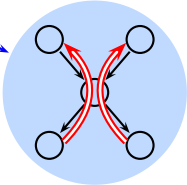
A Taxonomy of Bit Vector Data Flow Frameworks

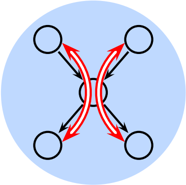


Any Path

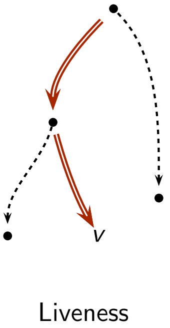
All Paths

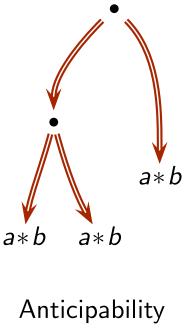
	Confluence	
	Union	Intersection
Forward	Reaching Definitions	Available Expressions
Backward	Live Variables	Anticipable Expressions
Bidirectional (limited)		Partial Redundancy Elimination (Original M-R Formulation)

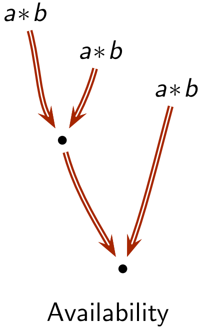


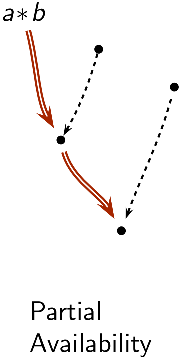


Data Flow Paths Discovered by Data Flow Analysis

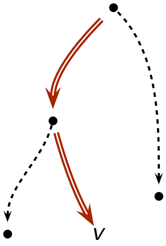








Data Flow Paths Discovered by Data Flow Analysis

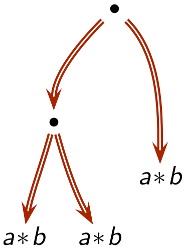


Liveness

Sequence of blocks $(n_1, n_2, \ldots, n_k)$ which is a prefix of some potential execution path starting at n_1 such that:

- n_k contains an upwards exposed use of v, **and**
- no other block on the path contains an assignment to v.

Data Flow Paths Discovered by Data Flow Analysis



Anticipability

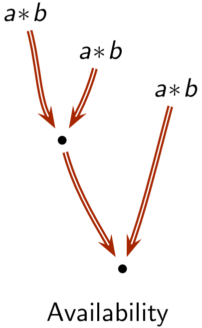
Sequence of blocks $(n_1, n_2, \ldots, n_k)$ which is a prefix of some potential execution path starting at n_1 such that:

- n_k contains an upwards exposed use of $a * b$, **and**
- no other block on the path contains an assignment to a or b, **and**
- every path starting at n_1 is an anticipability path of $a * b$.

Data Flow Paths Discovered by Data Flow Analysis

Sequence of blocks $(n_1, n_2, \ldots, n_k)$ which is a prefix of some potential execution path starting at n_1 such that:

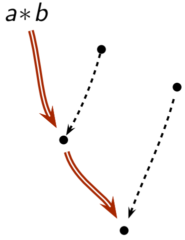
- n_1 contains a downwards exposed use of $a * b$, **and**
- no other block on the path contains an assignment to a or b, **and**
- every path ending at n_k is an availability path of $a * b$.



Data Flow Paths Discovered by Data Flow Analysis

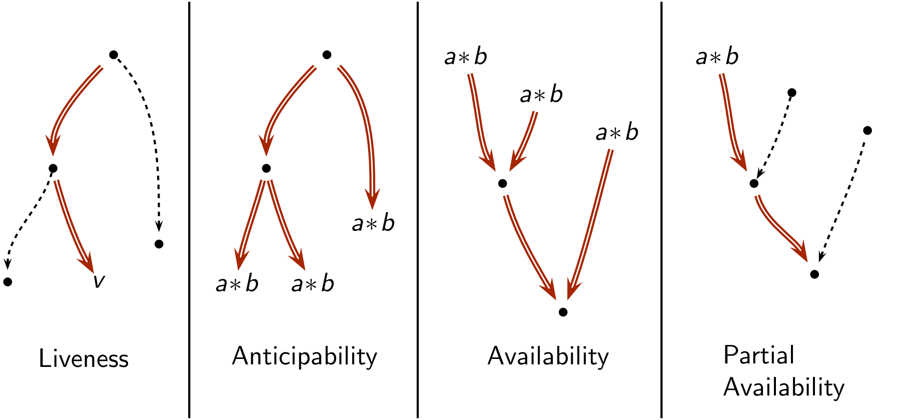
Sequence of blocks $(n_1, n_2, \ldots, n_k)$ which is a prefix of some potential execution path starting at n_1 such that:

- n_1 contains a downwards exposed use of $a * b$, **and**
- no other block on the path contains an assignment to a or b.



Partial Availability

Data Flow Paths Discovered by Data Flow Analysis



Part 9

Partial Redundancy Elimination

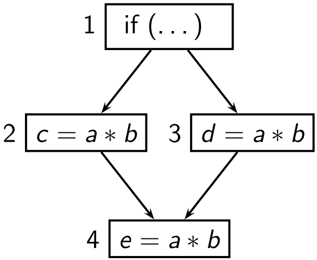
Precursor: Common Subexpression Elimination

Code Fragment	Flow Graph	Remarks

`if (...)`
`    c = a*b;`
`else`
`    d = a*b;`
`e = a*b;`		

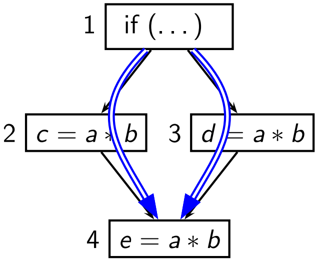
Precursor: Common Subexpression Elimination

Code Fragment	Flow Graph	Remarks

`if (...)`
`    c = a*b;`
`else`
`    d = a*b;`
`e = a*b;`
		• a and b are not modified along paths $1 \rightarrow 2 \rightarrow 4$ and $1 \rightarrow 3 \rightarrow 4$

Precursor: Common Subexpression Elimination

Code Fragment	Flow Graph	Remarks

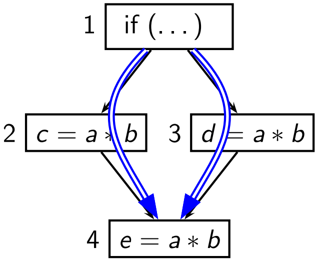
`if (...)`
`    c = a*b;`
`else`
`    d = a*b;`
`e = a*b;`
		• a and b are not modified along paths $1 \rightarrow 2 \rightarrow 4$ and $1 \rightarrow 3 \rightarrow 4$
		• Computation of $a * b$ in 4 is redundant

Precursor: Common Subexpression Elimination

Code Fragment

```
if (...)
    c = a*b;
else
    d = a*b;
e = a*b;
```

Flow Graph



Remarks

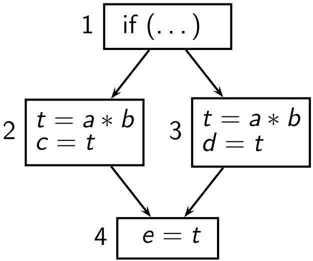
- a and b are not modified along paths $1 \rightarrow 2 \rightarrow 4$ and $1 \rightarrow 3 \rightarrow 4$
- Computation of $a * b$ in 4 is redundant
- Previous value can be used

Precursor: Common Subexpression Elimination

Code Fragment

```
if (...)
    c = a*b;
else
    d = a*b;
e = a*b;
```

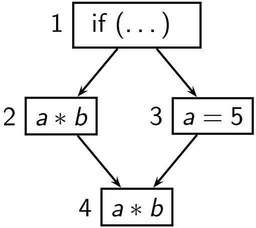
Flow Graph



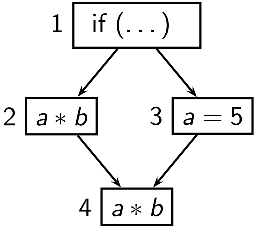
Remarks

- a and b are not modified along paths $1 \rightarrow 2 \rightarrow 4$ and $1 \rightarrow 3 \rightarrow 4$
- Computation of $a * b$ in 4 is redundant
- Previous value can be used

Partial Redundancy Elimination

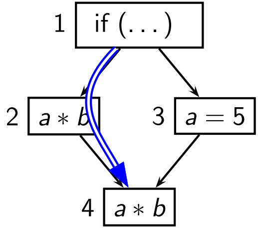


Partial Redundancy Elimination



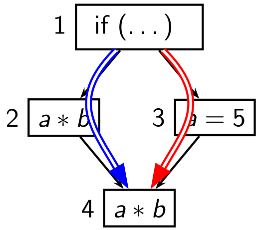
- Computation of $a * b$ in 4 is

Partial Redundancy Elimination



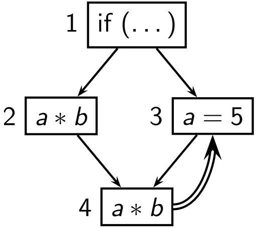
- Computation of $a * b$ in 4 is
  - redundant along path $1 \rightarrow 2 \rightarrow 4$, but ...

Partial Redundancy Elimination

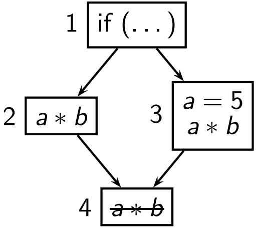


- Computation of $a * b$ in 4 is
  - redundant along path $1 \rightarrow 2 \rightarrow 4$, but . . .
  - not redundant along path $1 \rightarrow 3 \rightarrow 4$

Code Hoisting for Partial Redundancy Elimination



Code Hoisting for Partial Redundancy Elimination

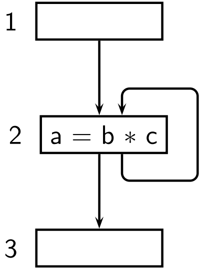


- Computation of $a * b$ in 3 becomes totally redundant
- Can be deleted

PRE Subsumes Loop Invariant Movement

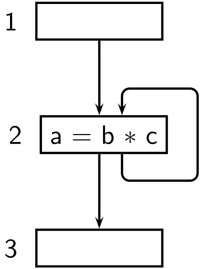
PRE Subsumes Loop Invariant Movement

What's that?



PRE Subsumes Loop Invariant Movement

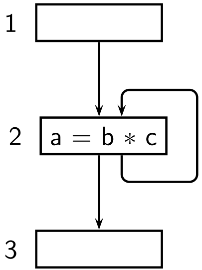
What's that?



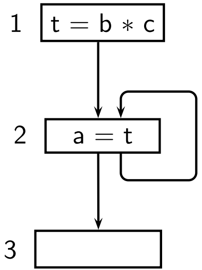
Translate to

PRE Subsumes Loop Invariant Movement

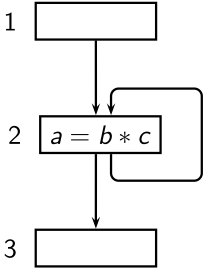
What's that?



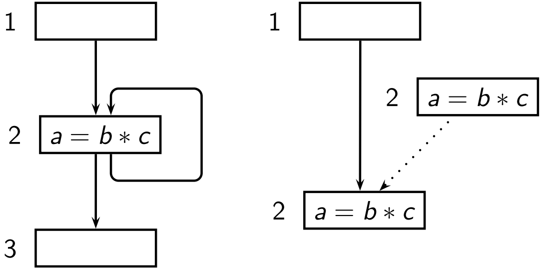
Translate to



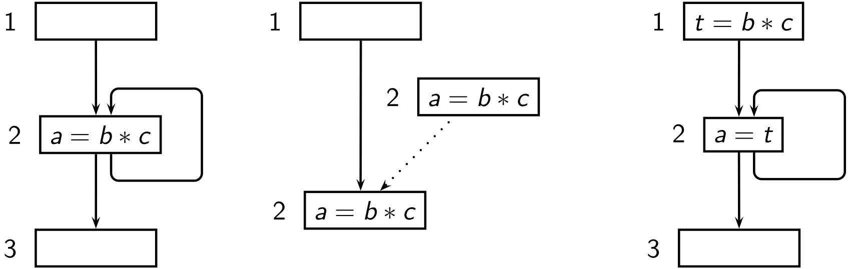
PRE Subsumes Loop Invariant Movement



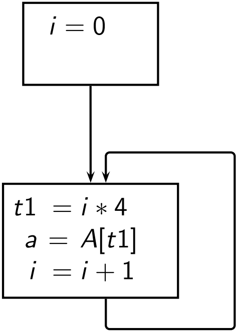
PRE Subsumes Loop Invariant Movement



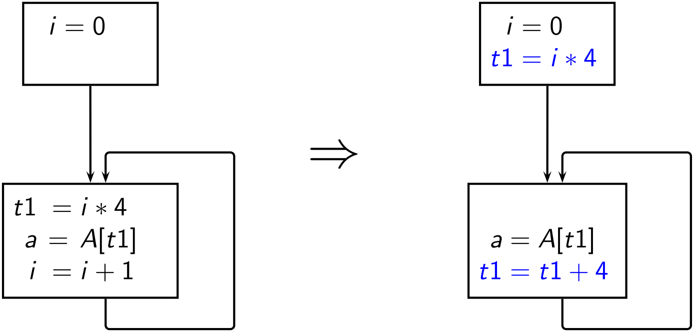
PRE Subsumes Loop Invariant Movement



PRE Can be Used for Strength Reduction

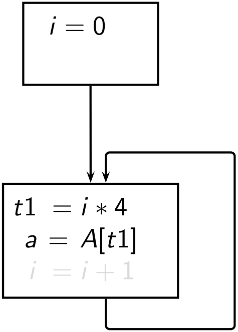


PRE Can be Used for Strength Reduction



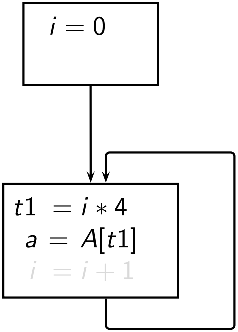
- $*$ in the loop has been replaced by $+$
- $i = i + 1$ in the loop has been eliminated

PRE Can be Used for Strength Reduction



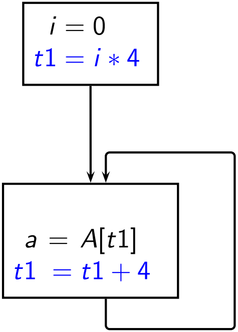
- Delete $i = i + 1$

PRE Can be Used for Strength Reduction



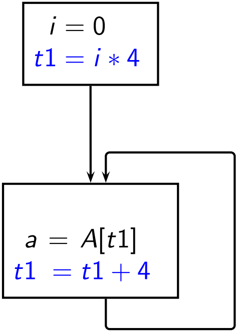
- Delete $i = i + 1$
- Expression $i * 4$ becomes loop invariant

PRE Can be Used for Strength Reduction



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

PRE Can be Used for Strength Reduction



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

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

Performing Partial Redundancy Elimination

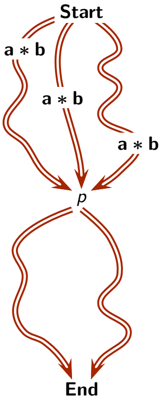
1. Identify partial redundancies
2. Identify program points where computations can be inserted
3. Insert expressions
4. Partial redundancies become total redundancies
   $\Longrightarrow$ Delete them.

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

Defining Hoisting Criteria

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

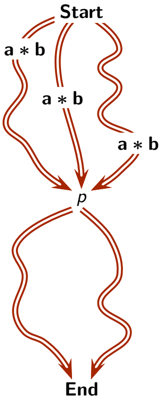
Available at *p*



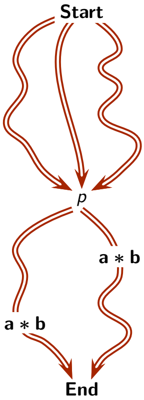
Defining Hoisting Criteria

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

Available at *p*

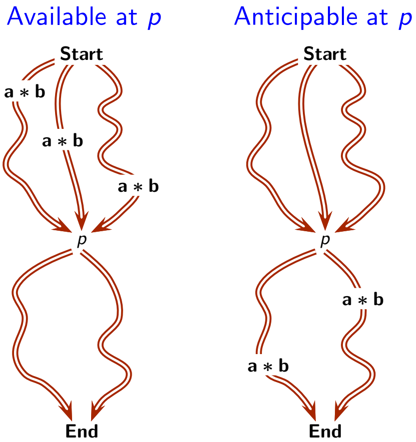


Anticipable at *p*



Defining Hoisting Criteria

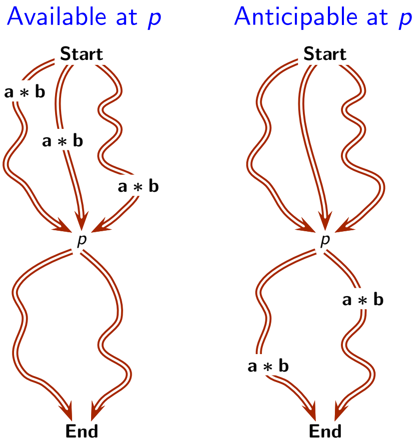
- An expression can be safely inserted at a program point *p* if it is



- 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

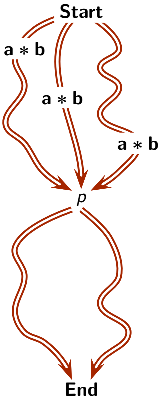


  - If it is available at *p*, then there is no need to insert it at *p*.
  - If it is anticipable at *p* then all such occurrences should be hoisted to *p*.

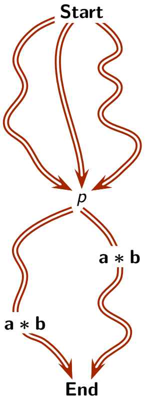
Defining Hoisting Criteria

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

Available at p

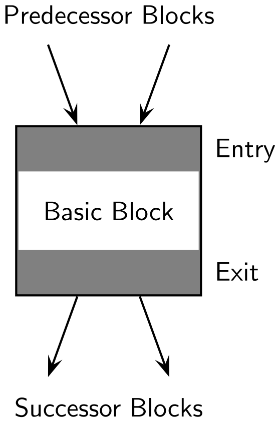


Anticipable at p

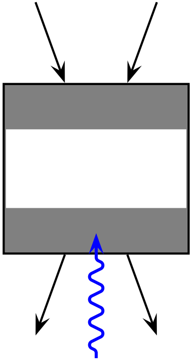


- If it is available at p, then there is no need to insert it at p.
- If it is anticipable at p then all such occurrences should be hoisted to p.
- *An expression should be hoisted to p provided it can be hoisted to p along all paths from p to exit.*

Safety of Hoisting an Expression

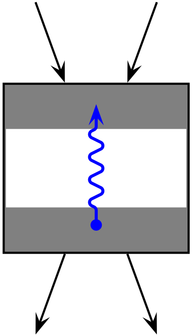


Safety of Hoisting an Expression



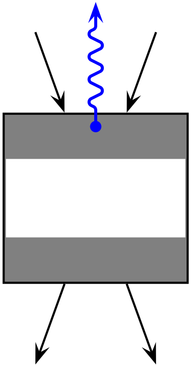
- *Safety of hoisting to the exit of a block*

Safety of Hoisting an Expression



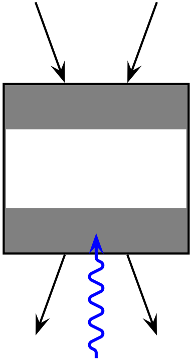
- *Safety of hoisting to the exit of a block*
- *Safety of hoisting to the entry of a block*

Safety of Hoisting an Expression



- *Safety of hoisting to the exit of a block*
- *Safety of hoisting to the entry of a block*
- *Safety of hoisting out of the entry of a block*

Safety of Hoisting an Expression



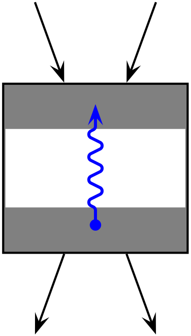
- *Safety of hoisting to the exit of a block*

  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks

- *Safety of hoisting to the entry of a block*

- *Safety of hoisting out of the entry of a block*

Safety of Hoisting an Expression



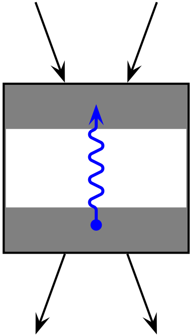
- *Safety of hoisting to the exit of a block*
- *Safety of hoisting to the entry of a block*

  S.2 Hoist only if

    S.2.a it is upwards exposed, or

- *Safety of hoisting out of the entry of a block*

Safety of Hoisting an Expression



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

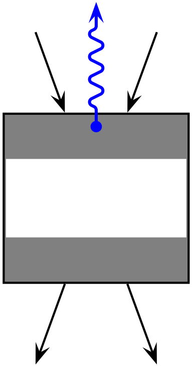
- *Safety of hoisting to the entry of a block*

  S.2 Hoist only if

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

- *Safety of hoisting out of the entry of a block*

Safety of Hoisting an Expression

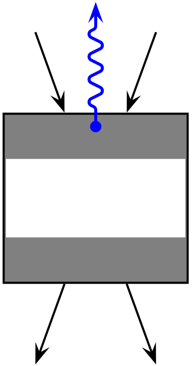


- *Safety of hoisting to the exit of a block*
- *Safety of hoisting to the entry of a block*
- *Safety of hoisting out of the entry of a block*

  S.3 Hoist only if for each predecessor

  S.3.a it can be hoisted to its exit, or

Safety of Hoisting an Expression

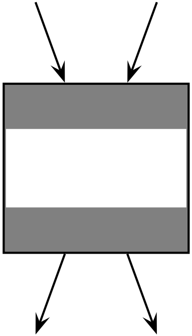


- *Safety of hoisting to the exit of a block*
- *Safety of hoisting to the entry of a block*
- *Safety of hoisting out of the entry of a block*

  S.3 Hoist only if for each predecessor

  S.3.a it can be hoisted to its exit, or
  S.3.b it is available at its exit.

Safety of Hoisting an Expression



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

  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks

- *Safety of hoisting to the entry of a block*

  S.2 Hoist only if

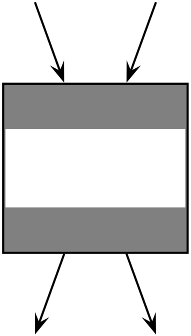
  - S.2.a it is upwards exposed, or
  - S.2.b it can be hoisted to its exit and is transparent in the block

- *Safety of hoisting out of the entry of a block*

  S.3 Hoist only if for each predecessor

  - S.3.a it can be hoisted to its exit, or
  - S.3.b it is available at its exit.

Safety of Hoisting an Expression



Anticipability

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

  S.1 Hoist only if it can ... ed out of the entries of all suc ... ocks

- *Safety of hoisti ... entry of a block*

  S.2 Hoist ...

  S.2.a ... ards exposed, or

  S.2.b it ... be hoisted to its exit and is transparent in the block

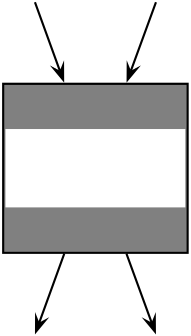
- *Safety of hoisting out of the entry of a block*

  S.3 Hoist only if for each predecessor

  S.3.a it can be hoisted to its exit, or

  S.3.b it is available at its exit.

Safety of Hoisting an Expression



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

  S.1 Hoist only if it can ... ed out of the entries of all suc ... ocks

- *Safety of hoisting ... entry of a block*

  S.2 Hoist ...

  S.2.a ... ards exposed, or

  S.2.b it ... be hoisted to its exit and is transparent in the block

- *Safety of hoisting out of the entry of a block*

  S.3 Hoist only if f ... h predecessor

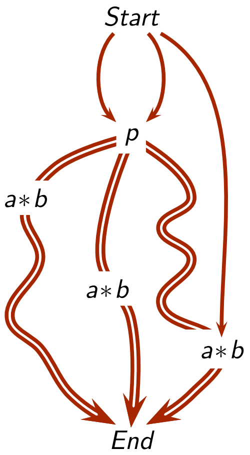
  S.3.a it can be ... ed to its exit, or

  S.3.b it is available at its exit.

Anticipability

??

Anticipability and Code Hoisting

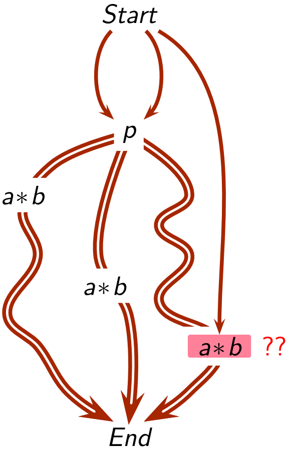


- What is the meaning of the assertion

  "$a * b$ is anticipable at program point p"

  - $a * b$ is computed along every path from p to *End* before a or b are modified
  - The value computed at p would be same as the next value computed on any path
  - $a * b$ can be safely inserted at p

Anticipability and Code Hoisting



- What is the meaning of the assertion

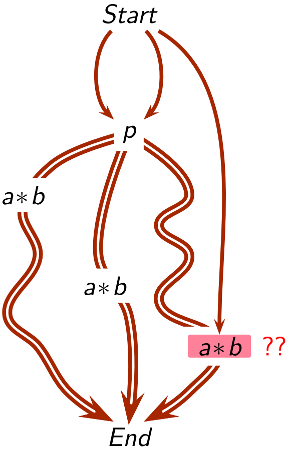
  "$a * b$ is anticipable at program point p"

  - $a * b$ is computed along every path from p to *End* before a or b are modified
  - The value computed at p would be same as the next value computed on any path
  - $a * b$ can be safely inserted at p

- It does not say that the subsequent computations of $a * b$ can be deleted

  (Expression may not be available at the subsequent points)

Anticipability and Code Hoisting



- What is the meaning of the assertion

  "$a * b$ is anticipable at program point p"

  - $a * b$ is computed along every path from p to *End* before a or b are modified
  - The value computed at p would be same as the next value computed on any path
  - $a * b$ can be safely inserted at p

- It does not say that the subsequent computations of $a * b$ can be deleted

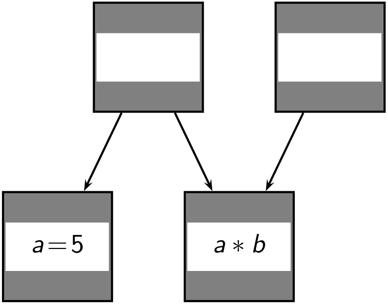
  (Expression may not be available at the subsequent points)

- Hoisting involves

  - making the expressions available and
  - deleting their subsequent computations

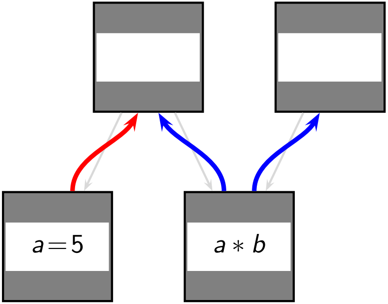
A Comparison of Anticipability and Hoistability

Anticipability	Hoistability



A Comparison of Anticipability and Hoistability

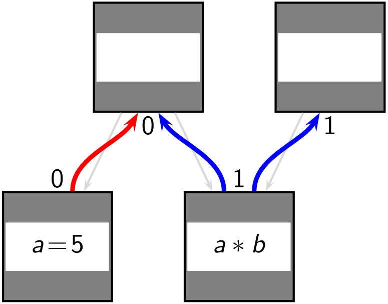
Anticipability



Hoistability

A Comparison of Anticipability and Hoistability

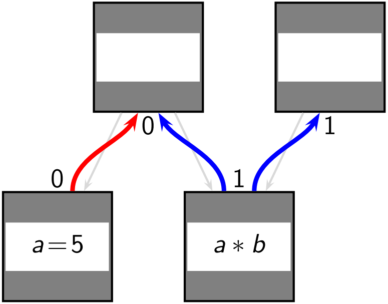
Anticipability



Hoistability

A Comparison of Anticipability and Hoistability

Anticipability

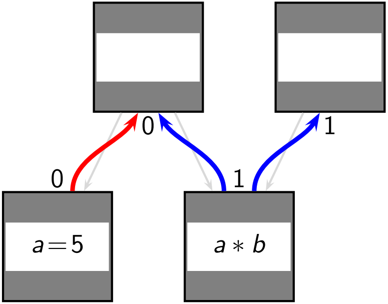


Characterises safety of placement but not safety of hoisting

Hoistability

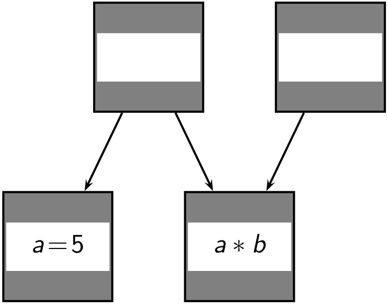
A Comparison of Anticipability and Hoistability

Anticipability



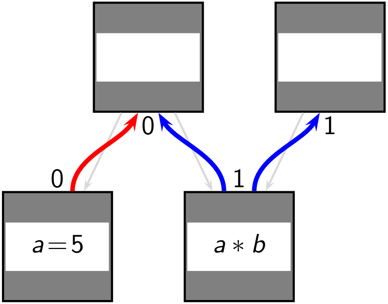
Characterises safety of placement but not safety of hoisting

Hoistability



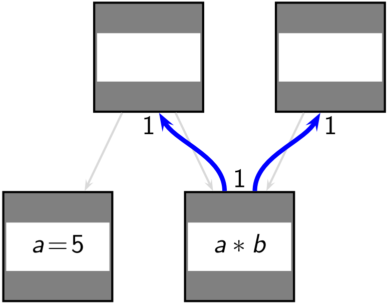
A Comparison of Anticipability and Hoistability

Anticipability



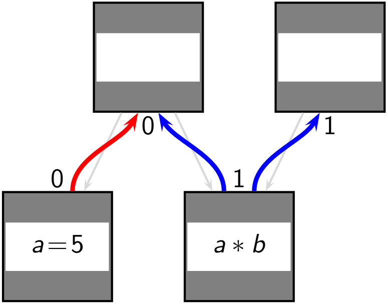
Characterises safety of placement but not safety of hoisting

Hoistability



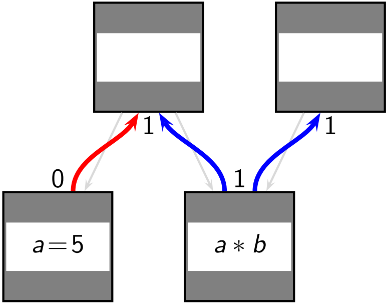
A Comparison of Anticipability and Hoistability

Anticipability

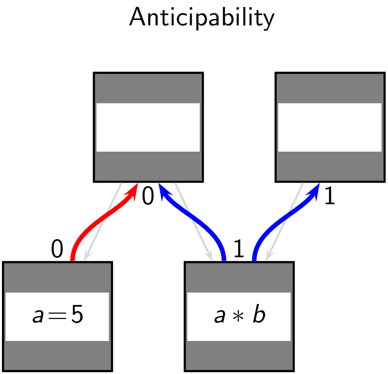


Characterises safety of placement but not safety of hoisting

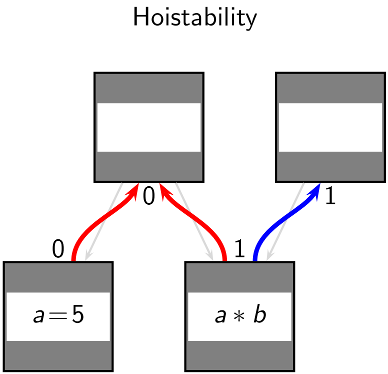
Hoistability



A Comparison of Anticipability and Hoistability

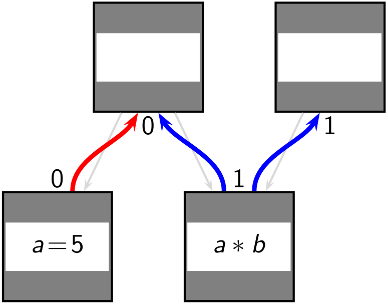


Characterises safety of placement but not safety of hoisting



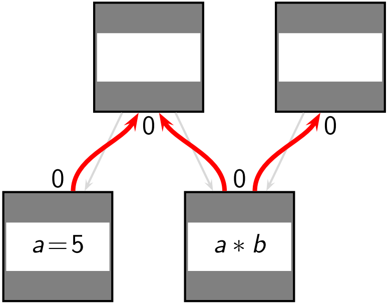
A Comparison of Anticipability and Hoistability

Anticipability



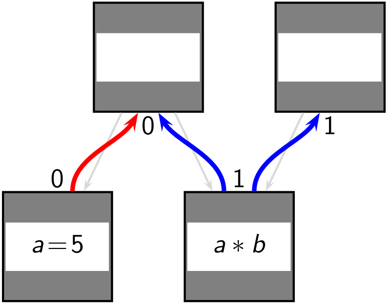
Characterises safety of placement but not safety of hoisting

Hoistability



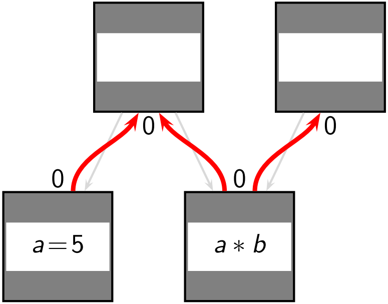
A Comparison of Anticipability and Hoistability

Anticipability



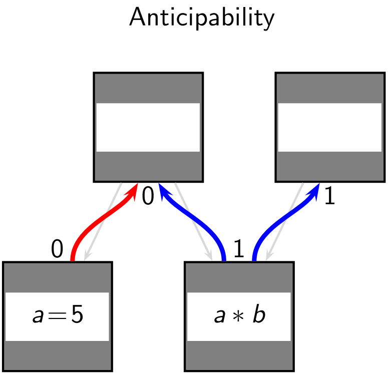
Characterises safety of placement but not safety of hoisting

Hoistability

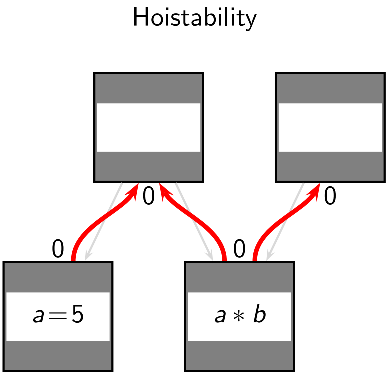


Characterises safety of hoisting

A Comparison of Anticipability and Hoistability



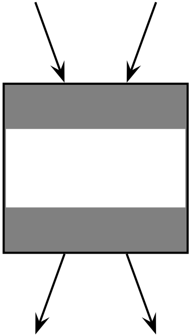
Characterises safety of placement but not safety of hoisting



Characterises safety of hoisting

Hoist an expression to the entry of a block only if it can be hoisted out of the block into all predecessor blocks

Revised Safety Criteria of Hoisting an Expression



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

  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks

- *Safety of hoisting to the entry of a block*

  S.2 Hoist only if

  S.2.a it is upwards exposed, or

  S.2.b it can be hoisted to its exit and is transparent in the block

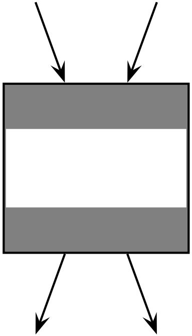
- *Safety of hoisting out of the entry of a block*

  S.3 Hoist only if for each predecessor

  S.3.a it can be hoisted to its exit, or

  S.3.b it is available at its exit.

Revised Safety Criteria of Hoisting an Expression



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

  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks

- *Safety of hoisting to the entry of a block*

  S.2 Hoist only if

  S.2.a it is upwards exposed, or

  S.2.b it can be hoisted to its exit and is transparent in the block

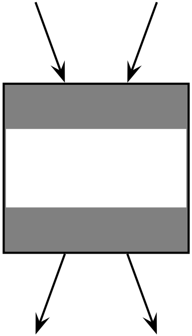
- *Safety of hoisting out of the entry of a block*

  S.3 Hoist only if for each predecessor

  S.3.a it can be hoisted to its exit, or

  S.3.b it is available at its exit.

Revised Safety Criteria of Hoisting an Expression



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

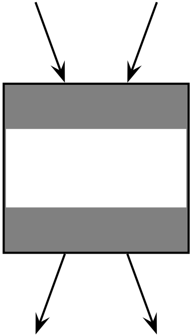
  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks

- *Safety of hoisting to the entry of a block*

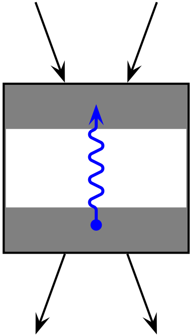
  S.2 Hoist only if
  - S.2.a it is upwards exposed, or
  - S.2.b it can be hoisted to its exit and is transparent in the block

  S.3 Hoist only if for each predecessor
  - S.3.a it can be hoisted to its exit, or
  - S.3.b it is available at its exit.

Desirability of Hoisting an Expression

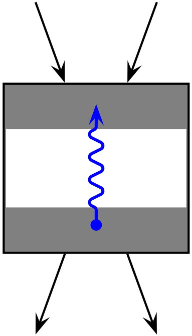


Desirability of Hoisting an Expression



- *Desirability of hoisting to the entry of a block*

Desirability of Hoisting an Expression



- *Desirability of hoisting to the entry of a block*

  D.1 Hoist only if it is partially available

Final Hoisting Criteria

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

  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks

- *Safety of hoisting to the entry of a block*

  S.2 Hoist only if
    - S.2.a it is upwards exposed, or
    - S.2.b it can be hoisted to its exit and is transparent in the block

  S.3 Hoist only if for each predecessor
    - S.3.a it can be hoisted to its exit, or
    - S.3.b it is available at its exit.

- *Desirability of hoisting to the entry of a block*

  D.1 Hoist only if it is partially available

From Hoisting Criteria to Data Flow Equations (1)

First Level Global Data Flow Properties in PRE

- Partial Availability.

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

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

- Total Availability.

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

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

From Hoisting Criteria to Data Flow Equations (2)

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

  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks

- *Safety of hoisting to the entry of a block*

  S.2 Hoist only if
  - S.2.a it is upwards exposed, or
  - S.2.b it can be hoisted to its exit and is transparent in the block

  S.3 Hoist only if for each predecessor
  - S.3.a it can be hoisted to its exit, or
  - S.3.b it is available at its exit.

- *Desirability of hoisting to the entry of a block*

  D.1 Hoist only if it is partially available

From Hoisting Criteria to Data Flow Equations (2)

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

  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks

- *Safety of hoisting to the entry of a block*

  S.2 Hoist only if
  - S.2.a it is upwards exposed, or
  - S.2.b it can be hoisted to its exit and is transparent in the block

  S.3 Hoist only if for each predecessor
  - S.3.a it can be hoisted to its exit, or
  - S.3.b it is available at its exit.

- *Desirability of hoisting to the entry of a block*

  D.1 Hoist only if it is partially available

From Hoisting Criteria to Data Flow Equations (2)

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

  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks

$$\forall s \in succ(n), \quad Out_n \subseteq In_s$$

- *Safety of hoisting to the entry of a block*

  S.2 Hoist only if

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

  S.3 Hoist only if for each predecessor

  - S.3.a it can be hoisted to its exit, or
  - S.3.b it is available at its exit.

- *Desirability of hoisting to the entry of a block*

  D.1 Hoist only if it is partially available

From Hoisting Criteria to Data Flow Equations (2)

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

  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks

$$\forall s \in succ(n),\quad Out_n \subseteq In_s$$

- *Safety of hoisting to the entry of a block*

  S.2 Hoist only if
  - S.2.a it is upwards exposed, or
  - S.2.b it can be hoisted to its exit and is transparent in the block

  S.3 Hoist only if for each predecessor
  - S.3.a it can be hoisted to its exit, or
  - S.3.b it is available at its exit.

$$In_n \subseteq AntGen_n \cup (Out_n - Kill_n)$$

- *Desirability of hoisting to the entry of a block*

  D.1 Hoist only if it is partially available

From Hoisting Criteria to Data Flow Equations (2)

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

  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks

$\forall s \in succ(n),\ Out_n \subseteq In_s$

- *Safety of hoisting to the entry of a block*

  S.2 Hoist only if
  - S.2.a it is upwards exposed, or
  - S.2.b it can be hoisted to its exit and is transparent in the block

  S.3 Hoist only if for each predecessor
  - S.3.a it can be hoisted to its exit, or
  - S.3.b it is available at its exit.

$In_n \subseteq AntGen_n \cup (Out_n - Kill_n)$

$\forall p \in pred(n),\ In_n \subseteq AvOut_p \cup Out_p$

- *Desirability of hoisting to the entry of a block*

  D.1 Hoist only if it is partially available

From Hoisting Criteria to Data Flow Equations (2)

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

  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks

  $\forall s \in succ(n),\ Out_n \subseteq In_s$

- *Safety of hoisting to the entry of a block*

  S.2 Hoist only if
  - S.2.a it is upwards exposed, or
  - S.2.b it can be hoisted to its exit and is transparent in the block

  $In_n \subseteq AntGen_n \cup (Out_n - Kill_n)$

  S.3 Hoist only if for each predecessor
  - S.3.a it can be hoisted to its exit, or
  - S.3.b it is available at its exit.

  $\forall p \in pred(n),\ In_n \subseteq AvOut_p \cup Out_p$

- *Desirability of hoisting to the entry of a block*

  D.1 Hoist only if it is partially available

  $In_n \subseteq PavIn_n$

From Hoisting Criteria to Data Flow Equations (2)

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

  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks

$\forall s \in succ(n),$
$Out_n \subseteq In_s$

- *Safety of hoisting to the entry of a block*

  S.2 Hoist only if
  - S.2.a it is upwards exposed, or
  - S.2.b it can be hoisted to its exit and is transparent in the block

  S.3 Hoist only if for each predecessor
  - S.3.a it can be hoisted to its exit, or
  - S.3.b it is available at its exit.

$In_n \subseteq AntGen_n \cup (Out_n - Kill_n)$

$\forall p \in pred(n),$
$In_n \subseteq AvOut_p \cup Out_p$

- *Desirability of hoisting to the entry of a block*

  D.1 Hoist only if it is partially available

$In_n \subseteq PavIn_n$

From Hoisting Criteria to Data Flow Equations (3)

$\forall s \in succ(n),$
$Out_n \subseteq In_s$

$In_n \subseteq AntGen_n \cup$
$(Out_n - Kill_n)$

$\forall p \in pred(n),$
$In_n \subseteq AvOut_p \cup$
Out_p

$In_n \subseteq PavIn_n$

From Hoisting Criteria to Data Flow Equations (3)

$\forall s \in succ(n)$,
$Out_n \subseteq In_s$

$In_n \subseteq AntGen_n \cup (Out_n - Kill_n)$

Find out the largest such set

$\forall p \in pred(n)$,
$In_n \subseteq AvOut_p \cup Out_p$

$In_n \subseteq PavIn_n$

From Hoisting Criteria to Data Flow Equations (3)

$\forall s \in succ(n),$
$Out_n \subseteq In_s$

$In_n \subseteq AntGen_n \cup (Out_n - Kill_n)$

$\forall p \in pred(n),$
$In_n \subseteq AvOut_p \cup Out_p$

$In_n \subseteq PavIn_n$

Desirability: D.1

$In_n = PavIn_n$

Expressions should be partially available, and

From Hoisting Criteria to Data Flow Equations (3)

Safety: S.2.a

$\forall s \in succ(n)$,
$Out_n \subseteq In_s$

$In_n \subseteq AntGen_n \cup (Out_n - Kill_n)$

$\forall p \in pred(n)$,
$In_n \subseteq AvOut_p \cup Out_p$

$In_n \subseteq PavIn_n$

$$In_n = PavIn_n \cap \Big(AntGen_n \cup \qquad \Big)$$

Expressions should be upwards exposed, or

From Hoisting Criteria to Data Flow Equations (3)

$\forall s \in succ(n),$
$\quad Out_n \subseteq In_s$

$In_n \subseteq AntGen_n \cup (Out_n - Kill_n)$

$\forall p \in pred(n),$
$\quad In_n \subseteq AvOut_p \cup Out_p$

$In_n \subseteq PavIn_n$

Safety: S.2.b

$$In_n = PavIn_n \cap \left(AntGen_n \cup (Out_n - Kill_n) \right)$$

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

From Hoisting Criteria to Data Flow Equations (3)

$\forall s \in succ(n),$
$Out_n \subseteq In_s$

$In_n \subseteq AntGen_n \cup (Out_n - Kill_n)$

$\forall p \in pred(n),$
$In_n \subseteq AvOut_p \cup Out_p$

$In_n \subseteq PavIn_n$

Safety: S.3.b

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

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

From Hoisting Criteria to Data Flow Equations (3)

$\forall s \in succ(n),$
$Out_n \subseteq In_s$

$In_n \subseteq AntGen_n \cup$
$(Out_n - Kill_n)$

$\forall p \in pred(n),$
$In_n \subseteq AvOut_p \cup$
Out_p

$In_n \subseteq PavIn_n$

Safety: S.3.a

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

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

From Hoisting Criteria to Data Flow Equations (3)

$\forall s \in succ(n),$
$Out_n \subseteq In_s$

$In_n \subseteq AntGen_n \cup (Out_n - Kill_n)$

$\forall p \in pred(n),$
$In_n \subseteq AvOut_p \cup Out_p$

$In_n \subseteq PavIn_n$

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

$$Out_n = \begin{cases} BI & n \text{ is } End \text{ block} \\ & \text{otherwise} \end{cases}$$

Boundary condition

From Hoisting Criteria to Data Flow Equations (3)

$\forall s \in succ(n),\ Out_n \subseteq In_s$

Safety: S.1

$In_n \subseteq AntGen_n \cup (Out_n - Kill_n)$

$\forall p \in pred(n),\ In_n \subseteq AvOut_p \cup Out_p$

$In_n \subseteq PavIn_n$

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

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

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

From Hoisting Criteria to Data Flow Equations (3)

$\forall s \in succ(n),$
$Out_n \subseteq In_s$

$In_n \subseteq AntGen_n \cup (Out_n - Kill_n)$

$\forall p \in pred(n),$
$In_n \subseteq AvOut_p \cup Out_p$

$In_n \subseteq PavIn_n$

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

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

Anticipability and PRE (Hoistability) Data Flow Equations

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

Anticipability and PRE (Hoistability) Data Flow Equations

PRE Hoistability	Anticipability
$In_n = PavIn_n \cap (AntGen_n \cup (Out_n - Kill_n)) \bigcap\limits_{p \in pred(n)} (Out_p \cup AvOut_p)$	$In_n = AntGen_n \cup (Out_n - Kill_n)$
$Out_n = \begin{cases} BI & n \text{ is } End \text{ block} \\ \bigcap\limits_{s \in succ(n)} In_s & \text{otherwise} \end{cases}$	$Out_n = \begin{cases} BI & n \text{ is } End \text{ block} \\ \bigcap\limits_{s \in succ(n)} In_s & \text{otherwise} \end{cases}$

Anticipability and PRE (Hoistability) Data Flow Equations

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

Anticipability and PRE (Hoistability) Data Flow Equations

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

PRE Hoistability is anticipability restricted by

Anticipability and PRE (Hoistability) Data Flow Equations

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

PRE Hoistability is anticipability restricted by

- *safety of hoisting and*
- *partial availability*

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.

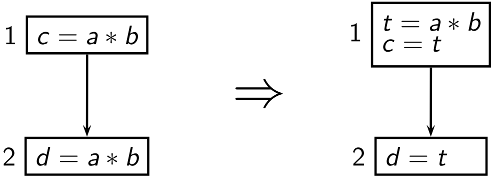
Insertion Criteria in PRE

- 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*.

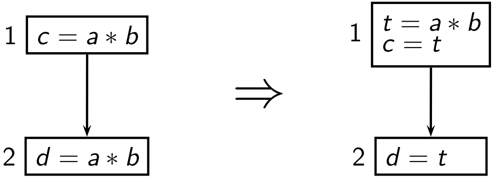
$$Insert_n = Out_n \cap (\neg AvOut_n) \cap (\neg In_n \cup Kill_n)$$

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

Performing PRE by Computing *In*/*Out*: Simple Cases (1)

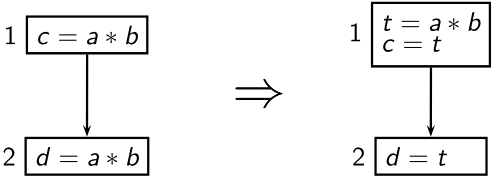


Performing PRE by Computing *In*/*Out*: Simple Cases (1)



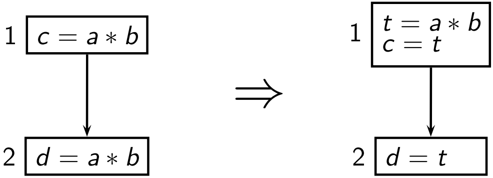
Node	First Level Values				Init.		Iter. 1		Iter. 2		Redund.	Insert
	AntGen	*Kill*	*PavIn*	*AvOut*	*Out*	*In*	*Out*	*In*	*Out*	*In*		
2												
1												

Performing PRE by Computing *In*/*Out*: Simple Cases (1)



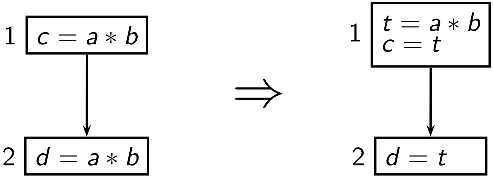
Node	First Level Values				Init.		Iter. 1		Iter. 2		Redund.	Insert
	AntGen	*Kill*	*PavIn*	*AvOut*	*Out*	*In*	*Out*	*In*	*Out*	*In*		
2	1	0	1	1								
1	1	0	0	1								

Performing PRE by Computing *In*/*Out*: Simple Cases (1)



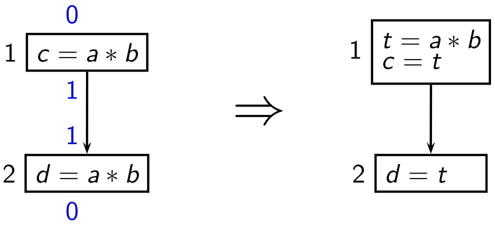
Node	First Level Values				Init.		Iter. 1		Iter. 2		Redund.	Insert
	AntGen	*Kill*	*PavIn*	*AvOut*	*Out*	*In*	*Out*	*In*	*Out*	*In*		
2	1	0	1	1	0	1						
1	1	0	0	1	1	1						

Performing PRE by Computing *In*/*Out*: Simple Cases (1)



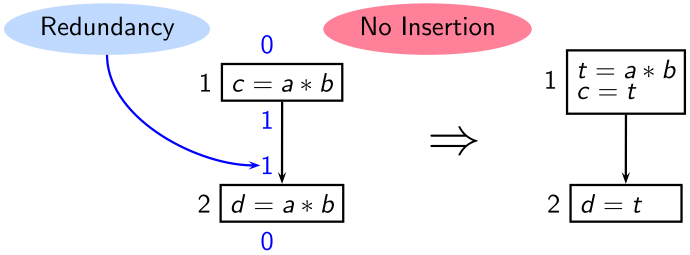
Node	First Level Values				Init.		Iter. 1		Iter. 2		Redund.	Insert
	AntGen	*Kill*	*PavIn*	*AvOut*	*Out*	*In*	*Out*	*In*	*Out*	*In*		
2	1	0	1	1	0	1	0	1				
1	1	0	0	1	1	1	1	0				

Performing PRE by Computing *In*/*Out*: Simple Cases (1)



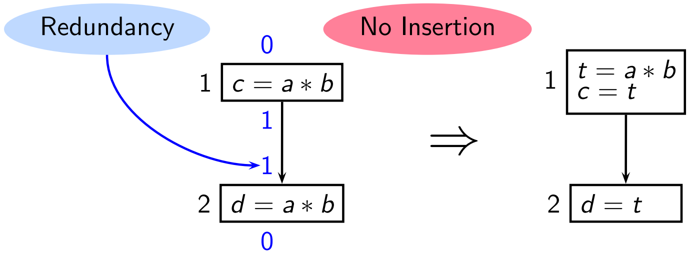
Node	First Level Values				Init.		Iter. 1		Iter. 2		Redund.	Insert
	AntGen	*Kill*	*PavIn*	*AvOut*	*Out*	*In*	*Out*	*In*	*Out*	*In*		
2	1	0	1	1	0	1	0	1	0	1		
1	1	0	0	1	1	1	1	0	1	0		

Performing PRE by Computing *In*/*Out*: Simple Cases (1)



Node	First Level Values				Init.		Iter. 1		Iter. 2		Redund.	Insert
	AntGen	*Kill*	*PavIn*	*AvOut*	*Out*	*In*	*Out*	*In*	*Out*	*In*		
2	1	0	1	1	0	1	0	1	0	1	1	0
1	1	0	0	1	1	1	1	0	1	0	0	0

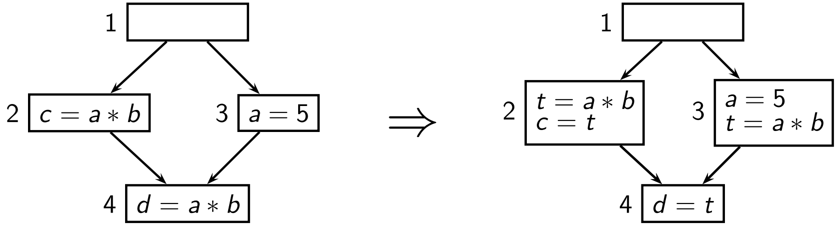
Performing PRE by Computing *In*/*Out*: Simple Cases (1)



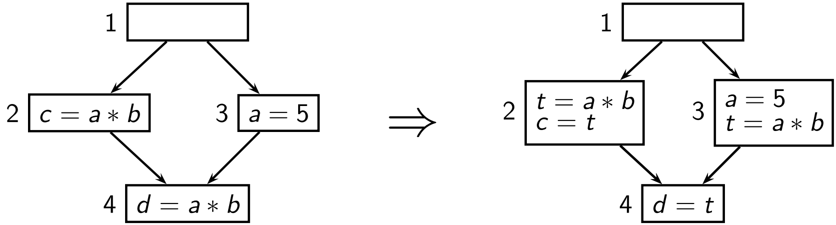
Node	First Level Values				Init.		Iter. 1		Iter. 2		Redund.	Insert
	AntGen	*Kill*	*PavIn*	*AvOut*	*Out*	*In*	*Out*	*In*	*Out*	*In*		
2	1	0	1	1	0	1	0	1	0	1	1	0
1	1	0	0	1	1	1	1	0	1	0	0	0

This is an instance of Common Subexpression Elimination

Performing PRE by Computing *In*/*Out*: Simple Cases (2)

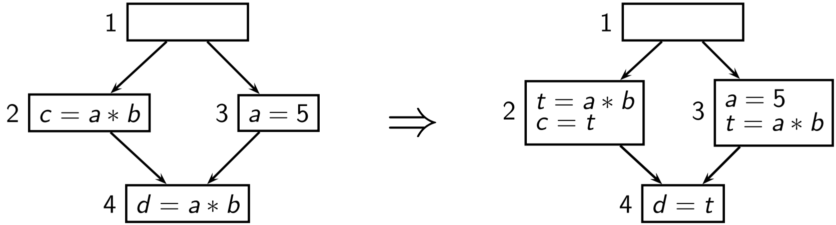


Performing PRE by Computing *In*/*Out*: Simple Cases (2)



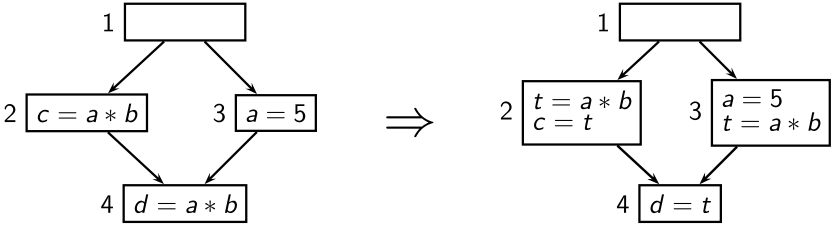
Node	First Level Values				Init.		Iter. 1		Iter. 2		Redund.	Insert
	AntGen	*Kill*	*PavIn*	*AvOut*	*Out*	*In*	*Out*	*In*	*Out*	*In*		
4												
3												
2												
1												

Performing PRE by Computing *In*/*Out*: Simple Cases (2)



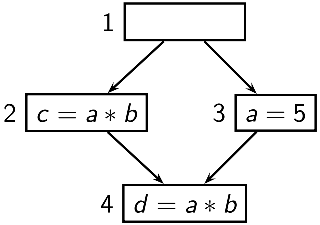
Node	First Level Values				Init.		Iter. 1		Iter. 2		Redund.	Insert
	AntGen	*Kill*	*PavIn*	*AvOut*	*Out*	*In*	*Out*	*In*	*Out*	*In*		
4	1	0	1	1								
3	0	1	0	0								
2	1	0	0	1								
1	0	0	0	0								

Performing PRE by Computing *In*/*Out*: Simple Cases (2)

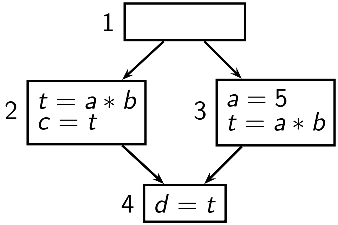


Node	First Level Values				Init.		Iter. 1		Iter. 2		Redund.	Insert
	AntGen	*Kill*	*PavIn*	*AvOut*	*Out*	*In*	*Out*	*In*	*Out*	*In*		
4	1	0	1	1	0	1						
3	0	1	0	0	1	1						
2	1	0	0	1	1	1						
1	0	0	0	0	1	1						

Performing PRE by Computing *In*/*Out*: Simple Cases (2)

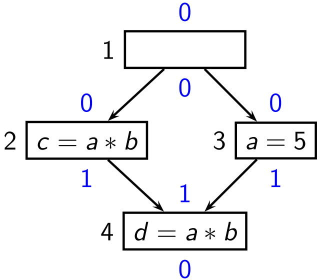


$\Rightarrow$

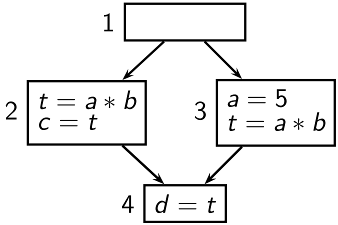


Node	First Level Values				Init.		Iter. 1		Iter. 2		Redund.	Insert
	AntGen	*Kill*	*PavIn*	*AvOut*	*Out*	*In*	*Out*	*In*	*Out*	*In*		
4	1	0	1	1	0	1	0	1				
3	0	1	0	0	1	1	1	0				
2	1	0	0	1	1	1	1	0				
1	0	0	0	0	1	1	0	0				

Performing PRE by Computing *In*/*Out*: Simple Cases (2)

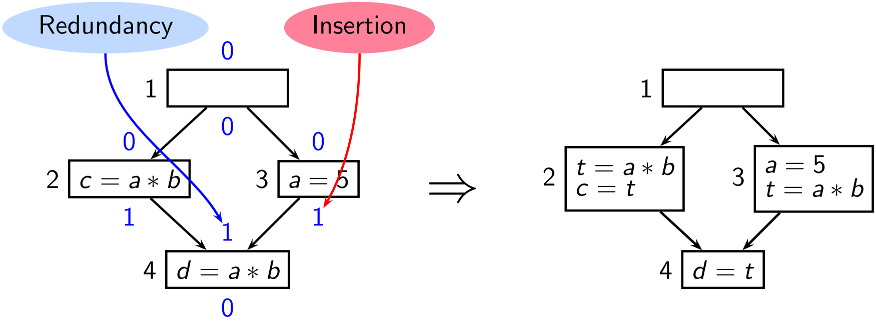


⇒



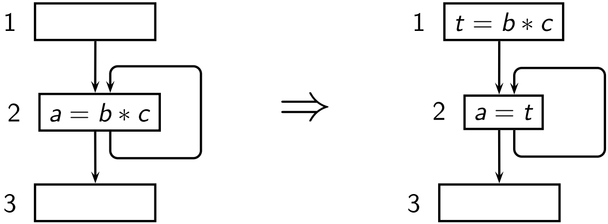
Node	First Level Values				Init.		Iter. 1		Iter. 2		Redund.	Insert
	AntGen	*Kill*	*PavIn*	*AvOut*	*Out*	*In*	*Out*	*In*	*Out*	*In*		
4	1	0	1	1	0	1	0	1	0	1		
3	0	1	0	0	1	1	1	0	1	0		
2	1	0	0	1	1	1	1	0	1	0		
1	0	0	0	0	1	1	0	0	0	0		

Performing PRE by Computing *In*/*Out*: Simple Cases (2)

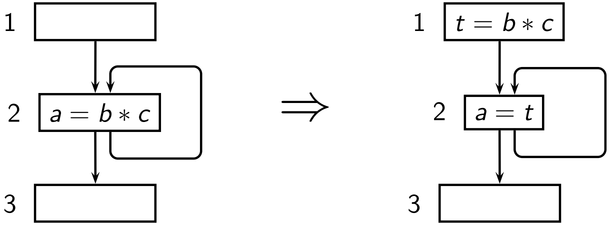


Node	First Level Values				Init.		Iter. 1		Iter. 2		Redund.	Insert
	AntGen	*Kill*	*PavIn*	*AvOut*	*Out*	*In*	*Out*	*In*	*Out*	*In*		
4	1	0	1	1	0	1	0	1	0	1	1	0
3	0	1	0	0	1	1	1	0	1	0	0	1
2	1	0	0	1	1	1	1	0	1	0	0	0
1	0	0	0	0	1	1	0	0	0	0	0	0

Performing PRE by Computing *In*/*Out*: Simple Cases (3)

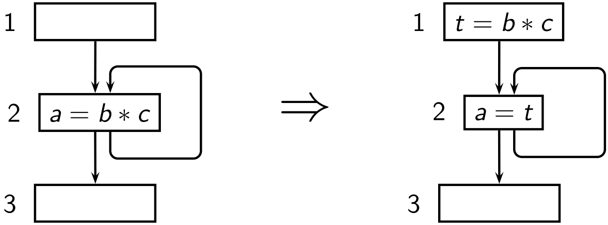


Performing PRE by Computing *In*/*Out*: Simple Cases (3)



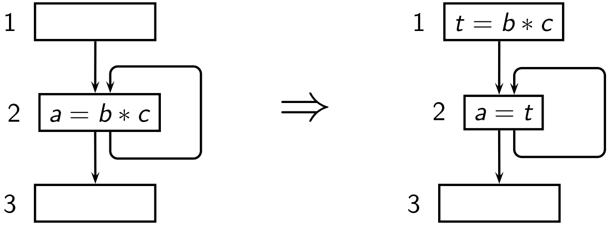
Node	First Level Values				Init.		Iter. 1		Iter. 2		Redund.	Insert
	AntGen	*Kill*	*PavIn*	*AvOut*	*Out*	*In*	*Out*	*In*	*Out*	*In*		
3												
2												
1												

Performing PRE by Computing *In*/*Out*: Simple Cases (3)



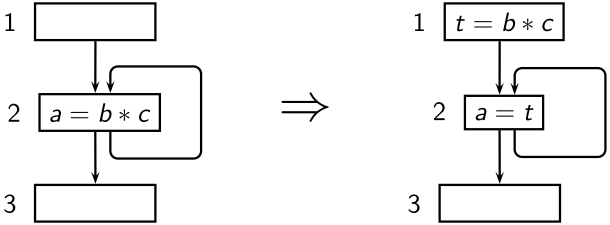
Node	First Level Values				Init.		Iter. 1		Iter. 2		Redund.	Insert
	AntGen	*Kill*	*PavIn*	*AvOut*	*Out*	*In*	*Out*	*In*	*Out*	*In*		
3	0	0	1	1								
2	1	0	1	1								
1	0	0	0	0								

Performing PRE by Computing *In*/*Out*: Simple Cases (3)



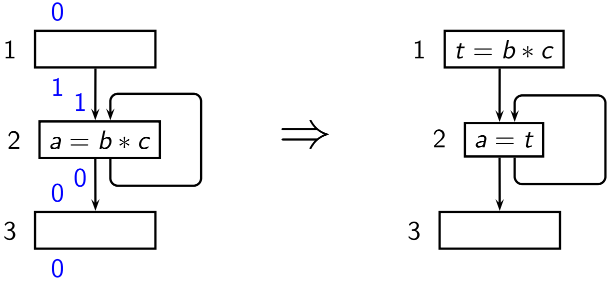
Node	First Level Values				Init.		Iter. 1		Iter. 2		Redund.	Insert
	AntGen	*Kill*	*PavIn*	*AvOut*	*Out*	*In*	*Out*	*In*	*Out*	*In*		
3	0	0	1	1	0	1						
2	1	0	1	1	1	1						
1	0	0	0	0	1	1						

Performing PRE by Computing *In*/*Out*: Simple Cases (3)



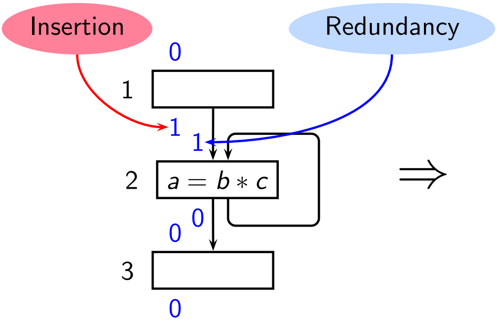
Node	First Level Values				Init.		Iter. 1		Iter. 2		Redund.	Insert
	AntGen	*Kill*	*PavIn*	*AvOut*	*Out*	*In*	*Out*	*In*	*Out*	*In*		
3	0	0	1	1	0	1	0	0				
2	1	0	1	1	1	1	0	1				
1	0	0	0	0	1	1	1	0				

Performing PRE by Computing *In*/*Out*: Simple Cases (3)

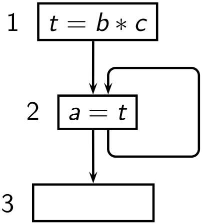


Node	First Level Values				Init.		Iter. 1		Iter. 2		Redund.	Insert
	AntGen	*Kill*	*PavIn*	*AvOut*	*Out*	*In*	*Out*	*In*	*Out*	*In*		
3	0	0	1	1	0	1	0	0	0	0		
2	1	0	1	1	1	1	0	1	0	1		
1	0	0	0	0	1	1	1	0	1	0		

Performing PRE by Computing *In*/*Out*: Simple Cases (3)

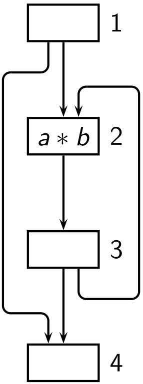


⇒



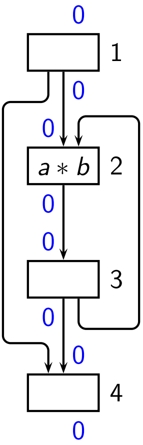
Node	First Level Values				Init.		Iter. 1		Iter. 2		Redund.	Insert
	AntGen	*Kill*	*PavIn*	*AvOut*	*Out*	*In*	*Out*	*In*	*Out*	*In*		
3	0	0	1	1	0	1	0	0	0	0	0	0
2	1	0	1	1	1	1	0	1	0	1	1	0
1	0	0	0	0	1	1	1	0	1	0	0	1

Tutorial Problems for PRE



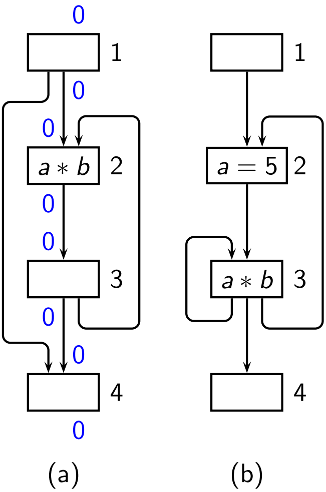
(a)

Tutorial Problems for PRE

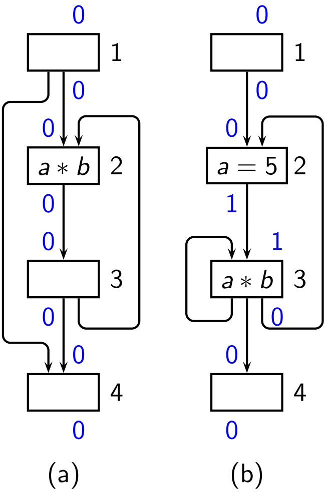


(a)

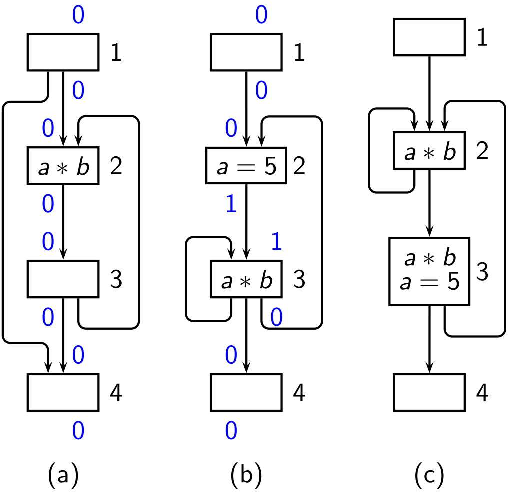
Tutorial Problems for PRE



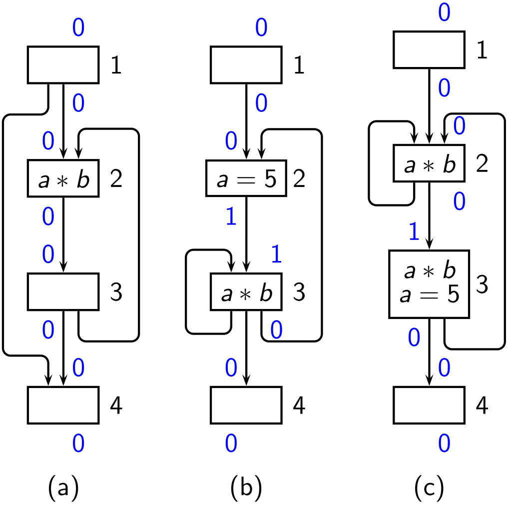
Tutorial Problems for PRE



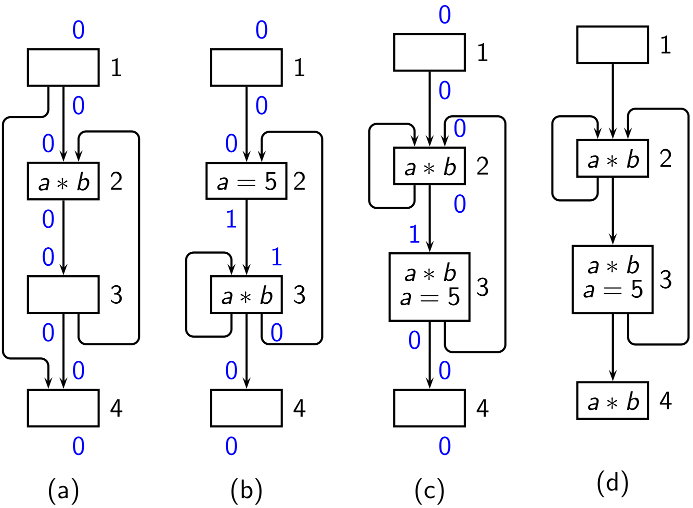
Tutorial Problems for PRE



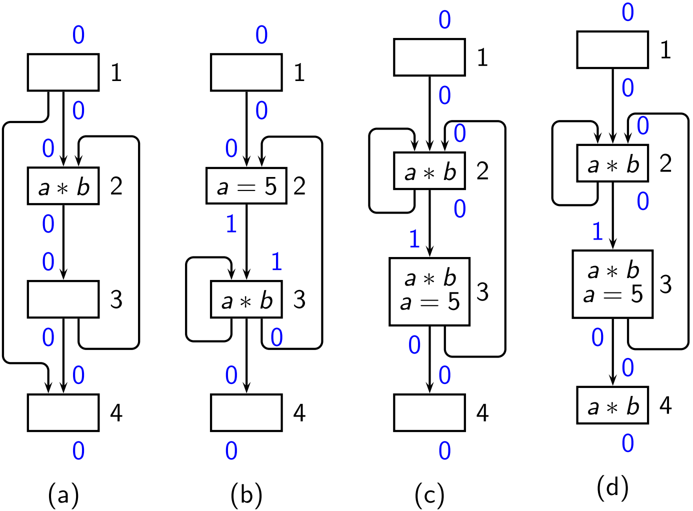
Tutorial Problems for PRE



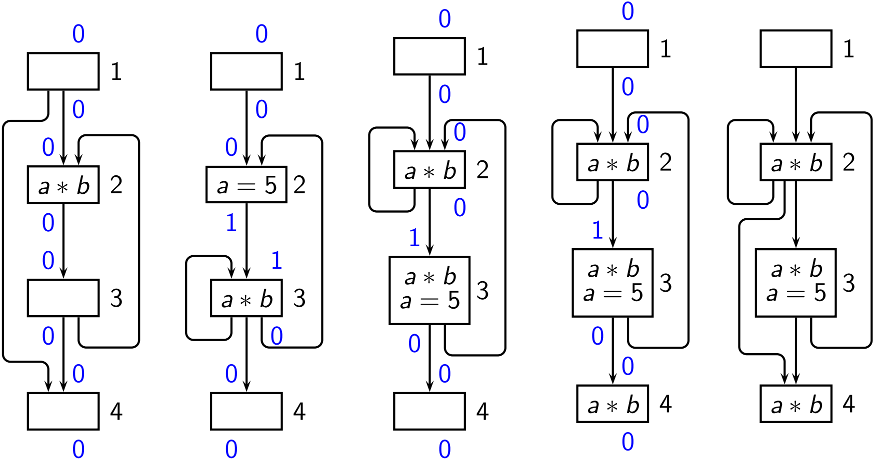
Tutorial Problems for PRE



Tutorial Problems for PRE

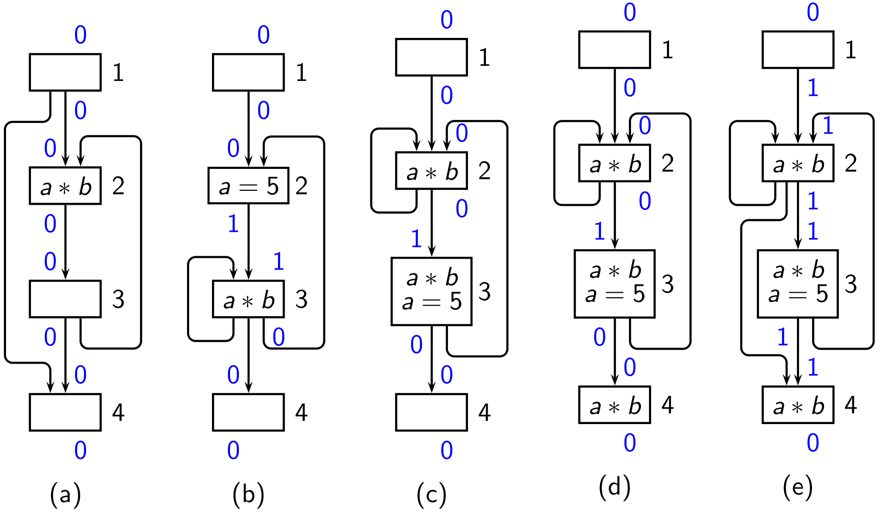


Tutorial Problems for PRE

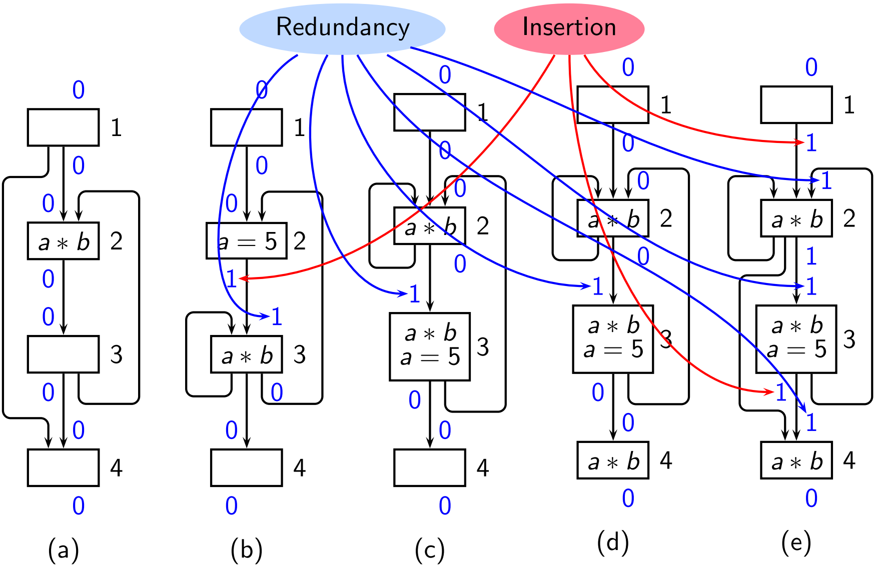


(a) (b) (c) (d) (e)

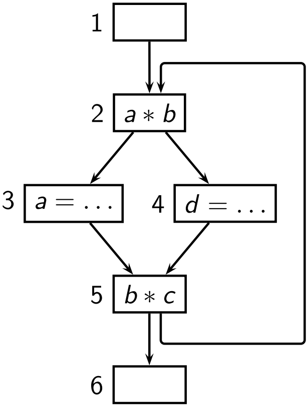
Tutorial Problems for PRE



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

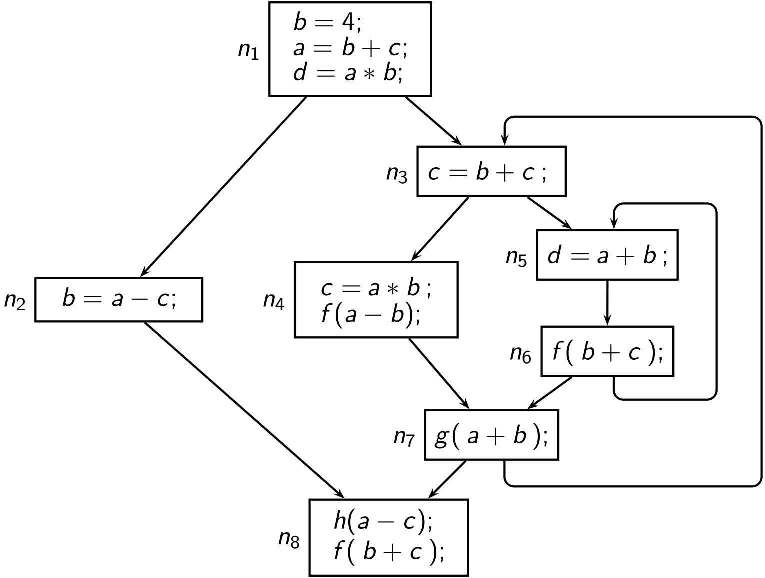
- Compute $In_n/Out_n/Redundant_n/Insert_n$
- Identify hoisting paths

Result of PRE Data Flow Analysis of the Running Example

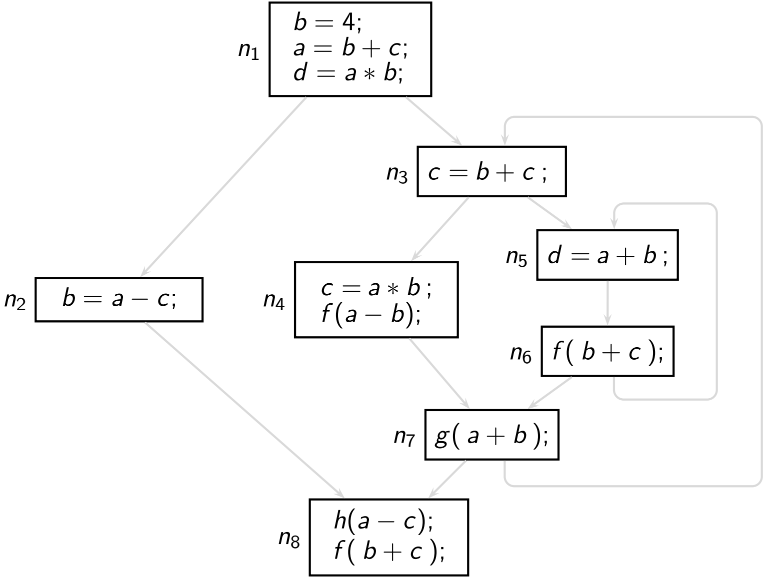
Bit vector	$a * b$	$a + b$	$a - b$	$a - c$	$b + c$

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				

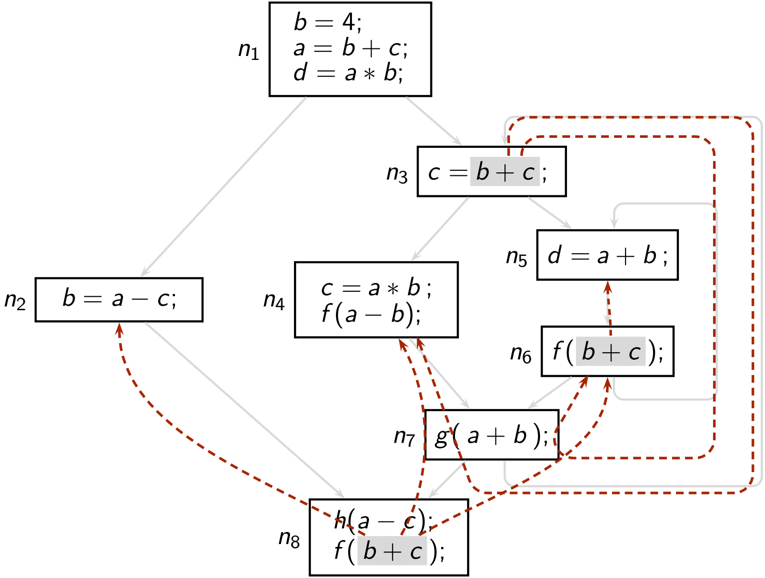
Hoisting Paths for Some Expressions in the Running Example



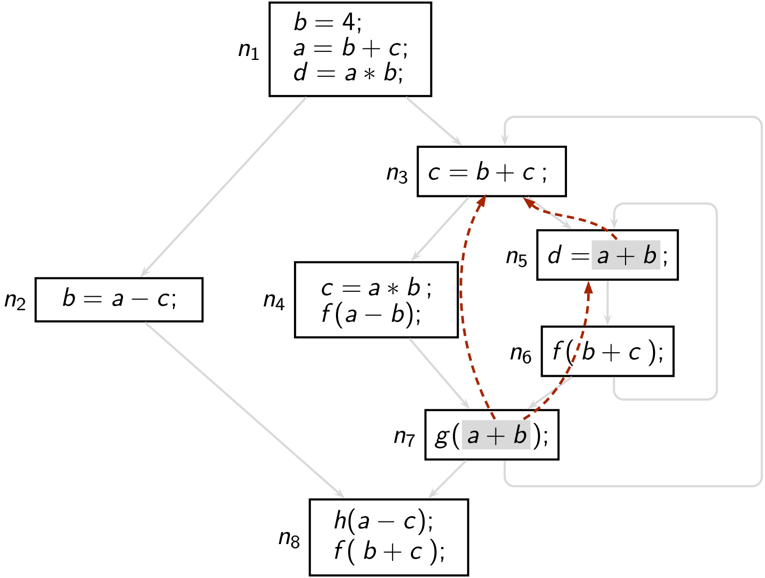
Hoisting Paths for Some Expressions in the Running Example



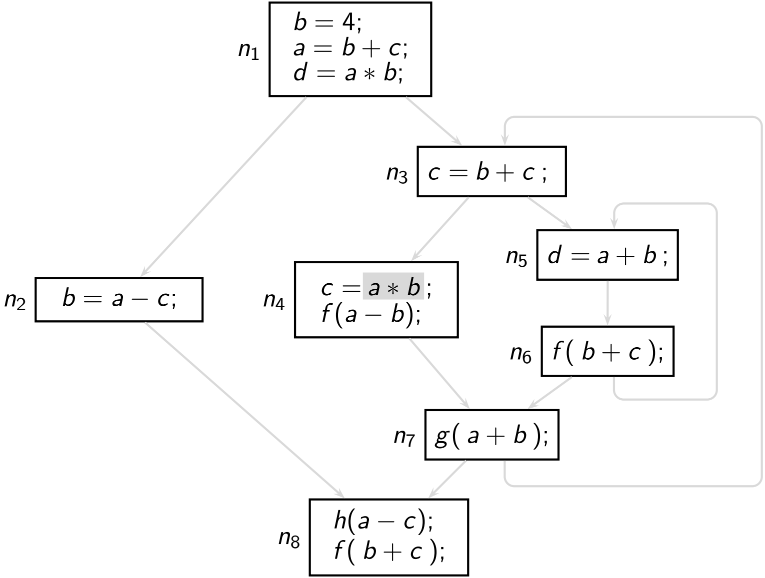
Hoisting Paths for Some Expressions in the Running Example



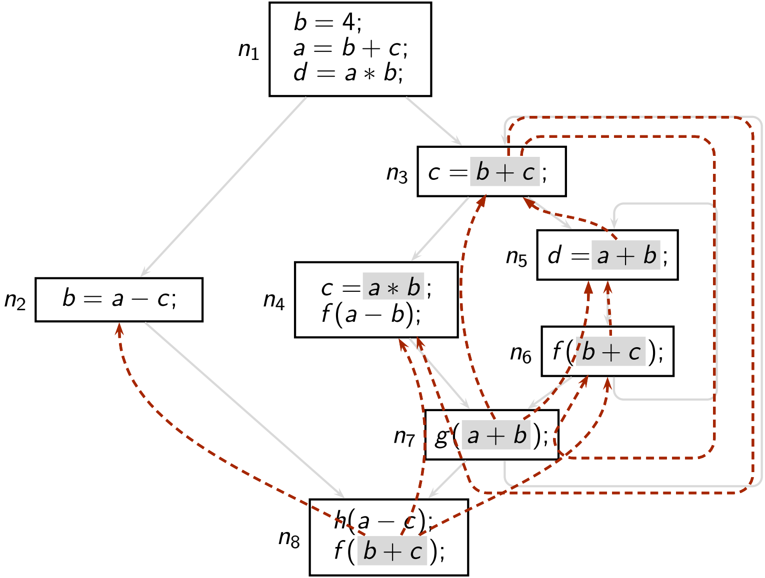
Hoisting Paths for Some Expressions in the Running Example



Hoisting Paths for Some Expressions in the Running Example



Hoisting Paths for Some Expressions in the Running Example



Optimized Version of the Running Example