Bit Vector Data Flow Frameworks

Uday P. Khedker

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

May 2011

Part 1

About These Slides

Copyright

Outline

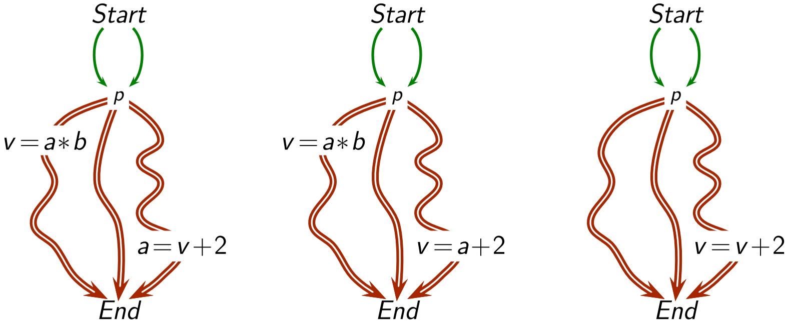
- Live Variables Analysis
- Available Expressions Analysis
- Anticipable Expressions Analysis
- Reaching Definitions Analysis
- Common Features of Bit Vector Frameworks

Part 2

Live Variables Analysis

Defining Live Variables Analysis

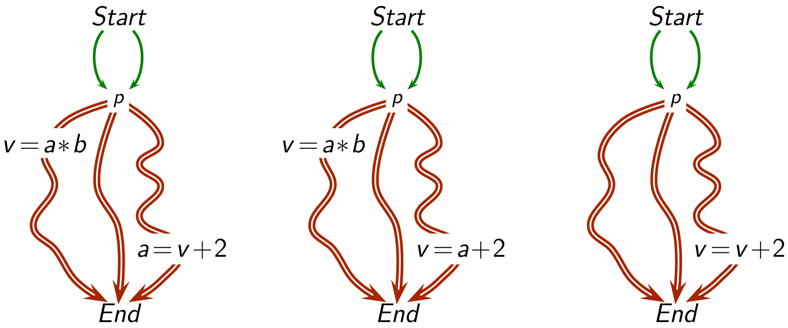
A variable *v* is live at a program point *p*, if some path from *p* to program exit contains an r-value occurrence of *v* which is not preceded by an l-value occurrence of *v*.



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.

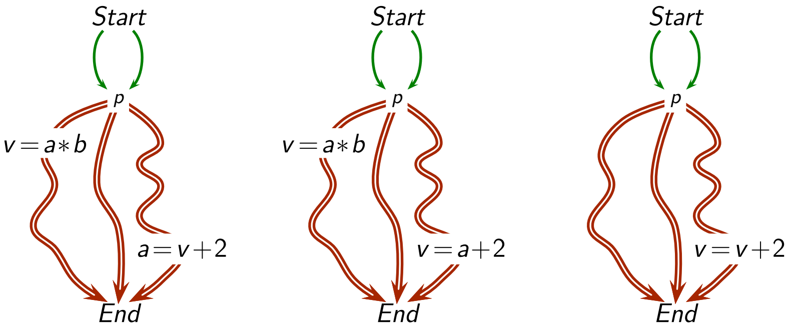
v is live at p



Defining Live Variables Analysis

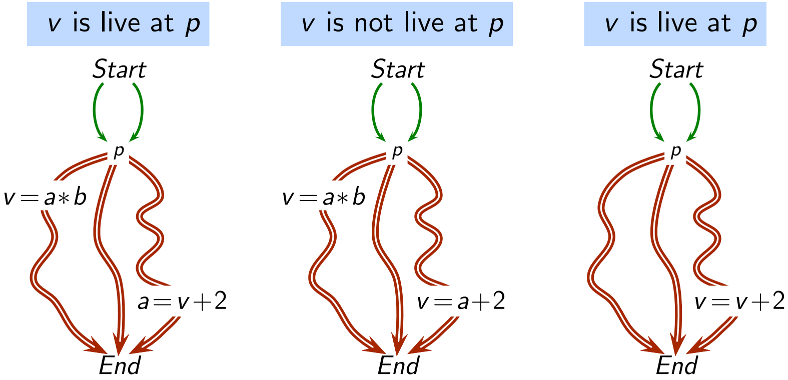
A variable v is live at a program point p, if some path from p to program exit contains an r-value occurrence of v which is not preceded by an l-value occurrence of v.

v is live at p | v is not live at p



Defining Live Variables Analysis

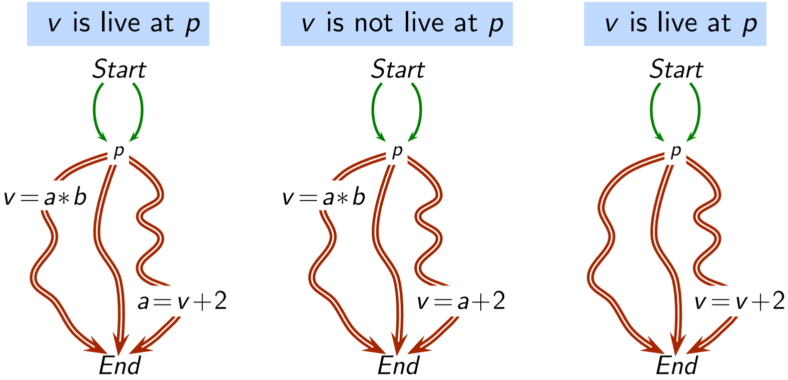
A variable v is live at a program point p, if some path from p to program exit contains an r-value occurrence of v which is not preceded by an l-value occurrence of v.



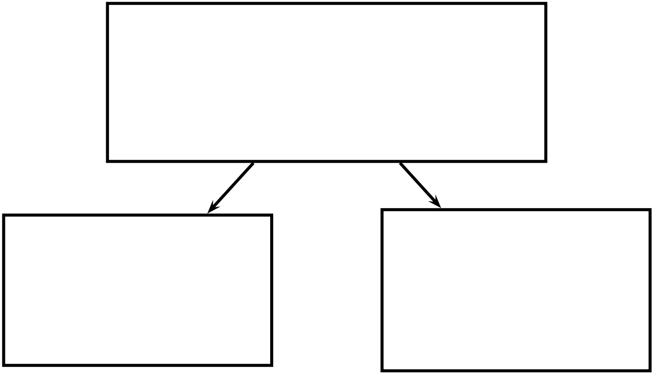
Defining Live Variables Analysis

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.

Path based specification

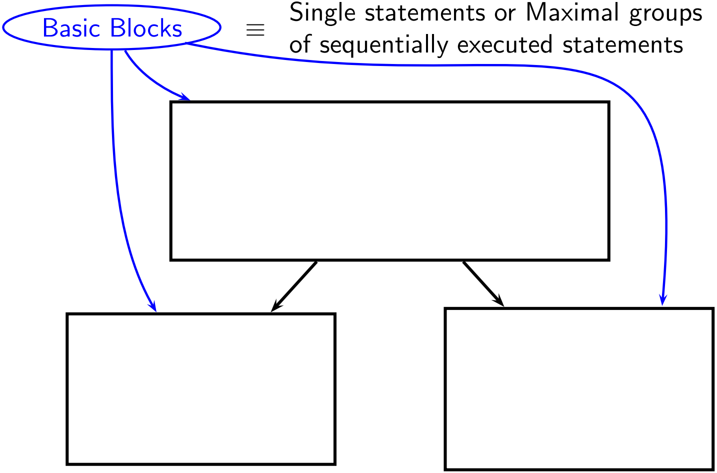


Defining Data Flow Analysis for Live Variables Analysis

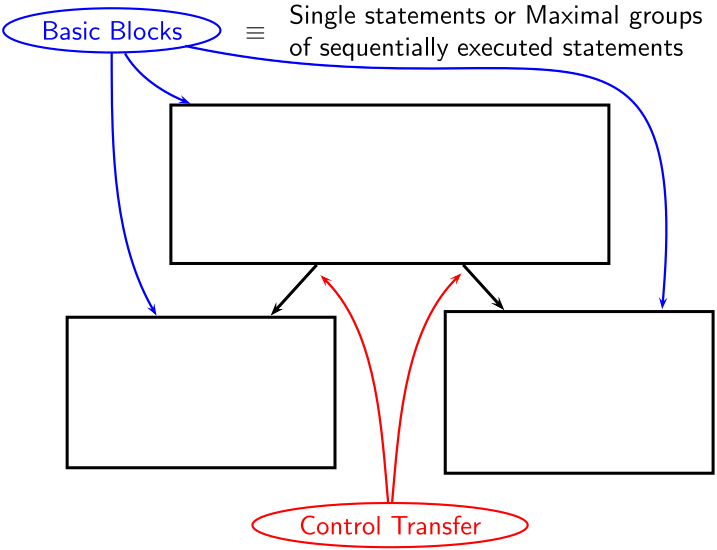


Defining Data Flow Analysis for Live Variables Analysis

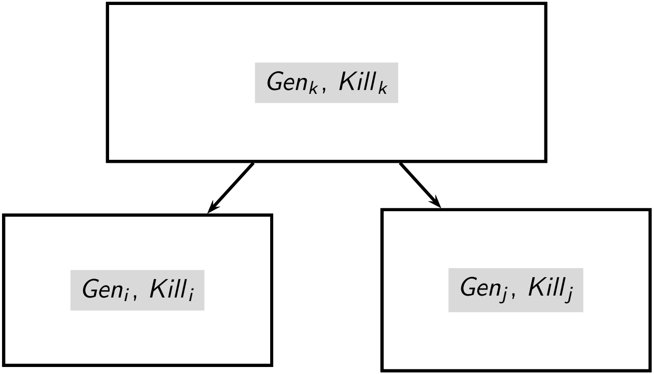
Basic Blocks ≡ Single statements or Maximal groups of sequentially executed statements



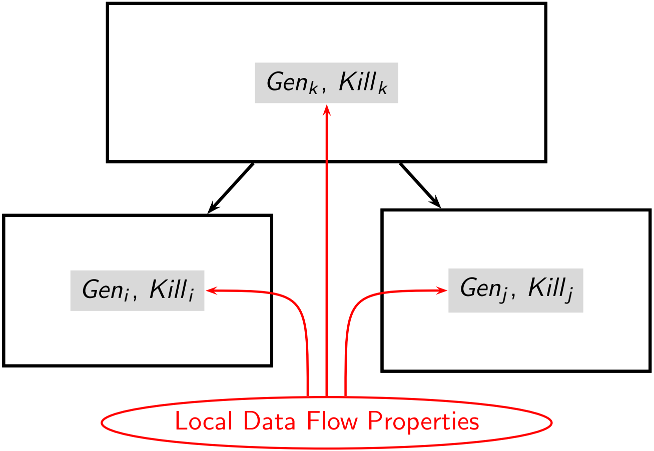
Defining Data Flow Analysis for Live Variables Analysis



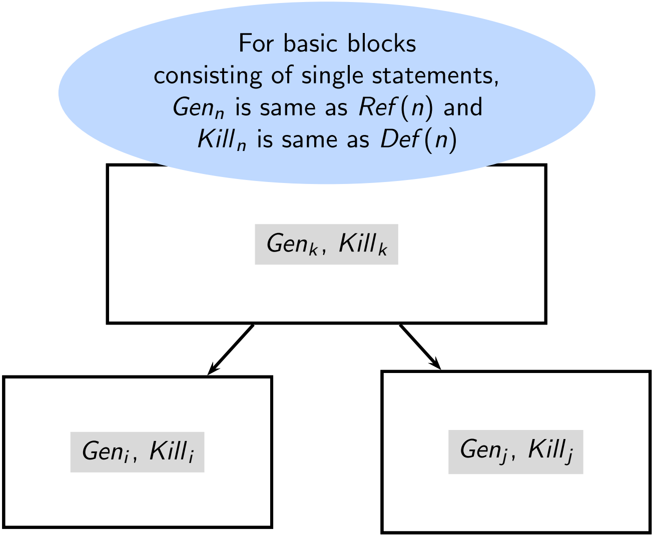
Defining Data Flow Analysis for Live Variables Analysis



Defining Data Flow Analysis for Live Variables Analysis



Defining Data Flow Analysis for Live Variables Analysis



Local Data Flow Properties for Live Variables Analysis

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

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

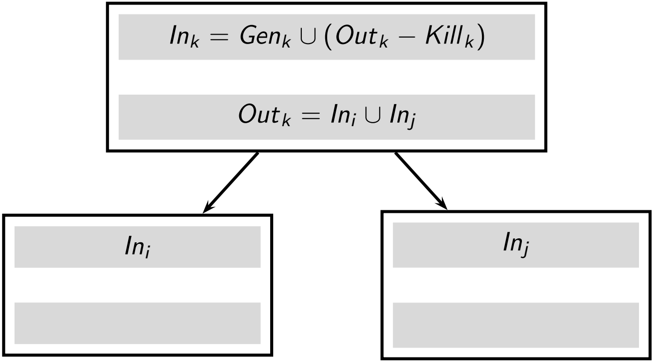
$$Gen_n = \{\, v \mid \text{variable } v \text{ is used in basic block } n \text{ and is not preceded by a definition of } v \,\}$$

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

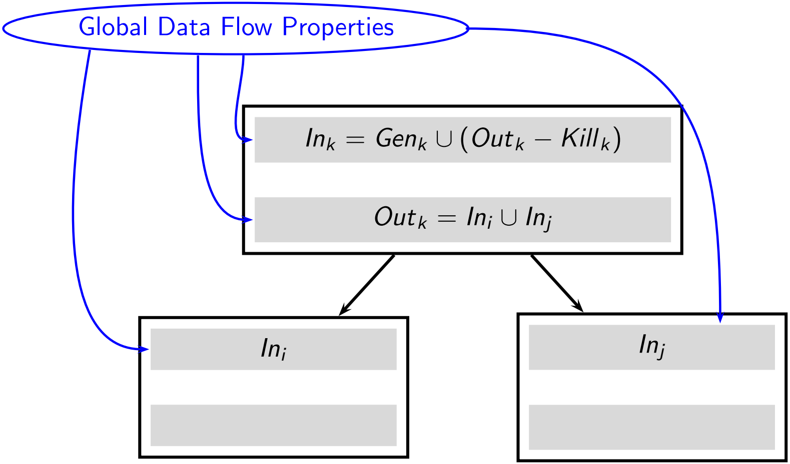
within n

anywhere in n

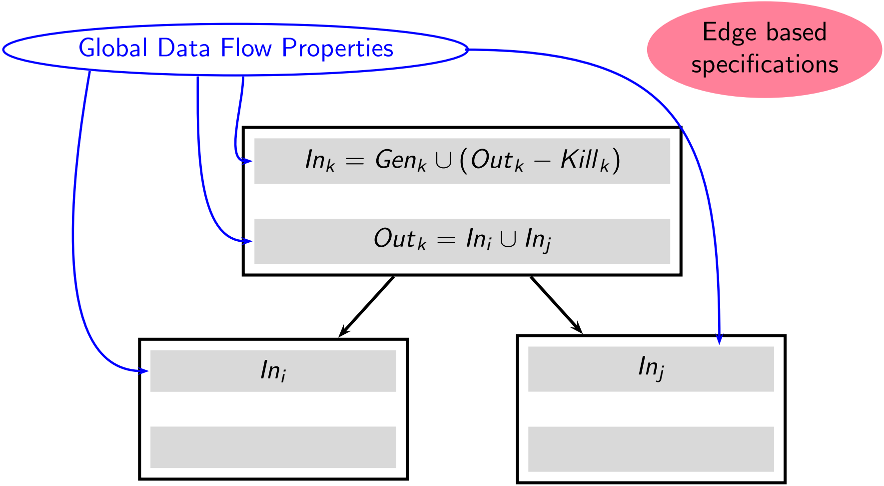
Defining Data Flow Analysis for Live Variables Analysis



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

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

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

- In_n and Out_n are sets of variables

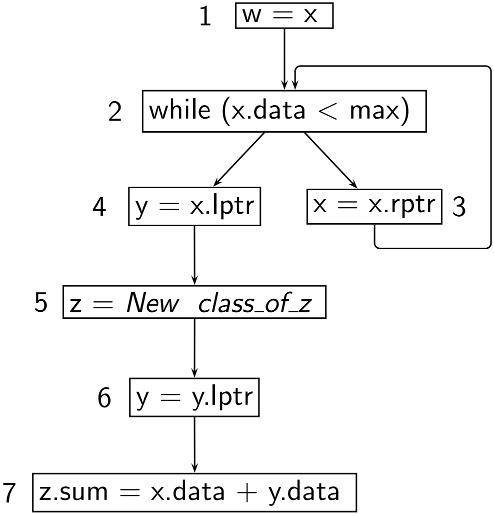
Data Flow Equations For Live Variables Analysis

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

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

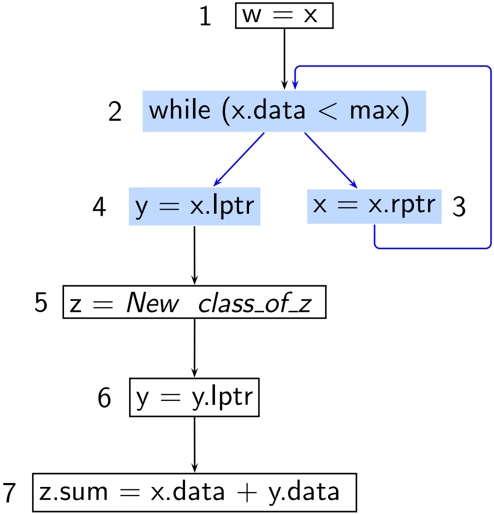
- In_n and Out_n are sets of variables
- BI is boundary information representin the effect of calling contexts
  - $\emptyset$ for local variables
  - set of global variables used further in any calling context (conveniently approximated by the set of all global variables)

Data Flow Equations for Our Example



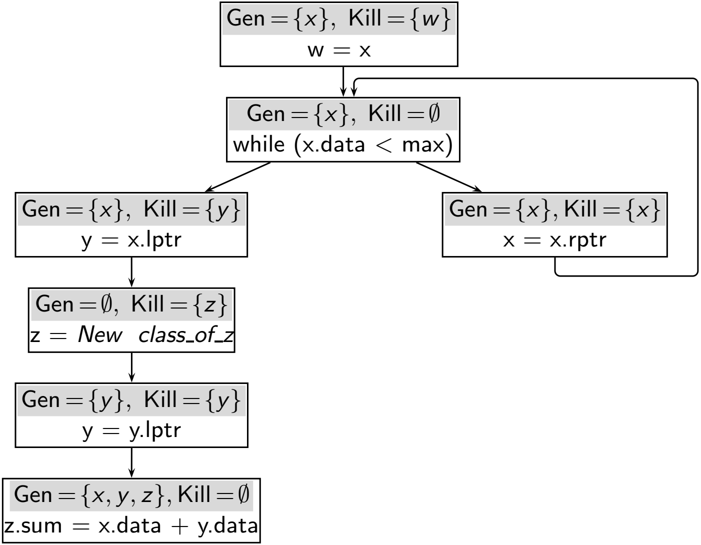
$$
\begin{aligned}
In_1 &= (Out_1 - Kill_1) \cup Gen_1 \\
Out_1 &= In_2 \\
In_2 &= (Out_2 - Kill_2) \cup Gen_2 \\
Out_2 &= In_3 \cup In_4 \\
In_3 &= (Out_3 - Kill_3) \cup Gen_3 \\
Out_3 &= In_2 \\
In_4 &= (Out_4 - Kill_4) \cup Gen_4 \\
Out_4 &= In_5 \\
In_5 &= (Out_5 - Kill_5) \cup Gen_5 \\
Out_5 &=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

Data Flow Equations for Our Example

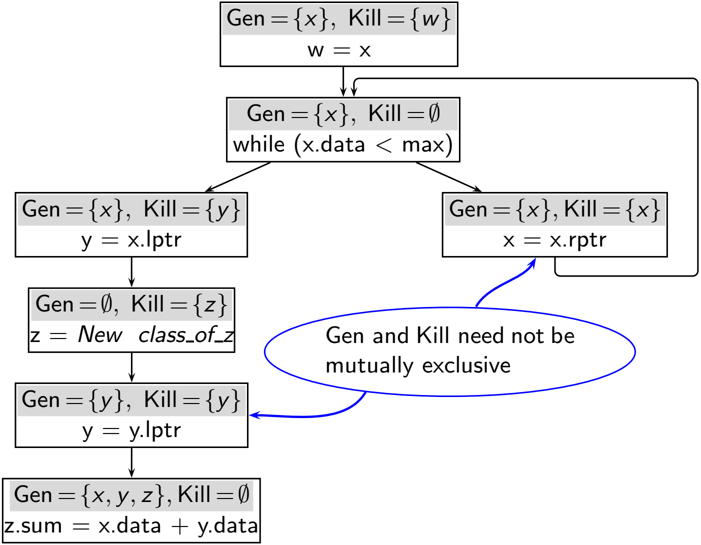


$$
\begin{aligned}
In_1 &= (Out_1 - Kill_1) \cup Gen_1 \\
Out_1 &= In_2 \\
In_2 &= (Out_2 - Kill_2) \cup Gen_2 \\
Out_2 &= In_3 \cup In_4 \\
In_3 &= (Out_3 - Kill_3) \cup Gen_3 \\
Out_3 &= In_2 \\
In_4 &= (Out_4 - Kill_4) \cup Gen_4 \\
Out_4 &= In_5 \\
In_5 &= (Out_5 - Kill_5) \cup Gen_5 \\
Out_5 &=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

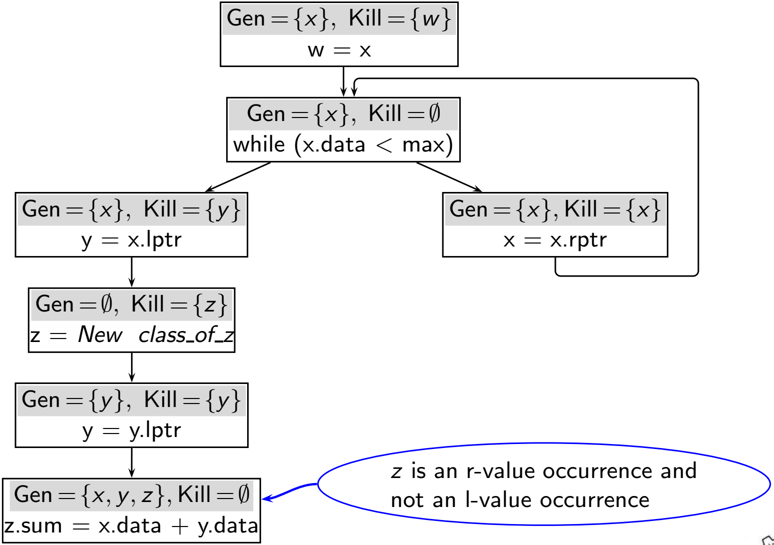
Performing Live Variables Analysis



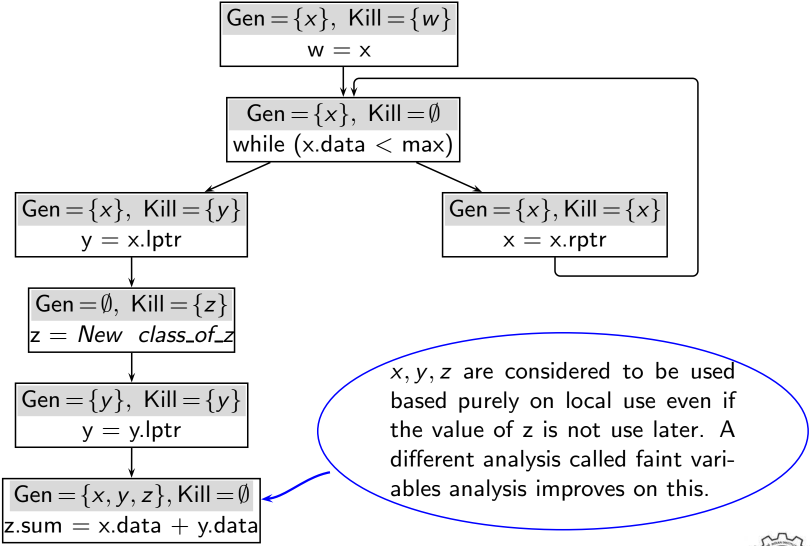
Performing Live Variables Analysis



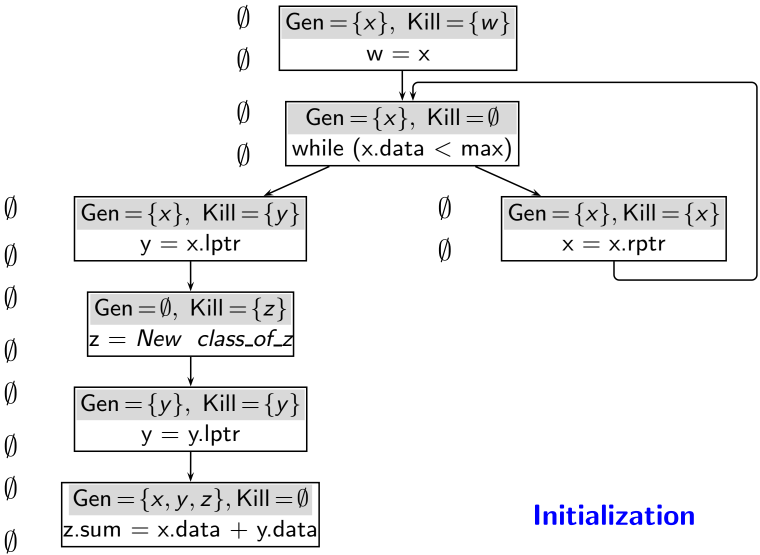
Performing Live Variables Analysis



Performing Live Variables Analysis

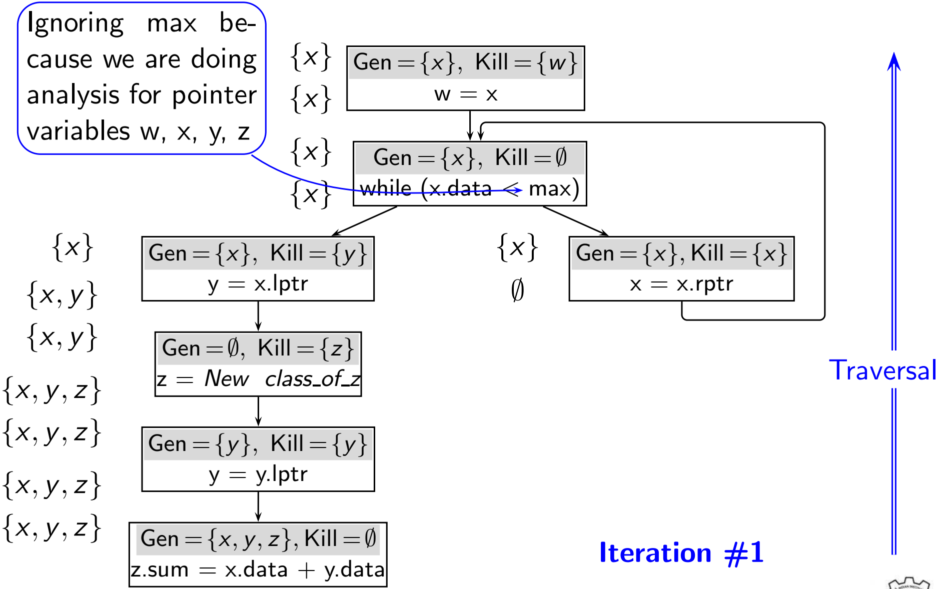


Performing Live Variables Analysis

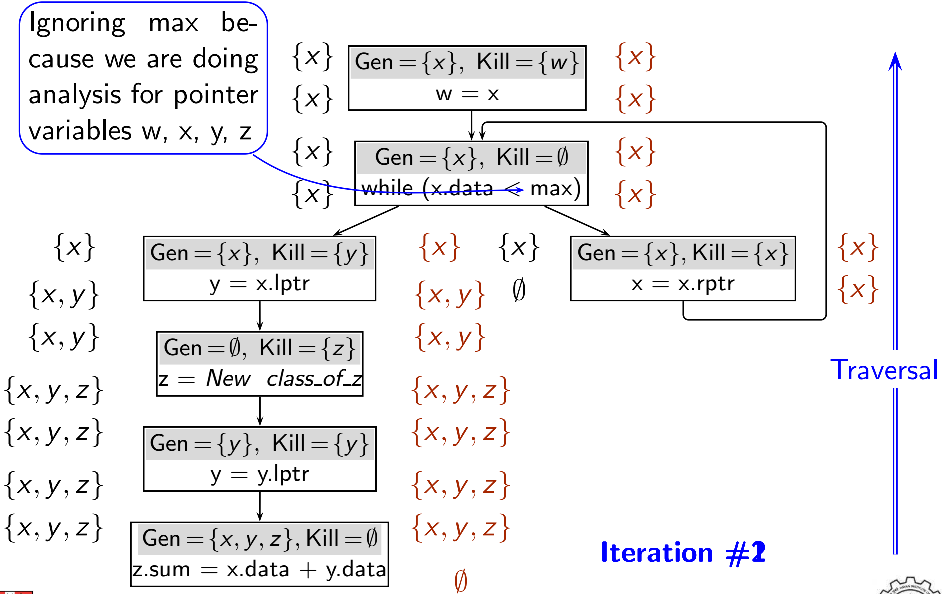


Initialization

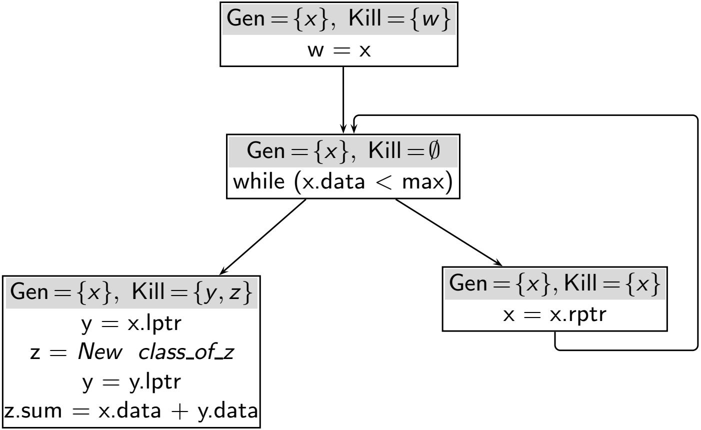
Performing Live Variables Analysis



Performing Live Variables Analysis



Performing Live Variables Analysis



Local Data Flow Properties for Live Variables Analysis

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

- Gen_n : Use not preceded by definition (Ref for a statement)
- $Kill_n$: Definition anywhere in a block (Def for a statement)

Local Data Flow Properties for Live Variables Analysis

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

- Gen_n : Use not preceded by definition (Ref for a statement)

  Upwards exposed use

- $Kill_n$: Definition anywhere in a block (Def for a statement)

  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 statement
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 statement

Using Data Flow Information of Live Variables Analysis

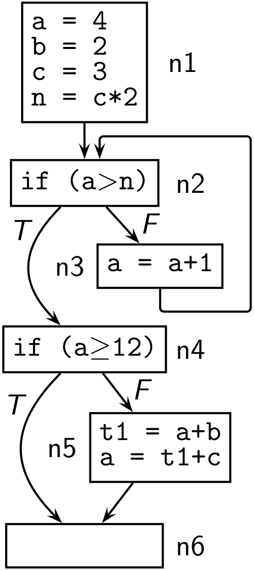
- Used for register allocation.
  If variable x is live in a basic block b, it is a potential candidate for register allocation.

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.

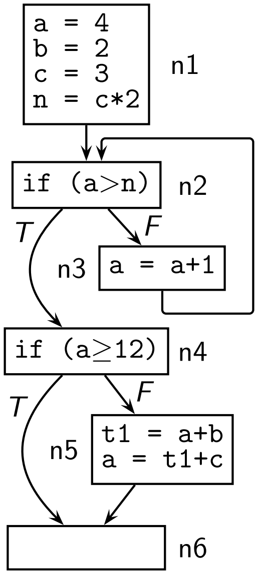
- Used for dead code elimination.
  If variable x is not live after an assignment $x = \ldots$, then the assginment is redundant and can be deleted as dead code.

Tutorial Problem 1 for Liveness Analysis



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	$\emptyset$	$\emptyset$

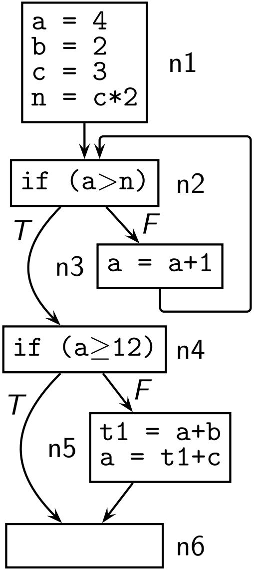
Tutorial Problem 1 for Liveness Analysis



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 for Liveness Analysis



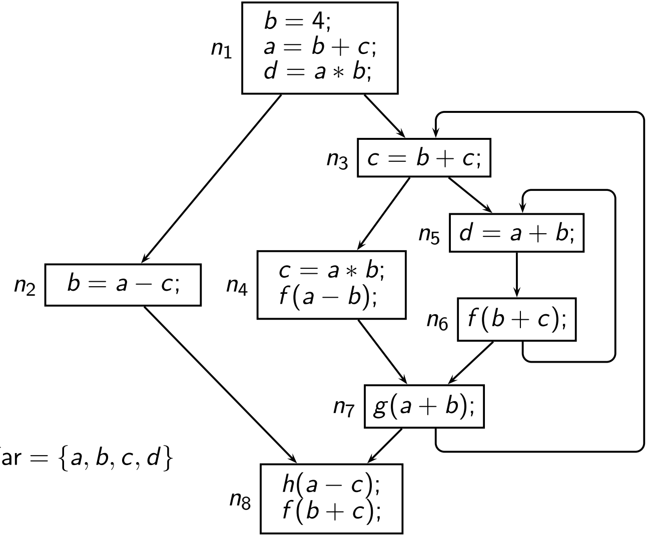
Local Data Flow Information		
	Gen	*Kill*
n1	$\emptyset$	$\{a, b, c, n\}$
n2	$\{a, n\}$	$\emptyset$
n3	$\{a\}$	$\{a\}$
n4	$\{a\}$	$\emptyset$
n5	$\{a, b, c\}$	$\{a, t1\}$
n6	$\emptyset$	$\emptyset$

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	$\emptyset$	$\{a\}$	$\{a, b, c\}$	$\{a, b, c\}$
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 2 for Liveness Analysis: C Program

```
int x, y, z;
int exmp(void)
{  int a, b, c, d;
   b = 4;
   a = b + c;
   d = a * b;
   if (x < y)
      b = a -c;
   else
   {  do
      {  c = b + c;
         if (y > x)
         {  do
            {  d = a + b;
               f(b + c);
            }  while(y > x);
         }
         else
         {  c = a * b;
            f(a - b);
         }
         g (a + b);
      }  while(z > x);
   }
   h(a-c);
   f(b+c);
}
```

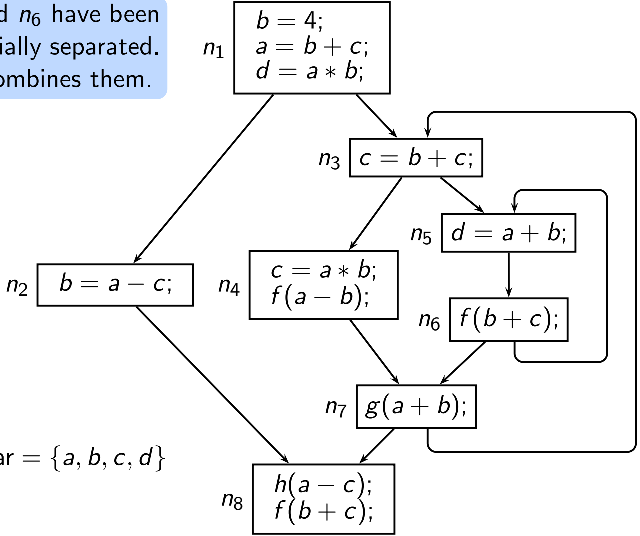
Tutorial Problem 2 for Liveness Analysis: Control Flow Graph



$\mathbb{V}\text{ar} = \{a, b, c, d\}$

Tutorial Problem 2 for Liveness Analysis: Control Flow Graph

n_5 and n_6 have been artificially separated. gcc combines them.



$\mathbb{V}\text{ar} = \{a, b, c, d\}$

Solution of the Tutorial Problem

Block	Local Information		Global Information			
			Iteration # 1		Iteration # 2	
	Gen_n	$Kill_n$	Out_n	In_n	Out_n	In_n
n_8	$\{a, b, c\}$	$\emptyset$	$\emptyset$	$\{a, b, c\}$	$\emptyset$	$\{a, b, c\}$
n_7	$\{a, b\}$	$\emptyset$	$\{a, b, c\}$	$\{a, b, c\}$	$\{a, b, c\}$	$\{a, b, c\}$
n_6	$\{b, c\}$	$\emptyset$	$\{a, b, c\}$	$\{a, b, c\}$	$\{a, b, c\}$	$\{a, b, c\}$
n_5	$\{a, b\}$	$\{d\}$	$\{a, b, c\}$	$\{a, b, c\}$	$\{a, b, c\}$	$\{a, b, c\}$
n_4	$\{a, b\}$	$\{c\}$	$\{a, b, c\}$	$\{a, b\}$	$\{a, b, c\}$	$\{a, b\}$
n_3	$\{b, c\}$	$\{c\}$	$\{a, b, c\}$	$\{a, b, c\}$	$\{a, b, c\}$	$\{a, b, c\}$
n_2	$\{a, c\}$	$\{b\}$	$\{a, b, c\}$	$\{a, c\}$	$\{a, b, c\}$	$\{a, c\}$
n_1	$\{c\}$	$\{a, b, d\}$	$\{a, b, c\}$	$\{c\}$	$\{a, b, c\}$	$\{c\}$

Tutorial Problems for Liveness Analysis

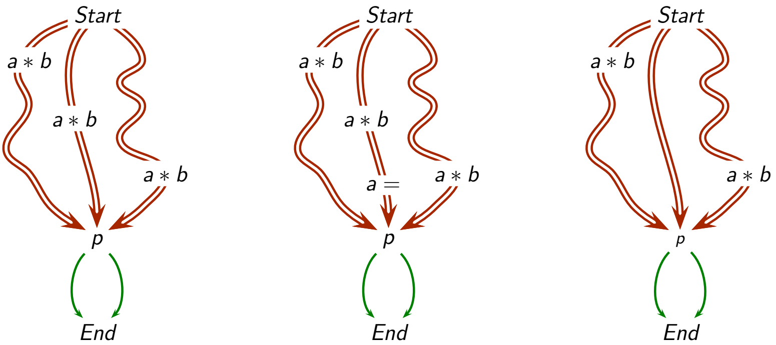
- Perform analysis with universal set $\mathbb{V}$ar as the initialization at internal nodes.
- Modify the previous program so that some data flow value computed in second iteration differs from the corresponding data flow value computed in the first iteration.
  (No structural changes, suggest at least two distinct kinds of modifications)
- Modify the above program so that some data flow value computed in third iteration differs from the corresponding data flow value computed in the second iteration.
  Write a C program corresponding to the modified control flow graph

Part 3

Available Expressions Analysis

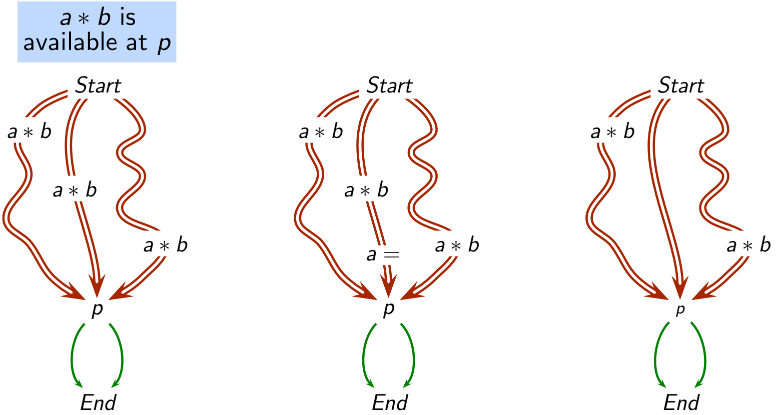
Defining Available Expressions Analysis

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.

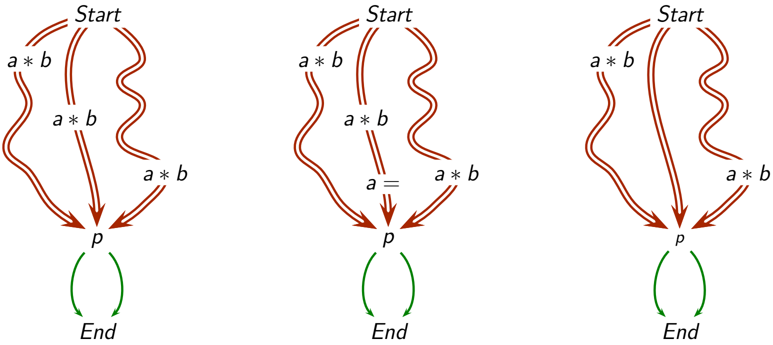


Defining Available Expressions Analysis

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.

$a * b$ is available at p

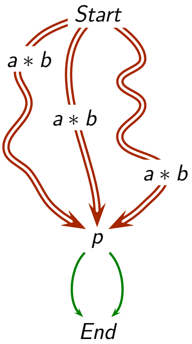
$a * b$ is not available at p



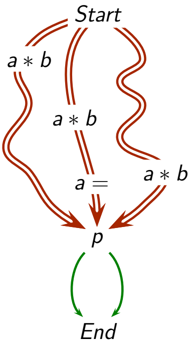
Defining Available Expressions Analysis

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.

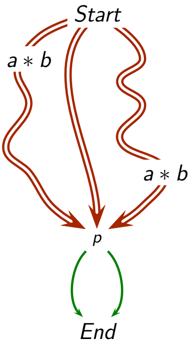
$a * b$ is available at p



$a * b$ is not available at p



$a * b$ is not available at p



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

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

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

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

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

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

Alternatively,

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

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

- In_n and Out_n are sets of expressions

Data Flow Equations For Available Expressions Analysis

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

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

Alternatively,

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

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

- 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 *b* **and**

Using Data Flow Information of Available Expressions Analysis

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

Using Data Flow Information of Available Expressions Analysis

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

Using Data Flow Information of Available Expressions Analysis

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

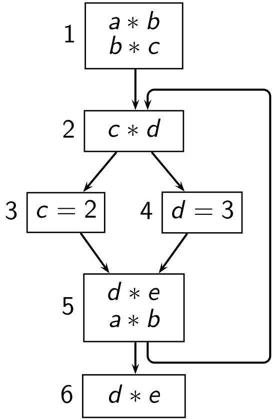
  Then the expression is redundant

Using Data Flow Information of Available Expressions Analysis

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

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

An Example of Available Expressions Analysis

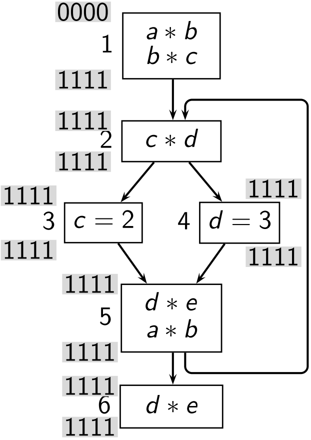


Let $e_1 \equiv a * b$, $e_2 \equiv b * c$, $e_3 \equiv c * d$, $e_4 \equiv d * e$

Node	Computed		Killed		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

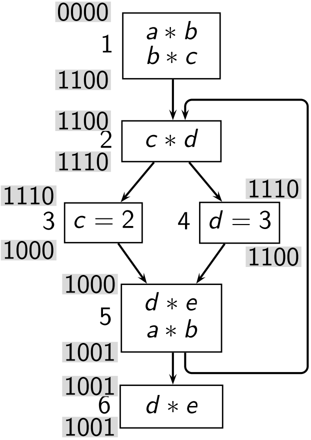


Let $e_1 \equiv a * b$, $e_2 \equiv b * c$, $e_3 \equiv c * d$, $e_4 \equiv d * e$

Node	Computed		Killed		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

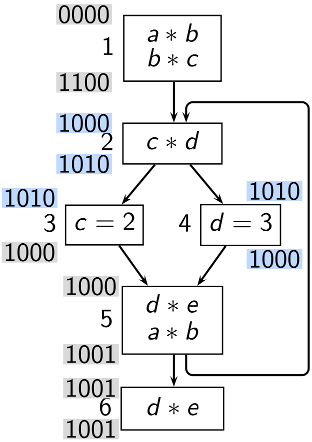


Let $e_1 \equiv a * b$, $e_2 \equiv b * c$, $e_3 \equiv c * d$, $e_4 \equiv d * e$

Node	Computed		Killed		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

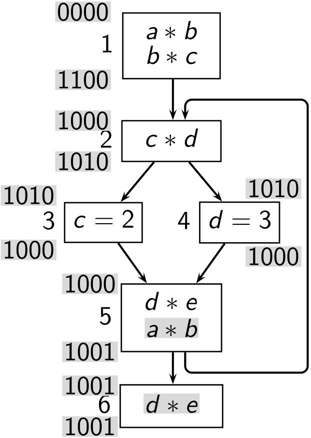


Let $e_1 \equiv a * b$, $e_2 \equiv b * c$, $e_3 \equiv c * d$, $e_4 \equiv d * e$

Node	Computed		Killed		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

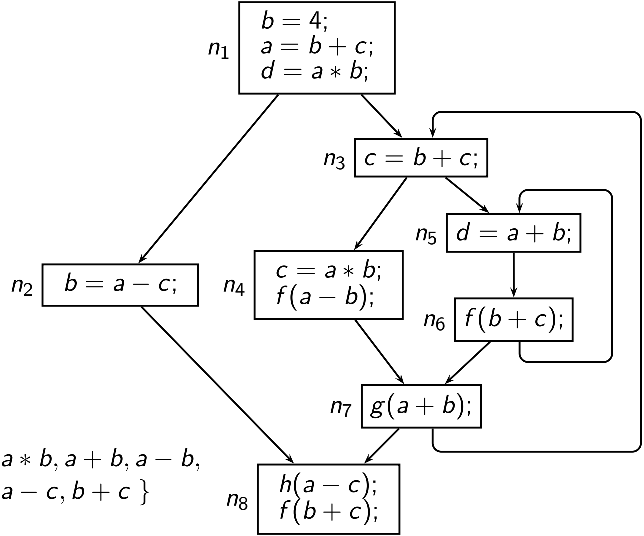
Final Result



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

Node	Computed		Killed		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 for Available Expressions Analysis



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

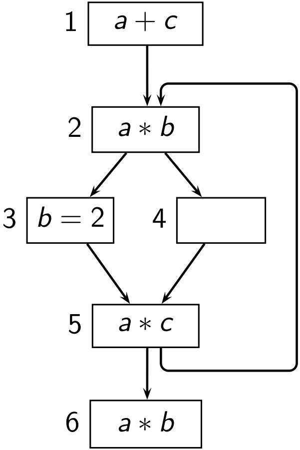
Solution of the Tutorial Problem

Bit vector

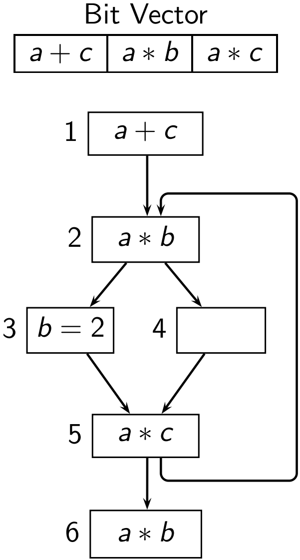
$a*b$	$a+b$	$a-b$	$a-c$	$b+c$

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	10001	11111	00000	00000	10001			00000
n_2	00010	11101	00010	10001	00010			00000
n_3	00000	00011	00001	10001	10000	10000		00000
n_4	10100	00011	10100	10000	10100			10000
n_5	01000	00000	01000	10000	11000			00000
n_6	00001	00000	00001	11000	11001			00000
n_7	01000	00000	01000	10000	11000			00000
n_8	00011	00000	00011	00000	00011			00000

Further Tutorial Problems

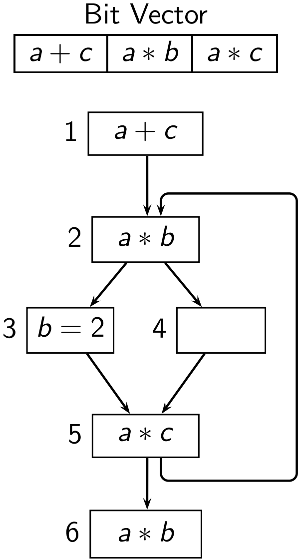


Further Tutorial Problems



BI	Node	Initialization $\mathbb{U}$		Initialization $\emptyset$	
		In_n	Out_n	In_n	Out_n
$\emptyset$	1				
	2				
	3				
	4				
	5				
	6				
$\mathbb{U}$	1				
	2				
	3				
	4				
	5				
	6				

Further Tutorial Problems

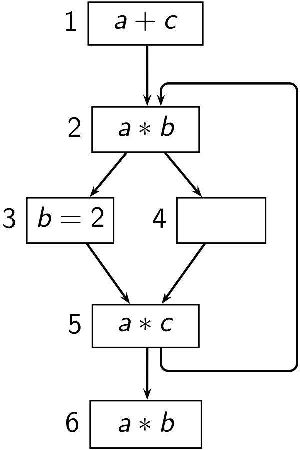


BI	Node	Initialization $\mathbb{U}$		Initialization $\emptyset$	
		In_n	Out_n	In_n	Out_n
$\emptyset$	1	000	100		
	2	100	110		
	3	110	100		
	4	110	110		
	5	100	101		
	6	101	111		
$\mathbb{U}$	1				
	2				
	3				
	4				
	5				
	6				

Further Tutorial Problems

Bit Vector

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

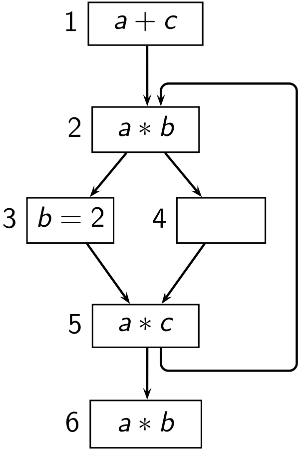


BI	Node	Initialization $\mathbb{U}$		Initialization $\emptyset$	
		In_n	Out_n	In_n	Out_n
$\emptyset$	1	000	100	000	100
	2	100	110	000	010
	3	110	100	010	000
	4	110	110	010	010
	5	100	101	000	001
	6	101	111	001	011
$\mathbb{U}$	1				
	2				
	3				
	4				
	5				
	6				

Further Tutorial Problems

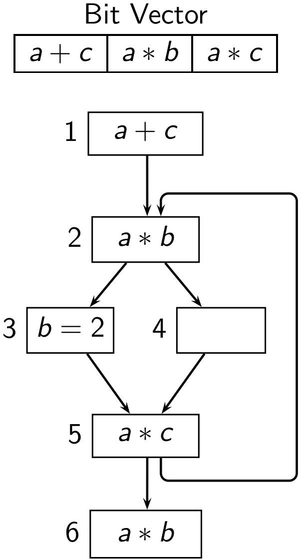
Bit Vector

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



BI	Node	Initialization $\mathbb{U}$		Initialization $\emptyset$	
		In_n	Out_n	In_n	Out_n
$\emptyset$	1	000	100	000	100
	2	100	110	000	010
	3	110	100	010	000
	4	110	110	010	010
	5	100	101	000	001
	6	101	111	001	011
$\mathbb{U}$	1	111	111		
	2	101	111		
	3	111	101		
	4	111	111		
	5	101	101		
	6	101	111		

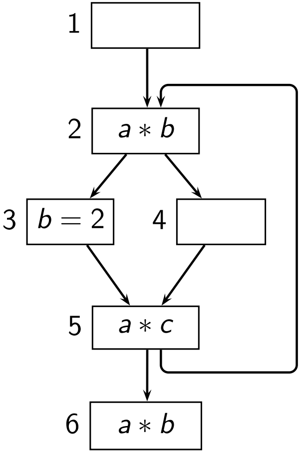
Further Tutorial Problems



BI	Node	Initialization $\mathbb{U}$		Initialization $\emptyset$	
		In_n	Out_n	In_n	Out_n
$\emptyset$	1	000	100	000	100
	2	100	110	000	010
	3	110	100	010	000
	4	110	110	010	010
	5	100	101	000	001
	6	101	111	001	011
$\mathbb{U}$	1	111	111	111	111
	2	101	111	001	011
	3	111	101	011	001
	4	111	111	011	011
	5	101	101	001	001
	6	101	111	001	011

More Tutorial Problems

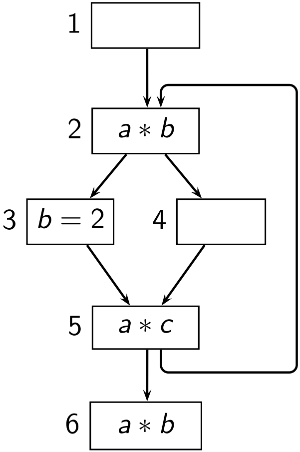
Number of iterations assuming that the order of In_i and Out_i computation is fixed (In_i is computed first and then Out_i is computed)



Traversal	Initialization			
	$\mathbb{U}$		$\emptyset$	
	BI		BI	
	$\mathbb{U}$	$\emptyset$	$\mathbb{U}$	$\emptyset$
Forward				
Backward				

More Tutorial Problems

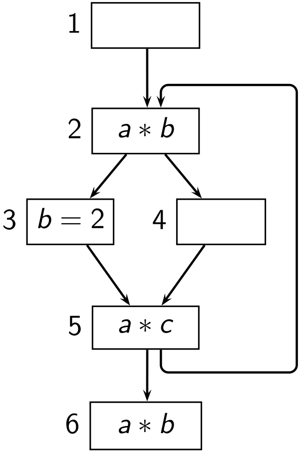
Number of iterations assuming that the order of In_i and Out_i computation is fixed (In_i is computed first and then Out_i is computed)



Traversal	Initialization			
	$\mathbb{U}$		$\emptyset$	
	BI		BI	
	$\mathbb{U}$	$\emptyset$	$\mathbb{U}$	$\emptyset$
Forward	2	1	2	1
Backward				

More Tutorial Problems

Number of iterations assuming that the order of In_i and Out_i computation is fixed (In_i is computed first and then Out_i is computed)



Traversal	Initialization			
	$\mathbb{U}$		$\emptyset$	
	BI		*BI*	
	$\mathbb{U}$	$\emptyset$	$\mathbb{U}$	$\emptyset$
Forward	2	1	2	1
Backward	3	4	4	2

Still More Tutorial Problems ☺

A New Data Flow Framework

- Partially available expressions at program point p are expressions that are computed and remain unmodified along some path reaching p. The data flow equations for partially available expressions analysis are same as the data flow equations of available expressions analysis except that the confluence is changed to $\cup$.

  Perform partially available expressions analysis for the previous example program.

Result of Partially Available Expressions Analysis

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

Node	Local Information			Global Information				
				Iteration # 1		Changes in iteration # 2		$ParRedund_n$
	Gen_n	$Kill_n$	$AntGen_n$	In_n	Out_n	In_n	Out_n	
n_1	10001	11111	00000	00000	10001			00000
n_2	00010	11101	00010	10001	00010			00000
n_3	00000	00011	00001	10001	10000	11101	11100	00001
n_4	10100	00011	10100	10000	10100	11100	11100	10100
n_5	01000	00000	01000	10000	11000	11101	11101	01000
n_6	00001	00000	00001	11000	11001	11101	11101	00001
n_7	01000	00000	01000	11101	11101			01000
n_8	00011	00000	00011	11111	11111			00011

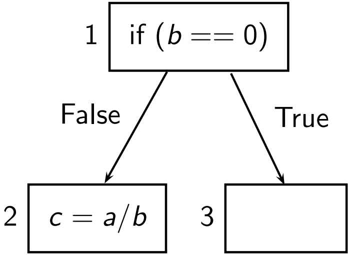
Part 4

Anticipable Expressions Analysis

Defining Anticipable Expressions Analysis

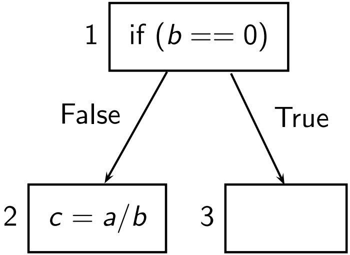
- An expression e is anticipable at a program point p, if every path from p to the program exit contains an evaluation of e which is not preceded by a redefinition of any operand of e.
- Application : Safety of Code Hoisting

Safety of Code Motion

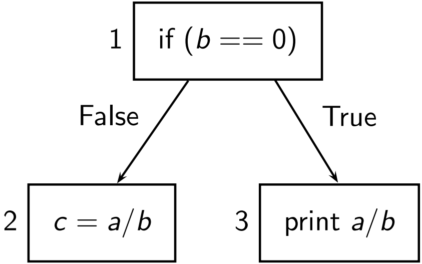


Hoisting a/b to the exit of 1 is unsafe ($\equiv$ can change the behaviour of the optimized program)

Safety of Code Motion

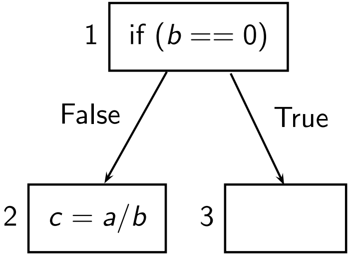


Hoisting a/b to the exit of 1 is unsafe ($\equiv$ can change the behaviour of the optimized program)

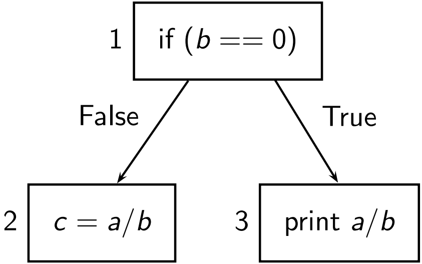


??

Safety of Code Motion



Hoisting a/b to 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 | expression e is evaluated in basic block n and this evaluation is not preceded (within n) by a definition of any operand of e}

$Kill_n$ = { e | basic block n contains a definition of an operand of e}

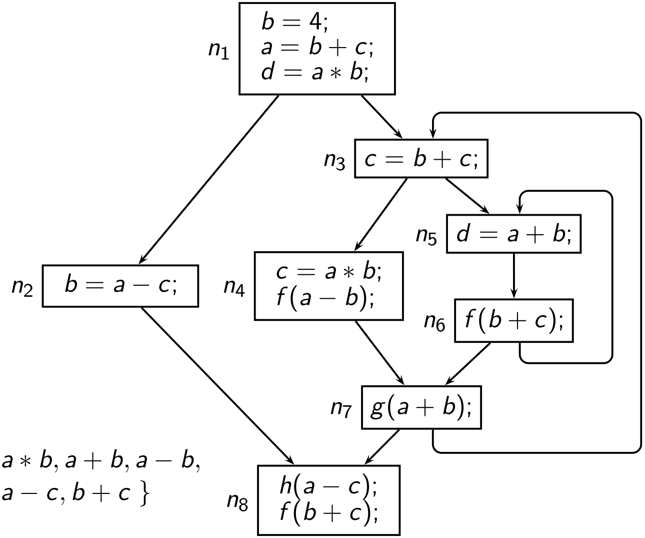
	Entity	Manipulation	Exposition
Gen_n	Expression	Use	Upwards
$Kill_n$	Expression	Modification	Anywhere

Data Flow Equations for Anticipable Expressions Analysis

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

In_n and Out_n are sets of expressions

Tutorial Problem for Anticipable Expressions Analysis



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

Result of Anticipable Expressions Analysis

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

Block	Local Information		Global Information: Iteration # 1		Global Information: Changes in iteration # 2	
	Gen_n	$Kill_n$	Out_n	In_n	Out_n	In_n
n_8	00011	00000	00000	00011		
n_7	01000	00000	00011	01011	00001	01001
n_6	00001	00000	01011	01011	01001	01001
n_5	01000	00000	01011	01011	01001	01001
n_4	10100	00011	01011	11100	01001	11100
n_3	00001	00011	01000	01001	01000	01001
n_2	00010	11101	00011	00010		
n_1	00000	11111	00000	00000		

Part 5

Reaching Definitions Analysis

Defining Reaching Definitions Analysis

- A definition $d_x : x = y$ reaches a program point u if it appears (without a refefinition of x) on some path from program entry to u
- Application : Copy Propagation
  A use of a variable x at a program point u 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.

Defining Data Flow Analysis for Reaching Definitions Analysis

Let d_v be a definition of variable v

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

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

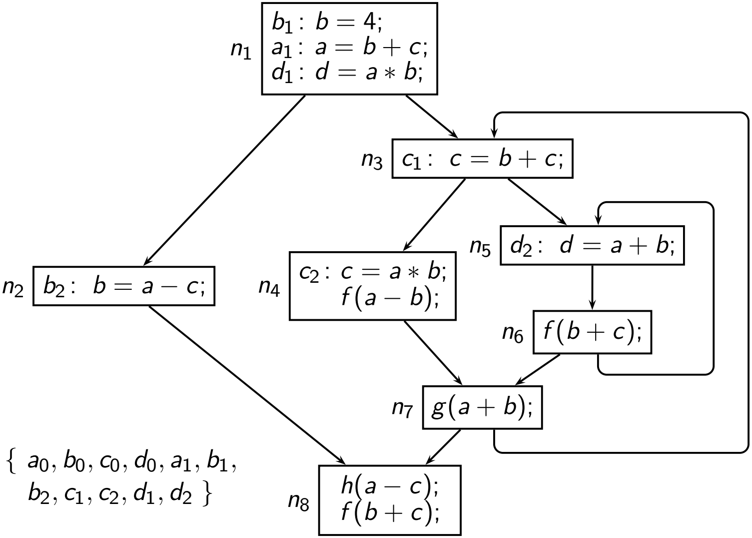
	Entity	Manipulation	Exposition
Gen_n	Definition	Occurence	Downwards
$Kill_n$	Definition	Occurence	Anywhere

Data Flow Equations for Reaching Definitions Analysis

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

In_n and Out_n are sets of definitions

Tutorial Problem for Reaching Definitions Analysis



$\mathbb{D}\text{efs} = \{ a_0, b_0, c_0, d_0, a_1, b_1, b_2, c_1, c_2, d_1, d_2 \}$

Result of Reaching Definitions Analysis

Block	Local Information		Global Information			
			Iteration # 1		Changes in iteration # 2	
	Gen_n	$Kill_n$	In_n	Out_n	In_n	Out_n
n_1	$\{a_1, b_1, d_1\}$	$\{a_0, a_1, b_0, b_1, b_2, d_0, d_1, d_2\}$	$\{a_0, b_0, c_0, d_0\}$	$\{a_1, b_1, c_0, d_1\}$		
n_2	$\{b_2\}$	$\{b_0, b_1, b_2\}$	$\{a_1, b_1, c_0, d_1\}$	$\{a_1, b_2, c_0, d_1\}$		
n_3	$\{c_1\}$	$\{c_0, c_1, c_2\}$	$\{a_1, b_1, c_0, d_1\}$	$\{a_1, b_1, c_1, d_1\}$	$\{a_1, b_1, c_0, c_1, c_2, d_1, d_2\}$	$\{a_1, b_1, c_1, d_1, d_2\}$
n_4	$\{c_2\}$	$\{c_0, c_1, c_2\}$	$\{a_1, b_1, c_1, d_1\}$	$\{a_1, b_1, c_2, d_1\}$	$\{a_1, b_1, c_1, d_1, d_2\}$	$\{a_1, b_1, c_2, d_1, d_2\}$
n_5	$\{d_2\}$	$\{d_0, d_1, d_2\}$	$\{a_1, b_1, c_1, d_1\}$	$\{a_1, b_1, c_1, d_2\}$	$\{a_1, b_1, c_1, d_1, d_2\}$	
n_6	$\emptyset$	$\emptyset$	$\{a_1, b_1, c_1, d_2\}$	$\{a_1, b_1, c_1, d_2\}$		
n_7	$\emptyset$	$\emptyset$	$\{a_1, b_1, c_1, c_2, d_1, d_2\}$	$\{a_1, b_1, c_1, c_2, d_1, d_2\}$		
n_8	$\emptyset$	$\emptyset$	$\{a_1, b_1, b_2, c_0, c_1, c_2, d_1, d_2\}$	$\{a_1, b_1, b_2, c_0, c_1, c_2, d_1, d_2\}$		

Part 6

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

$$X_i = f(Y_i)$$

$$Y_i = \bigsqcap X_j$$

Common Form of Data Flow Equations

Data Flow Information

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

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

Common Form of Data Flow Equations

Data Flow Information

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

Flow Function

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

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

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 Exressions
Bidirectional (limited)		Partial Redundancy Elimination (Original M-R Formulation)

A Taxonomy of Bit Vector Data Flow Frameworks

Any Path

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

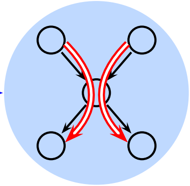
A Taxonomy of Bit Vector Data Flow Frameworks

Any Path → Union

All Paths → Intersection

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

A Taxonomy of Bit Vector Data Flow Frameworks

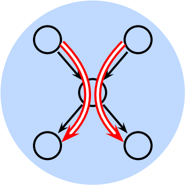


Any Path

All Paths

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

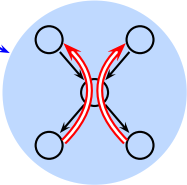
A Taxonomy of Bit Vector Data Flow Frameworks



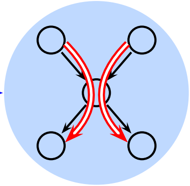
Any Path

All Paths

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



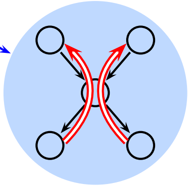
A Taxonomy of Bit Vector Data Flow Frameworks

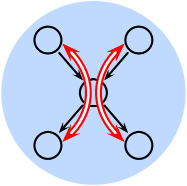


Any Path

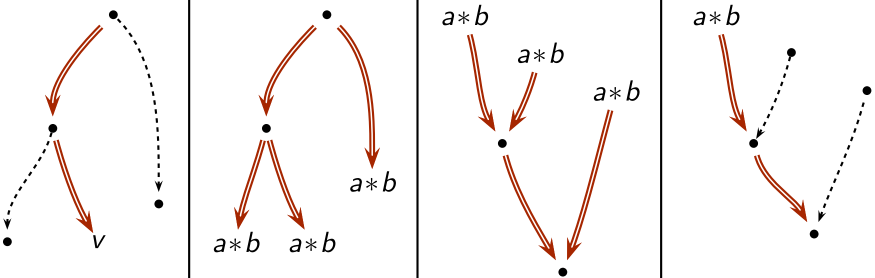
All Paths

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



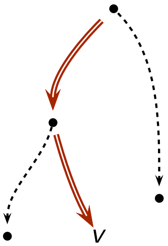


Data Flow Paths Discovered by Data Flow Analysis



Liveness | Anticipability | Availability | Partial Availability

Data Flow Paths Discovered by Data Flow Analysis

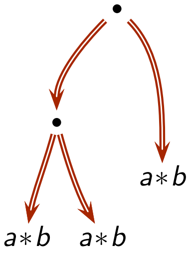


Liveness

Sequence of blocks $(b_1, b_2, \ldots, b_k)$ which is a prefix of some potential execution path starting at b_1 such that:

- b_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

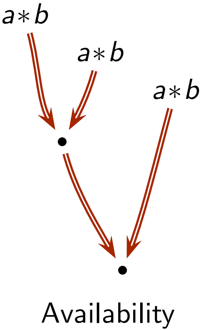
Sequence of blocks $(b_1, b_2, \ldots, b_k)$ which is a prefix of some potential execution path starting at b_1 such that:

- b_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 b_1 is an anticipability path of $a * b$.

Data Flow Paths Discovered by Data Flow Analysis

Sequence of blocks $(b_1, b_2, \ldots, b_k)$ which is a prefix of some potential execution path starting at b_1 such that:

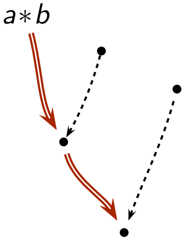
- b_1 contains a downwards exposed use of $a * b$, **and**
- no other block on the path contains an assignment to a or b, **and**
- every path ending at b_k is an availability path of $a * b$.



Data Flow Paths Discovered by Data Flow Analysis

Sequence of blocks $(b_1, b_2, \ldots, b_k)$ which is a prefix of some potential execution path starting at b_1 such that:

- b_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