Workshop on Essential Abstractions in GCC

Introduction to Data Flow Analysis

GCC Resource Center

(www.cse.iitb.ac.in/grc)

Department of Computer Science and Engineering,

Indian Institute of Technology, Bombay

July 2010

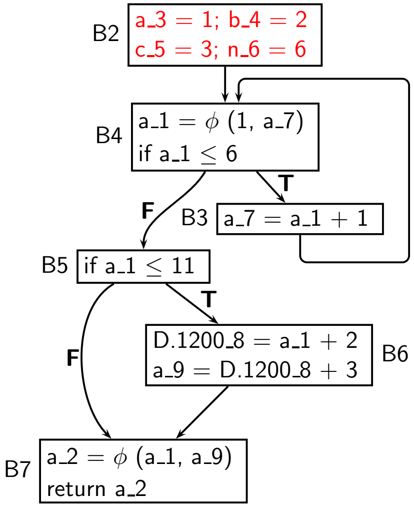
Outline

- Motivation
- Live Variables Analysis
- Available Expressions Analysis

Part 2

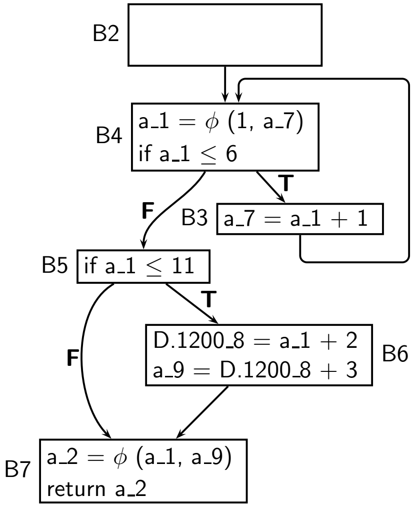
Motivation

Dead Code Elimination



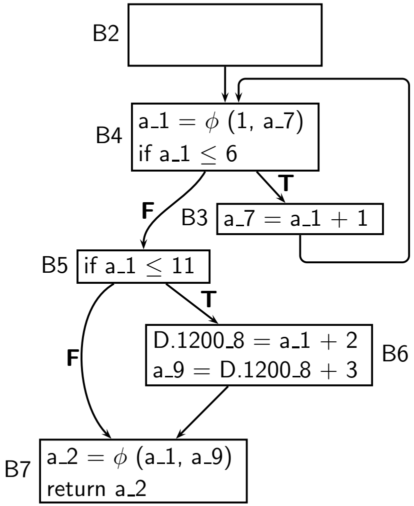
- No uses for variables a_3, b_4, c_5, and n_6

Dead Code Elimination



- No uses for variables a_3, `b_4`, `c_5`, and `n_6`

Dead Code Elimination

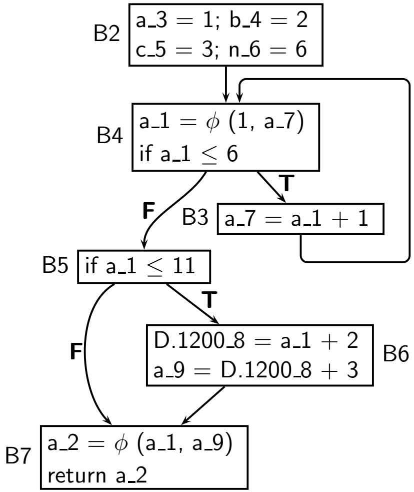


- No uses for variables a_3, b_4, c_5, and n_6
- Assignments to these variables can be deleted

How can we conclude this systematically?

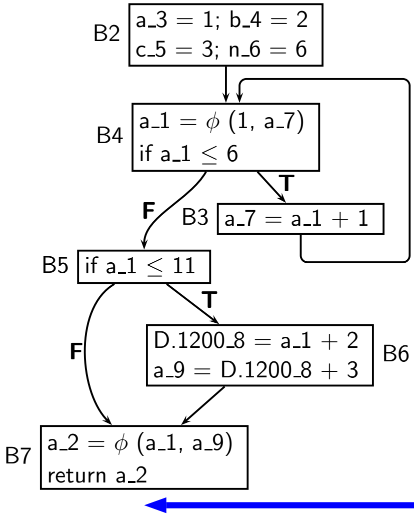
Liveness Analysis of Variables

Find out at each program point *p*, the variables that are used beyond *p*



Liveness Analysis of Variables

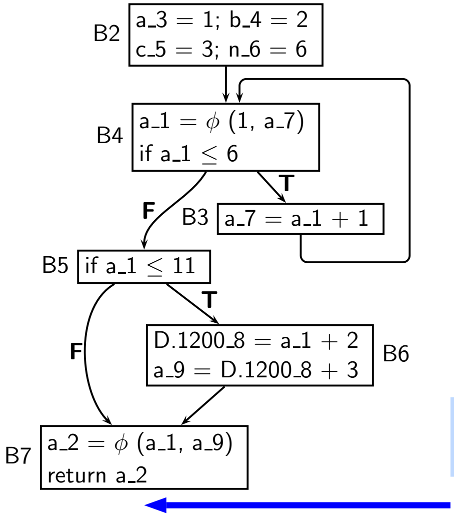
Find out at each program point *p*, the variables that are used beyond *p*



Which variables are used beyond this point?

Liveness Analysis of Variables

Find out at each program point *p*, the variables that are used beyond *p*

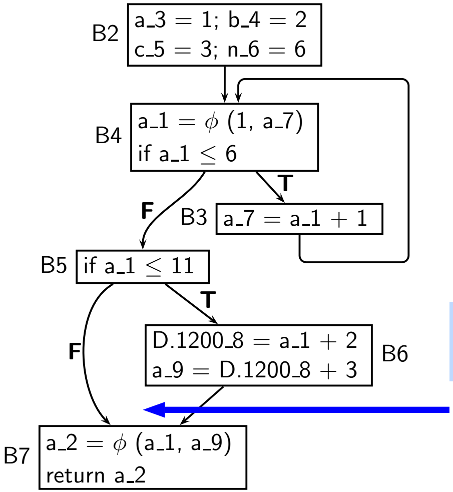


Which variables are used beyond this point?

$\emptyset$

Liveness Analysis of Variables

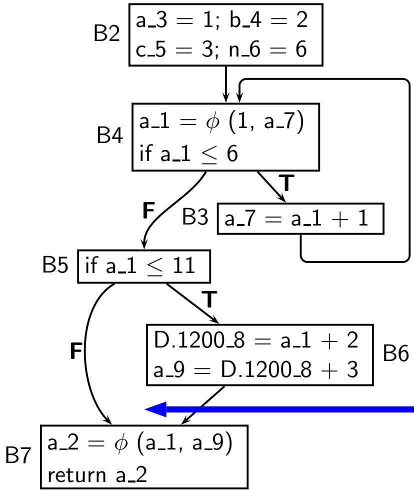
Find out at each program point *p*, the variables that are used beyond *p*



Which variables are used beyond this point?

Liveness Analysis of Variables

Find out at each program point *p*, the variables that are used beyond *p*

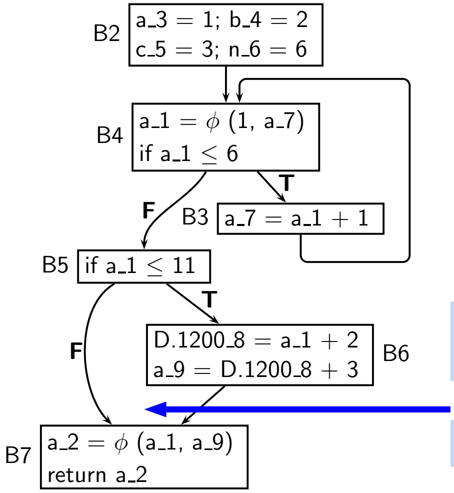


Which variables are used beyond this point?

{a_1, a_9}

Liveness Analysis of Variables

Find out at each program point *p*, the variables that are used beyond *p*



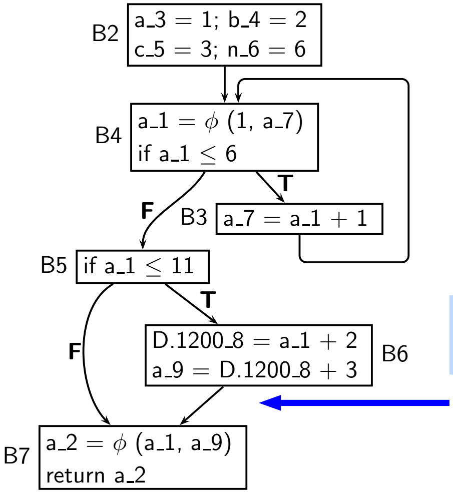
Which variables are used beyond this point?

{a_1, a_9}

What about a_2?

Liveness Analysis of Variables

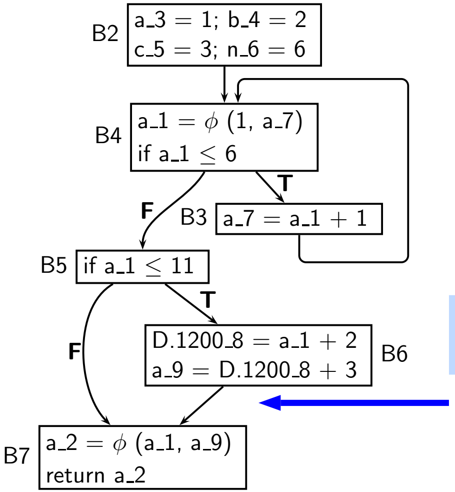
Find out at each program point *p*, the variables that are used beyond *p*



Which variables are used beyond this point?

Liveness Analysis of Variables

Find out at each program point *p*, the variables that are used beyond *p*

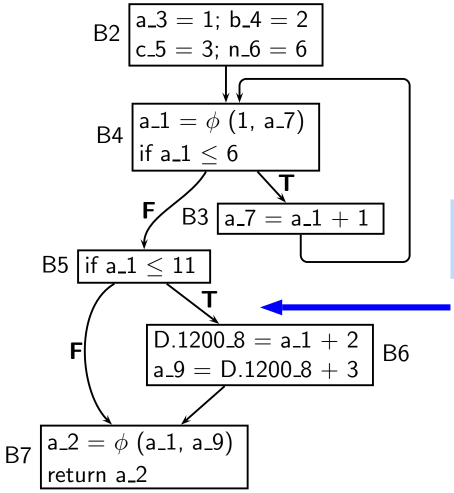


Which variables are used beyond this point?

{a_1, a_9}

Liveness Analysis of Variables

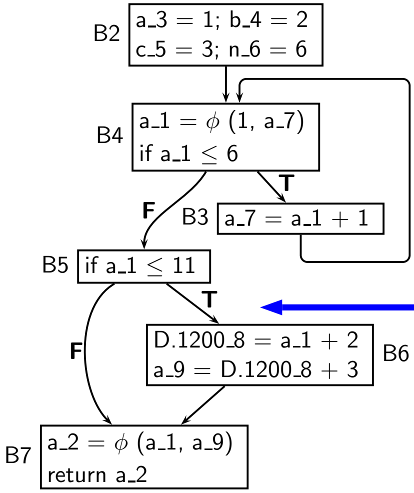
Find out at each program point *p*, the variables that are used beyond *p*



Which variables are used beyond this point?

Liveness Analysis of Variables

Find out at each program point *p*, the variables that are used beyond *p*

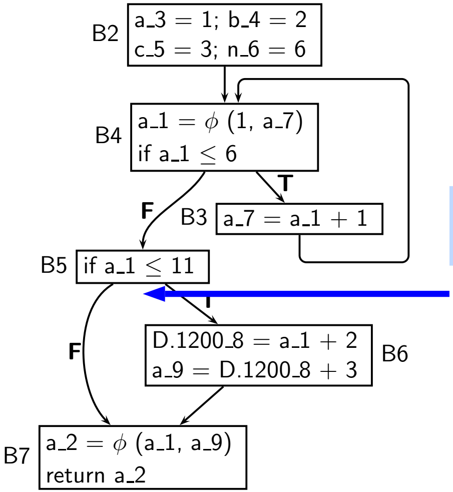


Which variables are used beyond this point?

{a_1}

Liveness Analysis of Variables

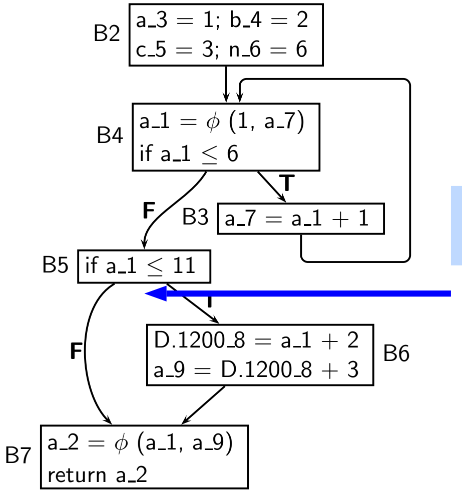
Find out at each program point *p*, the variables that are used beyond *p*



Which variables are used beyond this point?

Liveness Analysis of Variables

Find out at each program point *p*, the variables that are used beyond *p*

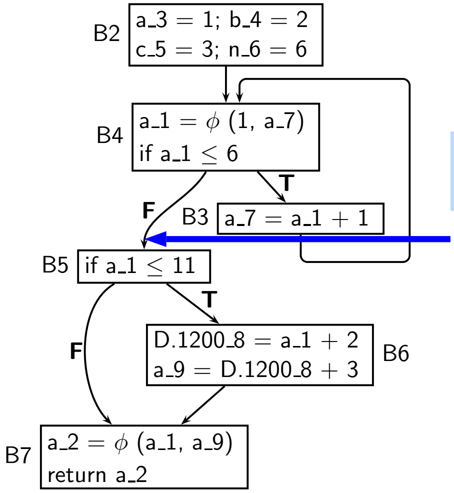


Which variables are used beyond this point?

{a_1, a_9}

Liveness Analysis of Variables

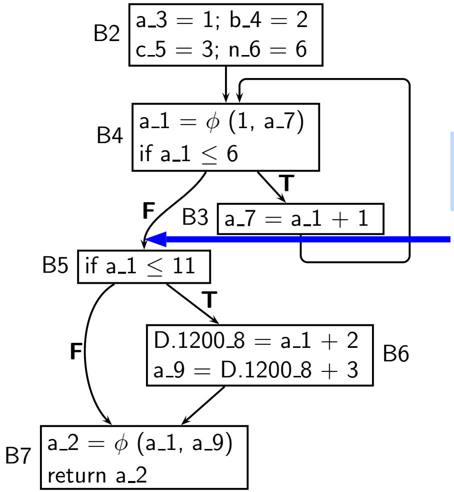
Find out at each program point *p*, the variables that are used beyond *p*



Which variables are used beyond this point?

Liveness Analysis of Variables

Find out at each program point *p*, the variables that are used beyond *p*

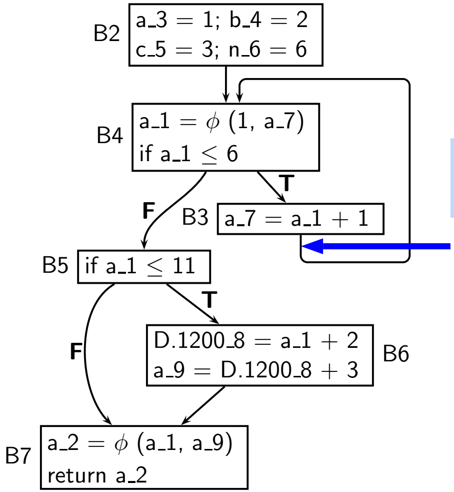


Which variables are used beyond this point?

{a_1, a_9}

Liveness Analysis of Variables

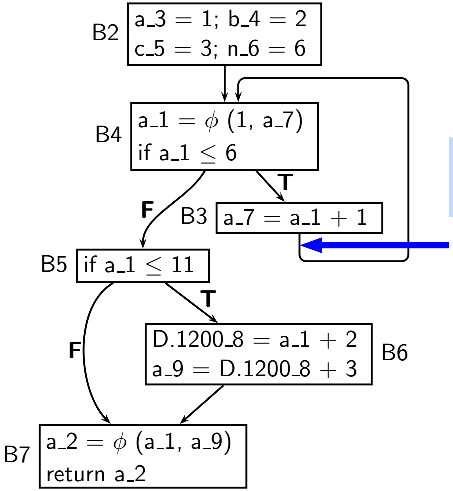
Find out at each program point *p*, the variables that are used beyond *p*



Which variables are used beyond this point?

Liveness Analysis of Variables

Find out at each program point *p*, the variables that are used beyond *p*

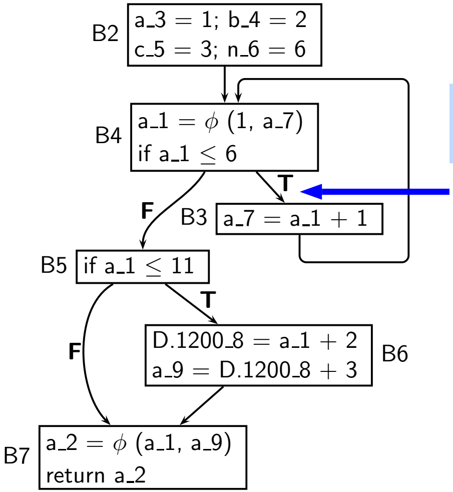


Which variables are used beyond this point?

∅ (Conservative assumption)

Liveness Analysis of Variables

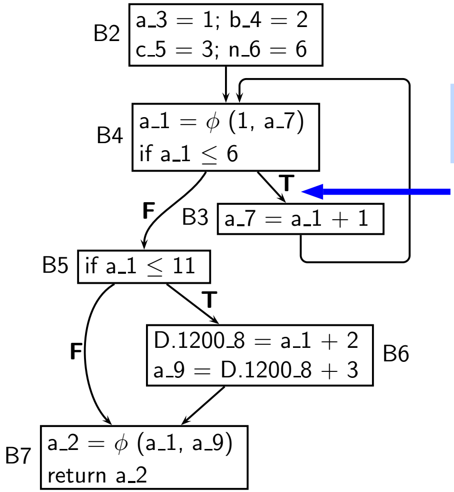
Find out at each program point *p*, the variables that are used beyond *p*



Which variables are used beyond this point?

Liveness Analysis of Variables

Find out at each program point *p*, the variables that are used beyond *p*

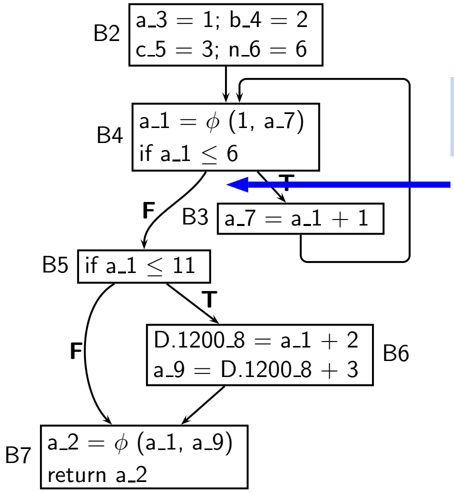


Which variables are used beyond this point?

{a_1}

Liveness Analysis of Variables

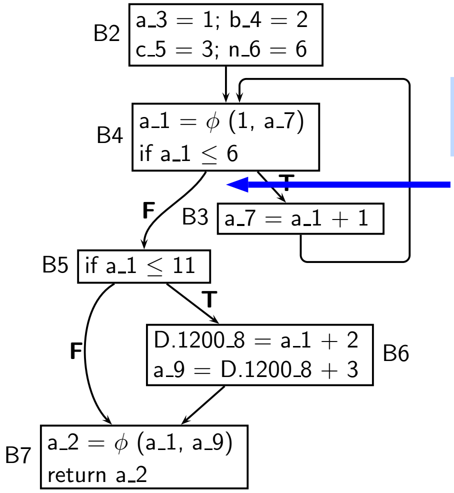
Find out at each program point *p*, the variables that are used beyond *p*



Which variables are used beyond this point?

Liveness Analysis of Variables

Find out at each program point *p*, the variables that are used beyond *p*

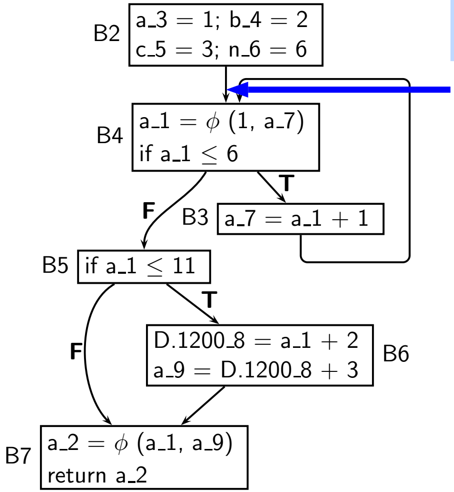


Which variables are used beyond this point?

{a_1, a_9}

Liveness Analysis of Variables

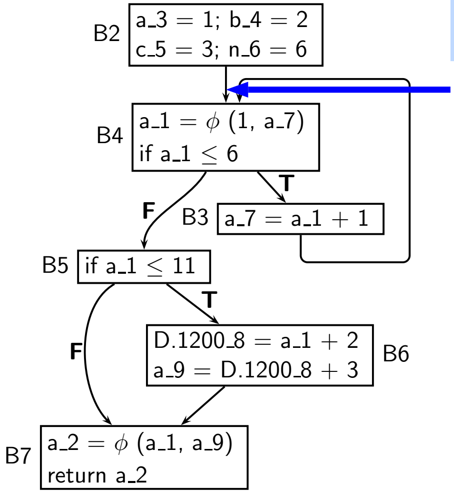
Find out at each program point *p*, the variables that are used beyond *p*



Which variables are used beyond this point?

Liveness Analysis of Variables

Find out at each program point *p*, the variables that are used beyond *p*

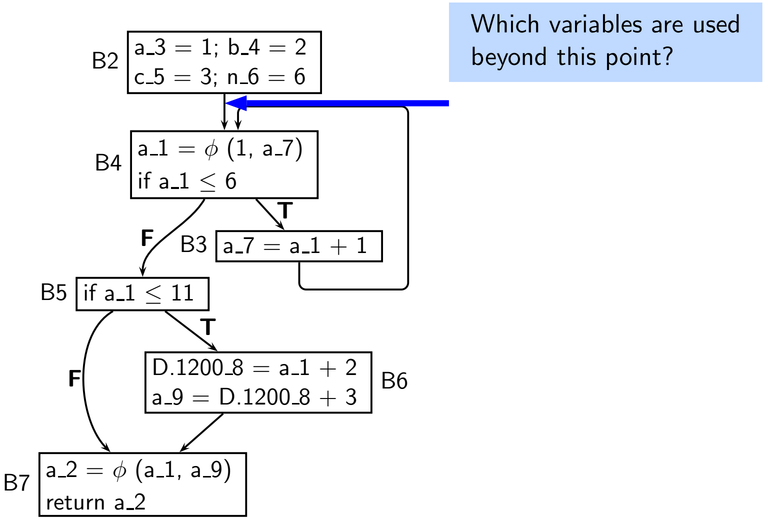


Which variables are used beyond this point?

{a_7, a_9}

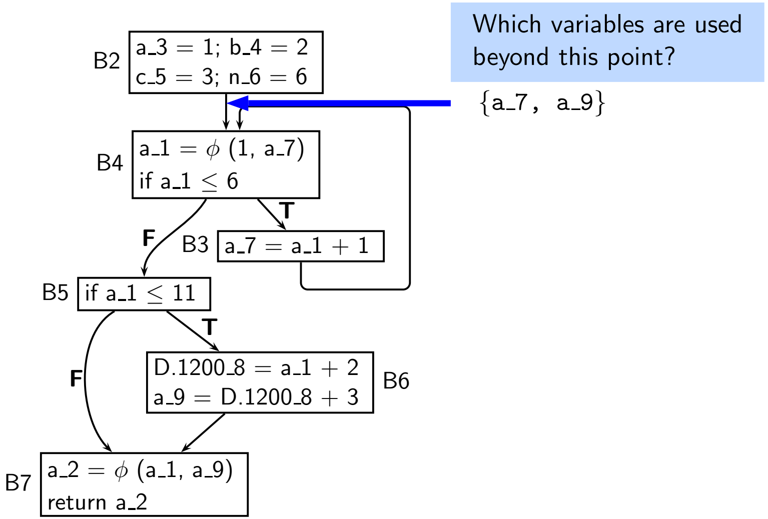
Liveness Analysis of Variables

Find out at each program point *p*, the variables that are used beyond *p*



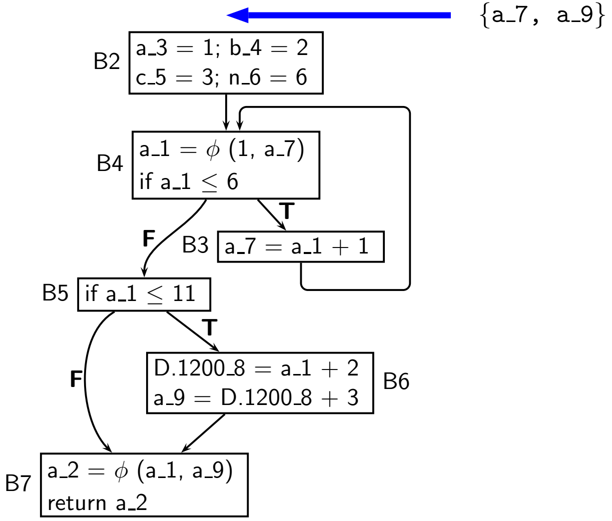
Liveness Analysis of Variables

Find out at each program point *p*, the variables that are used beyond *p*



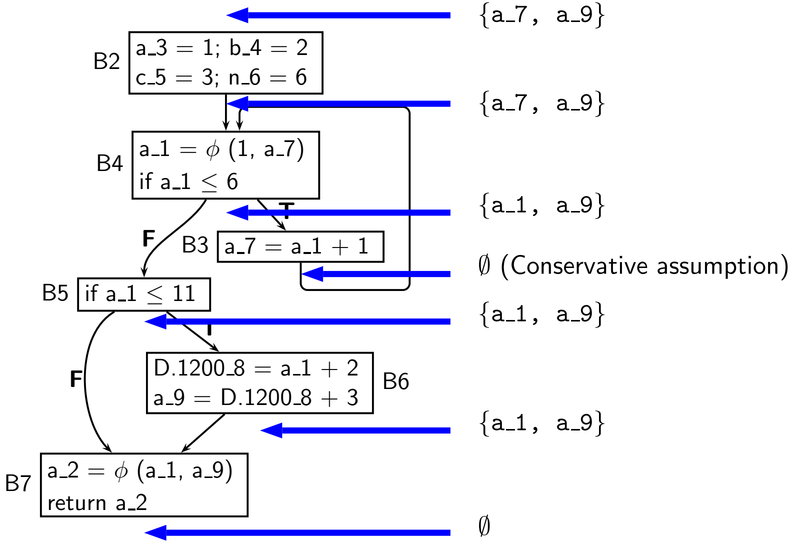
Liveness Analysis of Variables

Find out at each program point *p*, the variables that are used beyond *p*



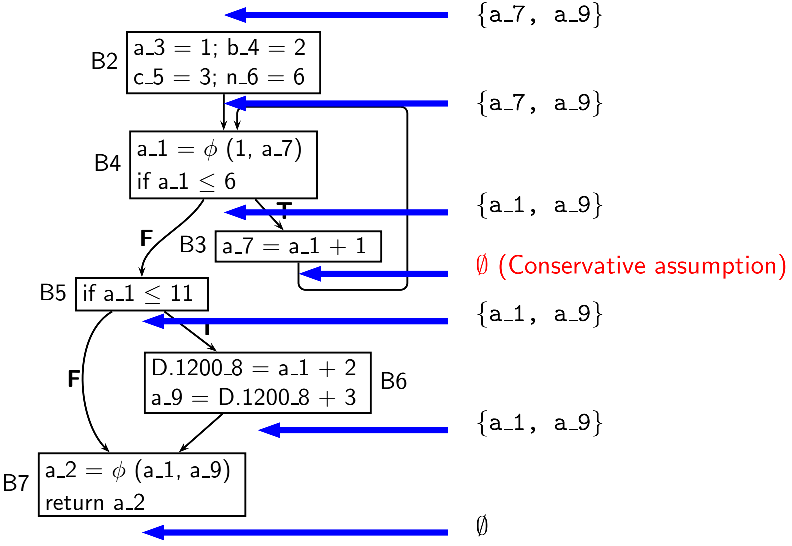
Liveness Analysis of Variables

Find out at each program point *p*, the variables that are used beyond *p*



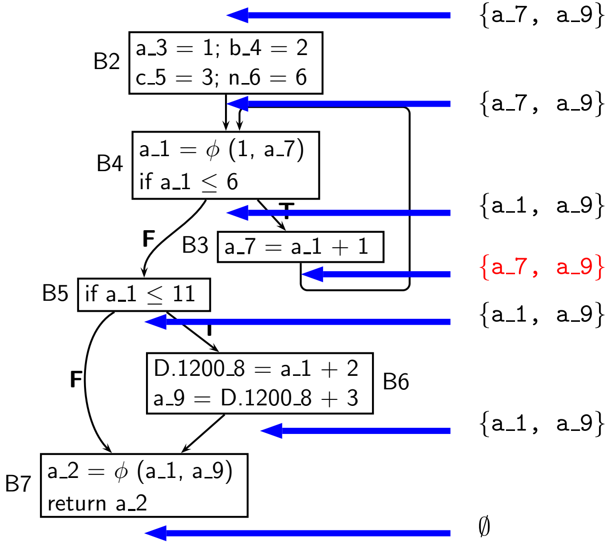
Liveness Analysis of Variables: Iteration 2

Find out at each program point *p*, the variables that are used beyond *p*

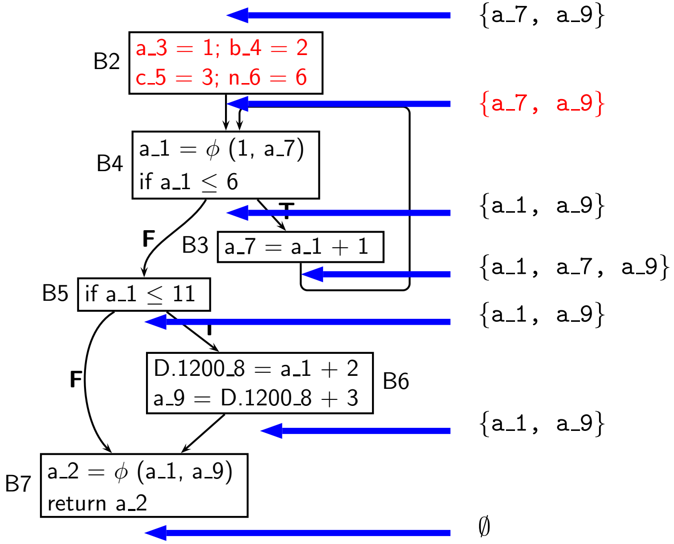


Liveness Analysis of Variables: Iteration 2

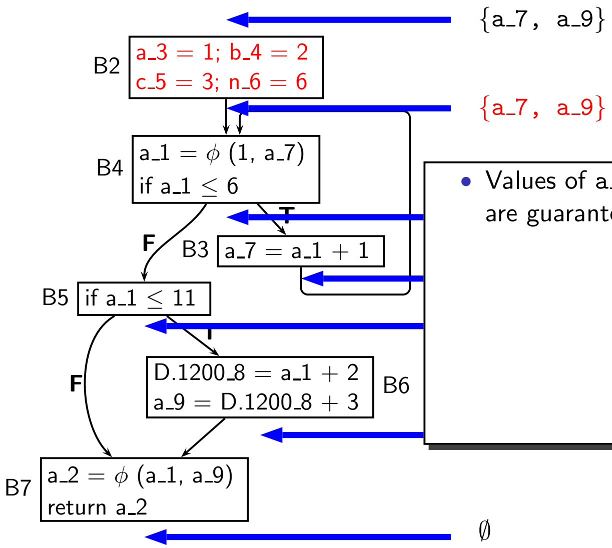
Find out at each program point *p*, the variables that are used beyond *p*



Using Liveness Analysis for Dead Code Elimination

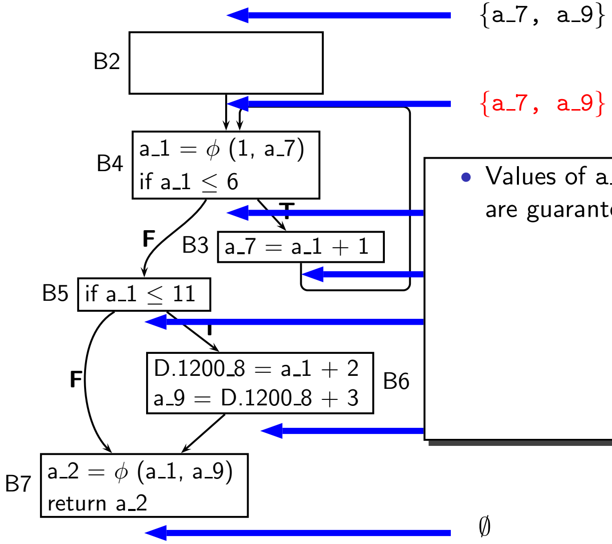


Using Liveness Analysis for Dead Code Elimination



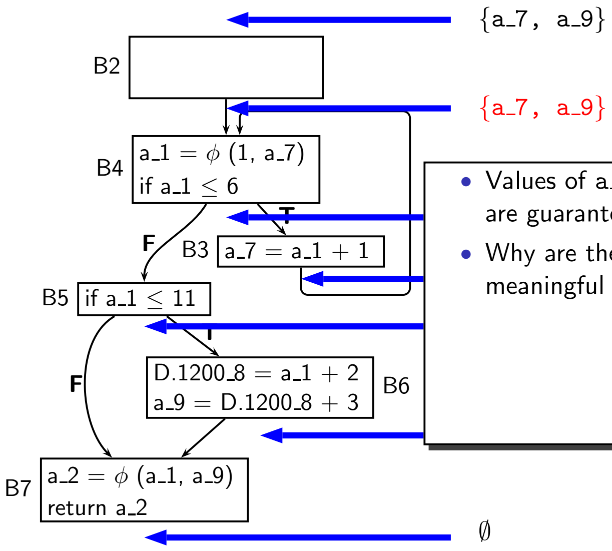
- Values of a_3, a_4, c_5, and n_6 are guaranteed not to be used

Using Liveness Analysis for Dead Code Elimination



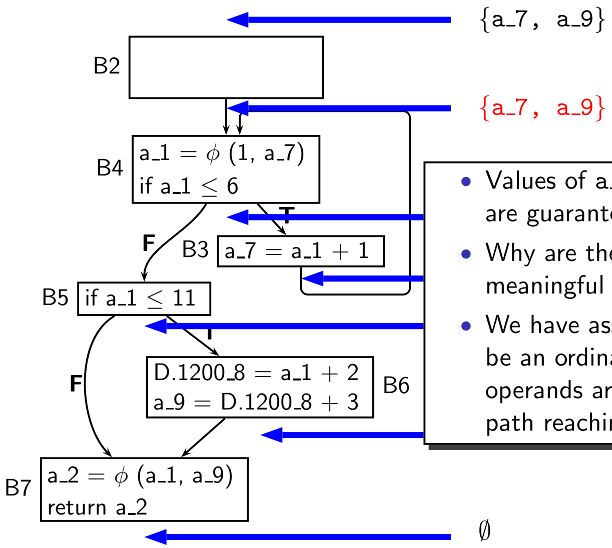
- Values of a_3, a_4, c_5, and n_6 are guaranteed not to be used

Using Liveness Analysis for Dead Code Elimination



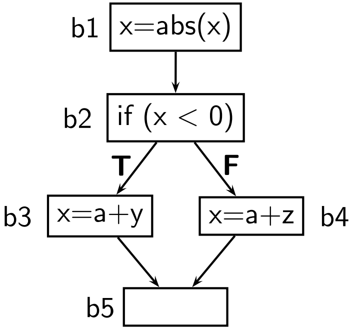
- Values of a_3, a_4, c_5, and n_6 are guaranteed not to be used
- Why are the values of a_7 and a_9 meaningful at the exit of B2?

Using Liveness Analysis for Dead Code Elimination

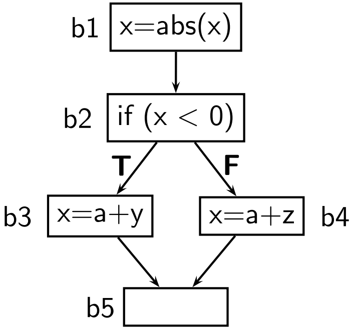


- Values of a_3, a_4, c_5, and n_6 are guaranteed not to be used
- Why are the values of a_7 and a_9 meaningful at the exit of B2?
- We have assumed a ϕ function to be an ordinary expression in which operands are computed along every path reaching the computation

Conservative Nature of Analysis (1)

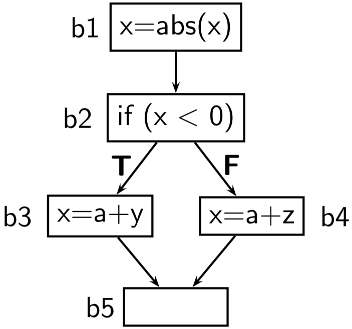


Conservative Nature of Analysis (1)



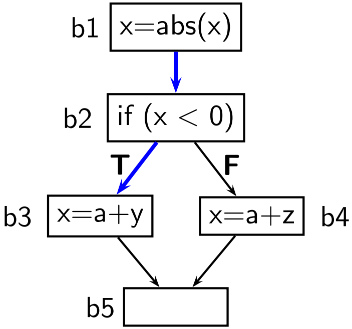
- abs(n) returns the absolute value of n

Conservative Nature of Analysis (1)



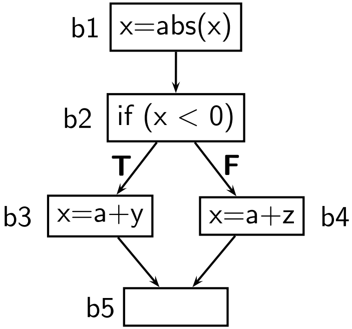
- abs(n) returns the absolute value of n
- Is y live on entry to block b2?

Conservative Nature of Analysis (1)



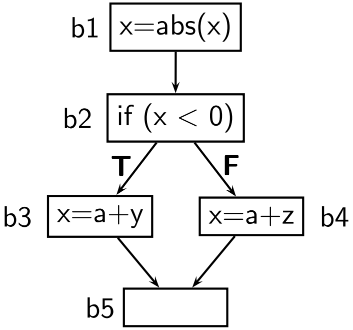
- abs(n) returns the absolute value of n
- Is y live on entry to block b2?
- By execution semantics, no
  Path b1→b2→b3 is an infeasible execution path

Conservative Nature of Analysis (1)



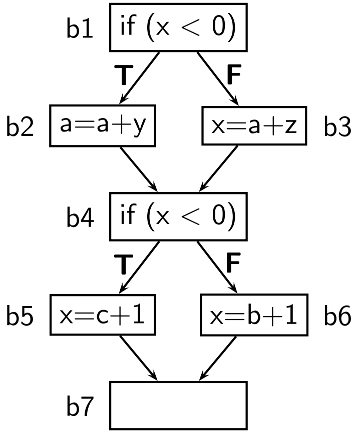
- abs(n) returns the absolute value of n
- Is y live on entry to block b2?
- By execution semantics, no
  Path b1→b2→b3 is an infeasible execution path
- A compiler make conservative assumptions: *All branch outcomes are possible*
  $\Rightarrow$ Consider every path in CFG as a potential execution execution path

Conservative Nature of Analysis (1)

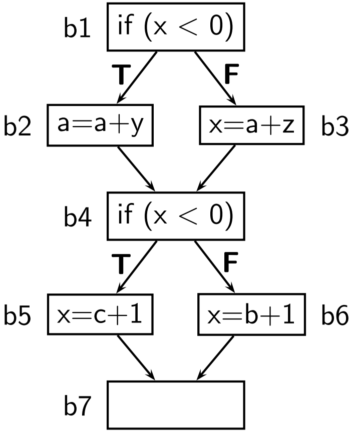


- abs(n) returns the absolute value of n
- Is y live on entry to block b2?
- By execution semantics, no
  Path b1→b2→b3 is an infeasible execution path
- A compiler make conservative assumptions: *All branch outcomes are possible*
  ⇒ Consider every path in CFG as a potential execution execution path
- Our analysis concludes that y is live on entry to block b2

Conservative Nature of Analysis (2)

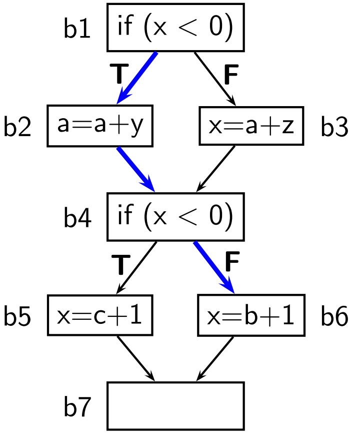


Conservative Nature of Analysis (2)



- Is b live on entry to block b2?

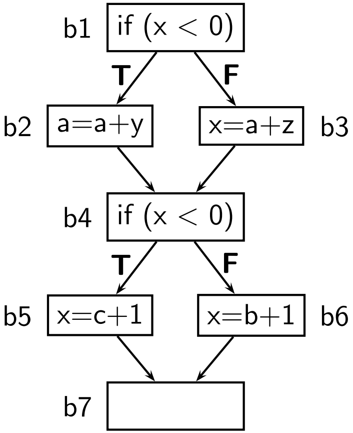
Conservative Nature of Analysis (2)



- 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)

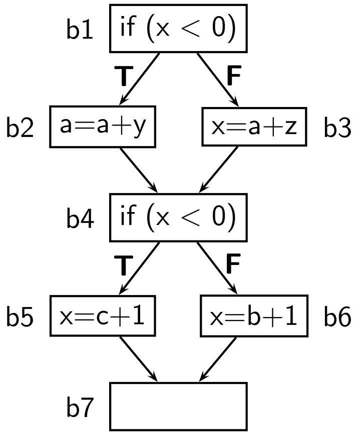


- Is b live on entry to block b2?
- By execution semantics, no

  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)



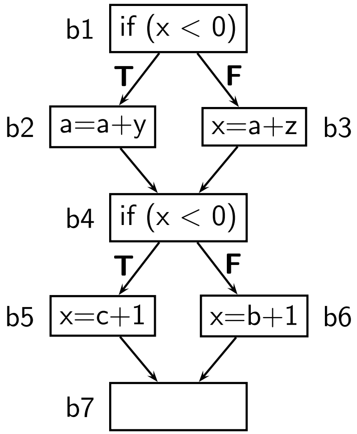
- Is b live on entry to block b2?
- By execution semantics, no

  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 ⇒ 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*

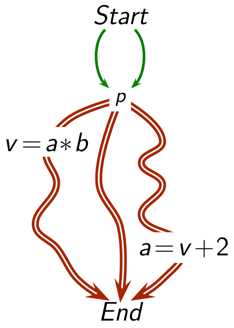
  Note: It is *flow sensitive* (i.e. information is computed for every control flow points)
- Our analysis concludes that b is live at the entry of b2 and c is live at the entry of b3

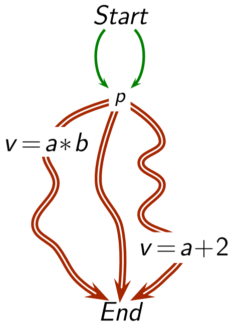
Part 3

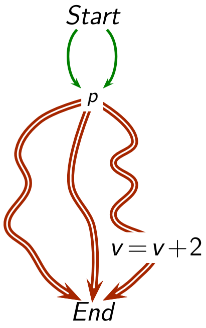
Live Variables Analysis

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.



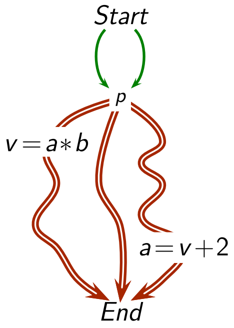


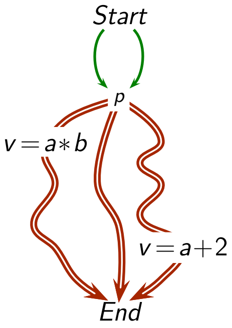


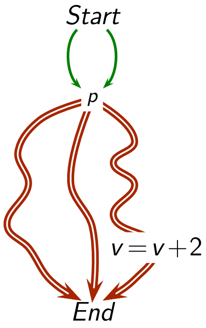
Defining Live Variables Analysis

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





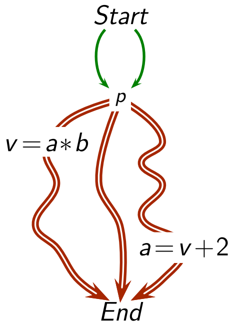


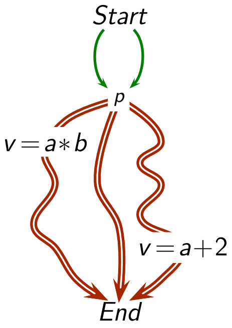
Defining Live Variables Analysis

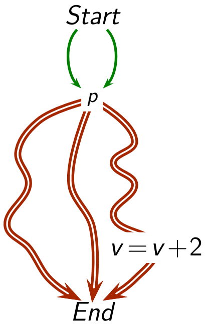
A variable v is live at a program point p, if some path from p to program exit contains an r-value occurrence of v which is not preceded by an l-value occurrence of v.

v is live at p

v is not live at p

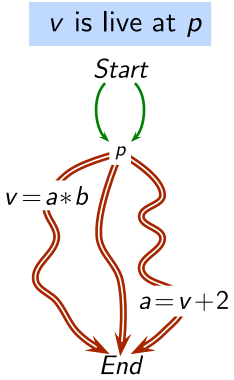


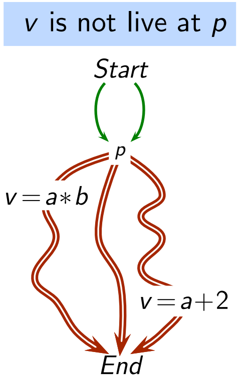


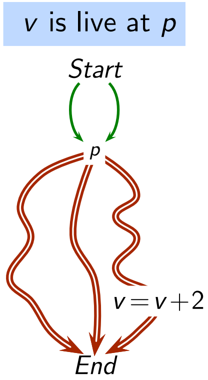


Defining Live Variables Analysis

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.





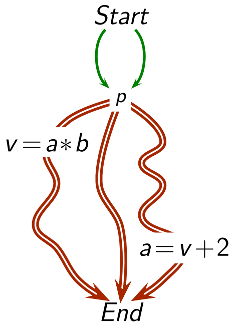


Defining Live Variables Analysis

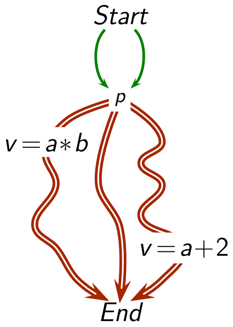
A variable v is live at a program point p, if some path from p to program exit contains an r-value occurrence of v which is not preceded by an l-value occurrence of v.

Path based specification

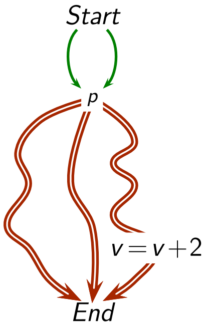
v is live at p



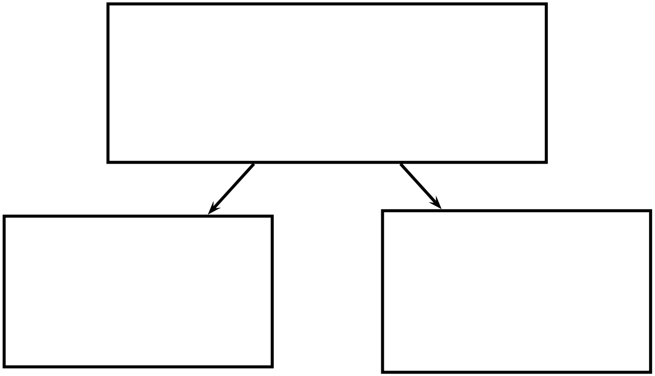
v is not live at p



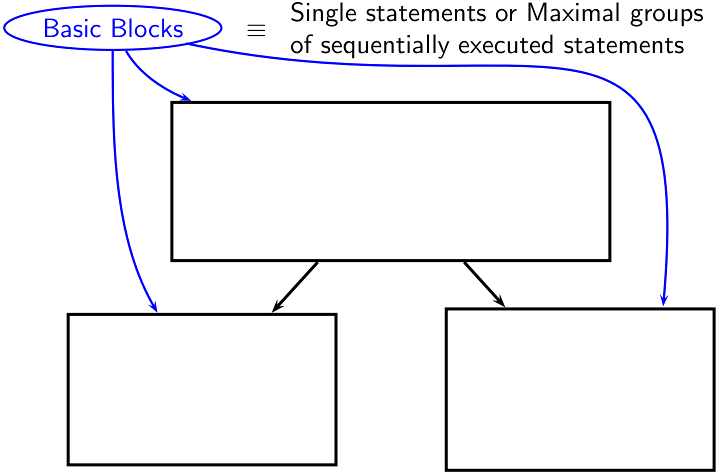
v is live at p



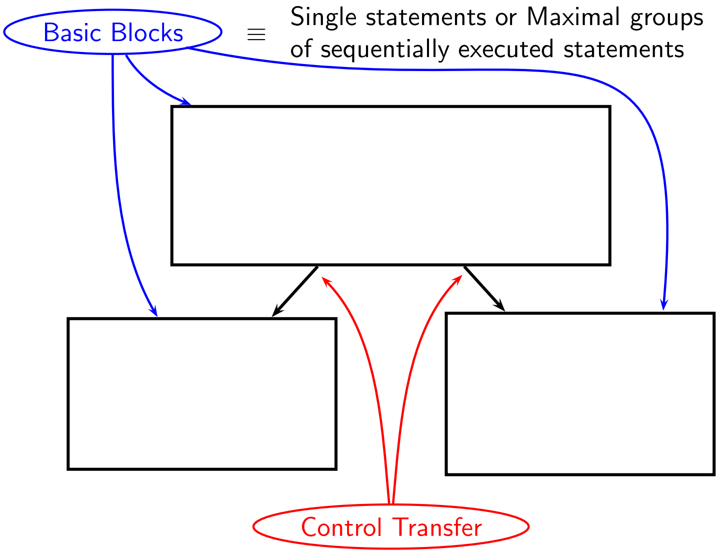
Defining Data Flow Analysis for Live Variables Analysis



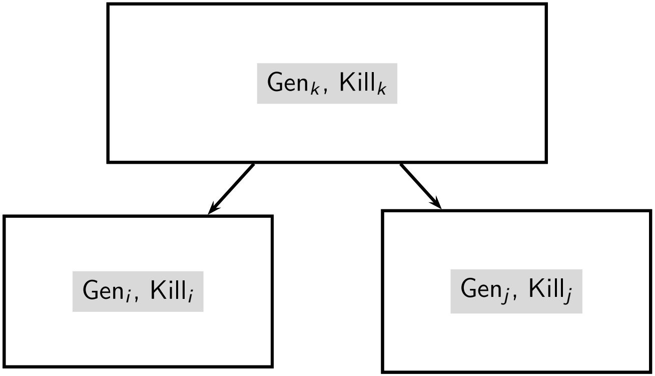
Defining Data Flow Analysis for Live Variables Analysis


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

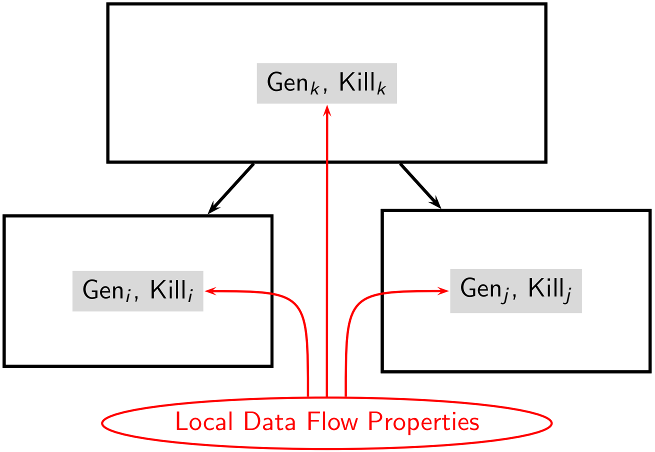
Defining Data Flow Analysis for Live Variables Analysis


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

Defining Data Flow Analysis for Live Variables Analysis


Gen_k, Kill_k
Gen_i, Kill_i
Gen_j, Kill_j

Defining Data Flow Analysis for Live Variables Analysis



Local Data Flow Properties for Live Variables Analysis

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

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

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

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`

l-value occurrence

Value is modified e.g. y in

`y = x.lptr`

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

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`

l-value occurrence

Value is modified e.g. y in

`y = x.lptr`

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

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`

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

within *n*

anywhere in *n*

Local Data Flow Properties for Live Variables Analysis

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

Local Data Flow Properties for Live Variables Analysis

- 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

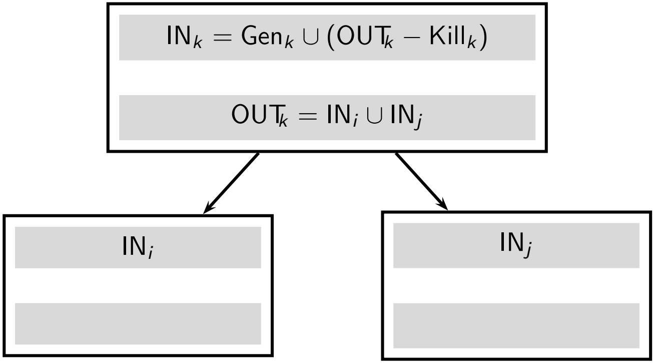
Local Data Flow Properties for Live Variables Analysis

Case	Local Information		Effect on Liveness
1	$v \notin \text{Gen}_n$	$v \notin \text{Kill}_n$	
2	$v \in \text{Gen}_n$	$v \notin \text{Kill}_n$	
3	$v \notin \text{Gen}_n$	$v \in \text{Kill}_n$	
4	$v \in \text{Gen}_n$	$v \in \text{Kill}_n$	

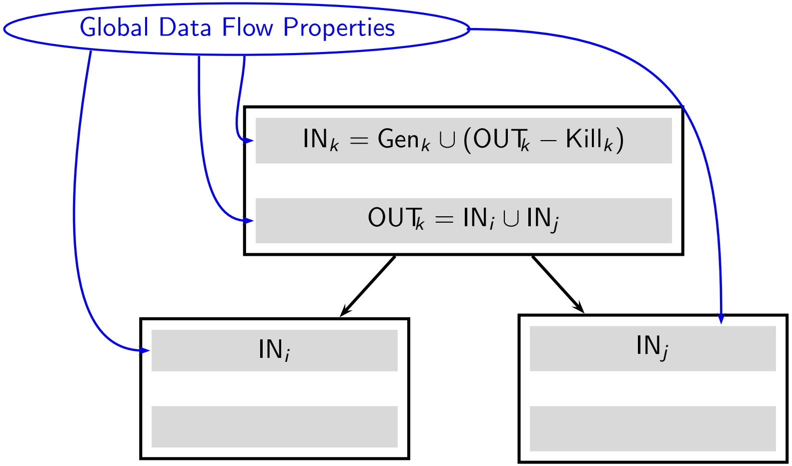
Local Data Flow Properties for Live Variables Analysis

Case	Local Information		Effect on Liveness
1	$v \notin \text{Gen}_n$	$v \notin \text{Kill}_n$	Liveness of v is unaffected in block n
2	$v \in \text{Gen}_n$	$v \notin \text{Kill}_n$	Liveness of v is generated in block n
3	$v \notin \text{Gen}_n$	$v \in \text{Kill}_n$	Liveness of v is killed in block n
4	$v \in \text{Gen}_n$	$v \in \text{Kill}_n$	Liveness of v is killed in block n but is re-generated in the same block

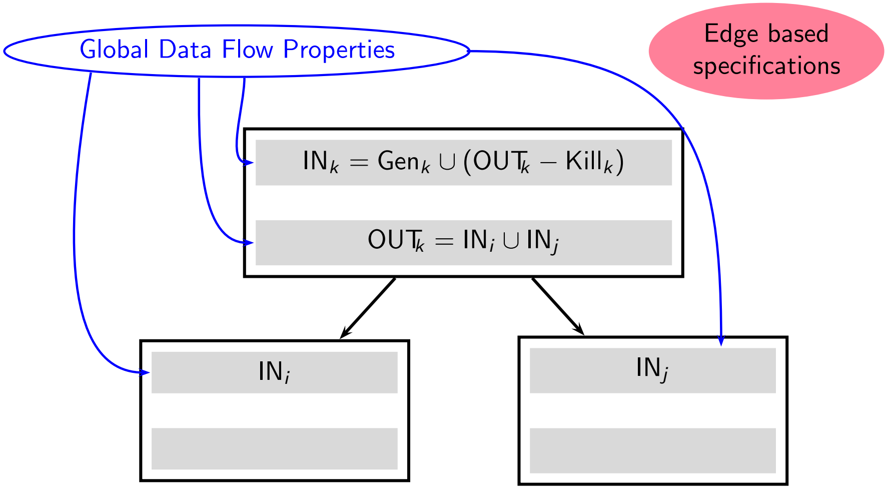
Defining Data Flow Analysis for Live Variables Analysis


$\text{IN}_k = \text{Gen}_k \cup (\text{OUT}_k - \text{Kill}_k)$
$\text{OUT}_k = \text{IN}_i \cup \text{IN}_j$
IN_i
IN_j

Defining Data Flow Analysis for Live Variables Analysis


Global Data Flow Properties
$\text{IN}_k = \text{Gen}_k \cup (\text{OUT}_k - \text{Kill}_k)$
$\text{OUT}_k = \text{IN}_i \cup \text{IN}_j$
IN_i
IN_j

Defining Data Flow Analysis for Live Variables Analysis


Global Data Flow Properties
Edge based specifications
$\text{IN}_k = \text{Gen}_k \cup (\text{OUT}_k - \text{Kill}_k)$
$\text{OUT}_k = \text{IN}_i \cup \text{IN}_j$
IN_i
IN_j

Data Flow Equations For Live Variables Analysis

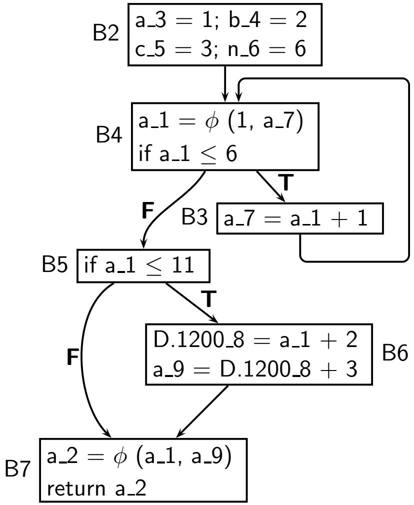
$$\begin{aligned}
\mathsf{IN}_n &= (\mathsf{OUT}_n - \mathsf{Kill}_n) \cup \mathsf{Gen}_n \\
\mathsf{OUT}_n &= \begin{cases} BI & n \text{ is } \textit{End} \text{ block} \\ \displaystyle\bigcup_{s \in succ(n)} \mathsf{IN}_s & \text{otherwise} \end{cases}
\end{aligned}$$

Data Flow Equations For Live Variables Analysis

$$
\begin{aligned}
\mathsf{IN}_n &= (\mathsf{OUT}_n - \mathsf{Kill}_n) \cup \mathsf{Gen}_n \\
\mathsf{OUT}_n &= \begin{cases} BI & n \text{ is } \textit{End} \text{ block} \\ \displaystyle\bigcup_{s \in succ(n)} \mathsf{IN}_s & \text{otherwise} \end{cases}
\end{aligned}
$$

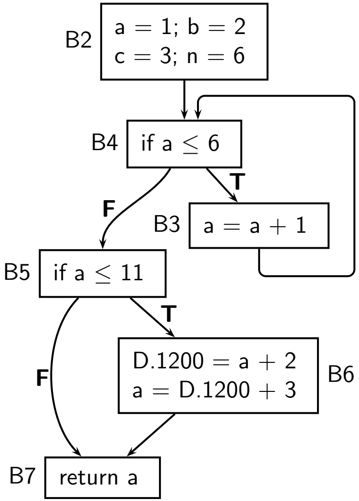
IN_n and OUT_n are sets of variables.

Performing Live Variables Analysis



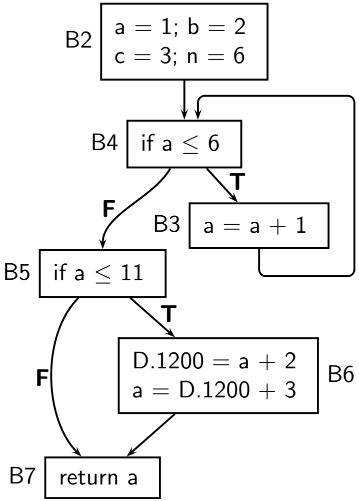
	Gen	Kill
B2	$\emptyset$	{a_3, b_4, c_5, n_6}
B4	{a_7}	{a_1}
B3	{a_1}	{a_7}
B5	{a_1}	$\emptyset$
B6	{a_1}	{a_9}
B7	{a_1, a_9}	{a_2}

Tutorial Problem for Live Variables Analysis



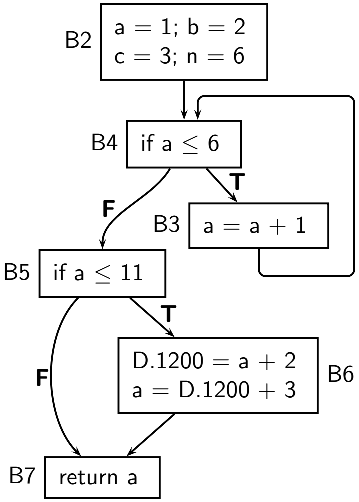
	Gen	Kill	IN	OUT
B2				
B4				
B3				
B5				
B6				
B7				

Tutorial Problem for Live Variables Analysis



	Gen	Kill	IN	OUT
B2	$\emptyset$	{a,b,c,n}		
B4	{a}	$\emptyset$		
B3	{a}	{a}		
B5	{a}	$\emptyset$		
B6	{a}	{a}		
B7	{a}	{a}		

Tutorial Problem for Live Variables Analysis



	Gen	Kill	IN	OUT
B2	$\emptyset$	{a,b,c,n}	$\emptyset$	{a}
B4	{a}	$\emptyset$	{a}	{a}
B3	{a}	{a}	{a}	{a}
B5	{a}	$\emptyset$	{a}	{a}
B6	{a}	{a}	{a}	{a}
B7	{a}	{a}	{a}	$\emptyset$

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.

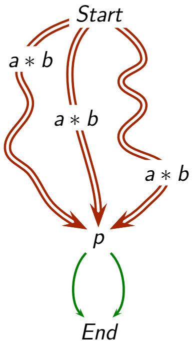
- Used for dead code elimination.
  If variable x is not live after an assignment $x = \ldots$, then the assginment is redundant and can be deleted as dead code.

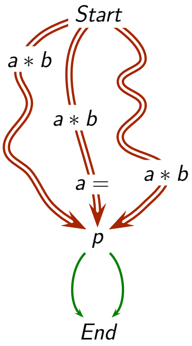
Part 4

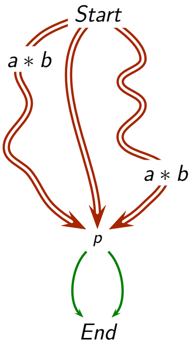
Available Expressions Analysis

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.



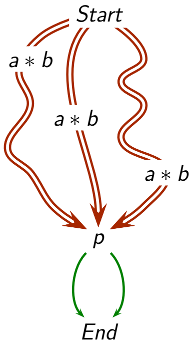


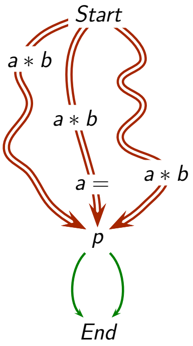


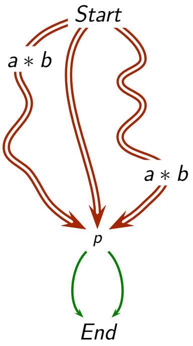
Defining Available Expressions Analysis

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.

$a * b$ is available at p

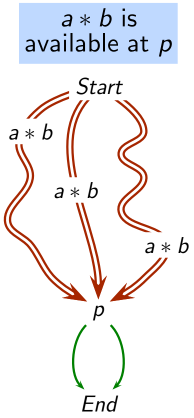


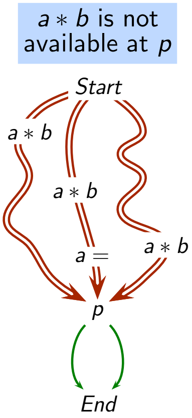


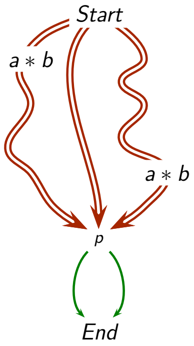


Defining Available Expressions Analysis

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.



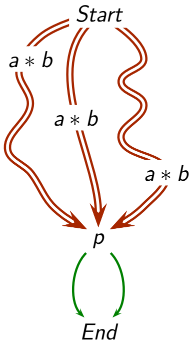




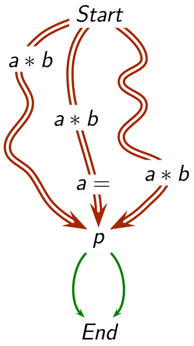
Defining Available Expressions Analysis

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.

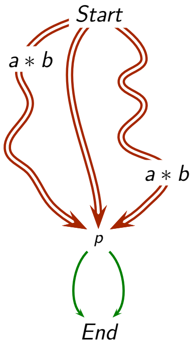
$a * b$ is available at p



$a * b$ is not available at p



$a * b$ is not available at p



Local Data Flow Properties for Available Expressions Analysis

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

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

	Entity	Manipulation	Exposition
Gen_n	Expression	Use	Downwards
Kill_n	Expression	Modification	Anywhere

Data Flow Equations For Available Expressions Analysis

$$\mathsf{IN}_n = \begin{cases} BI & n \text{ is } \mathit{Start} \text{ block} \\ \bigcap\limits_{p \in pred(n)} \mathsf{OUT}_p & \text{otherwise} \end{cases}$$

$$\mathsf{OUT}_n = \mathsf{Gen}_n \cup (\mathsf{IN}_n - \mathsf{Kill}_n)$$

Data Flow Equations For Available Expressions Analysis

$$\mathrm{IN}_n = \begin{cases} BI & n \text{ is } \mathit{Start} \text{ block} \\ \bigcap\limits_{p \in pred(n)} \mathrm{OUT}_p & \text{otherwise} \end{cases}$$

$$\mathrm{OUT}_n = \mathrm{Gen}_n \cup (\mathrm{IN}_n - \mathrm{Kill}_n)$$

Alternatively,

$$\mathrm{OUT}_n = f_n(\mathrm{IN}_n), \qquad \text{where}$$

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

Data Flow Equations For Available Expressions Analysis

$$\mathrm{IN}_n = \begin{cases} BI & n \text{ is } \textit{Start} \text{ block} \\ \bigcap\limits_{p \in pred(n)} \mathrm{OUT}_p & \text{otherwise} \end{cases}$$

$$\mathrm{OUT}_n = \mathrm{Gen}_n \cup (\mathrm{IN}_n - \mathrm{Kill}_n)$$

Alternatively,

$$\mathrm{OUT}_n = f_n(\mathrm{IN}_n), \quad \text{where}$$

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

IN_n and OUT_n are sets of expressions.

Using Data Flow Information of Available Expressions Analysis

- Common subsexpression elimination

Using Data Flow Information of Available Expressions Analysis

- Common subsexpression elimination
  - If an expression is available at the entry of a block *b* **and**

Using Data Flow Information of Available Expressions Analysis

- Common subsexpression 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 subsexpression 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 subsexpression 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 subsexpression 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

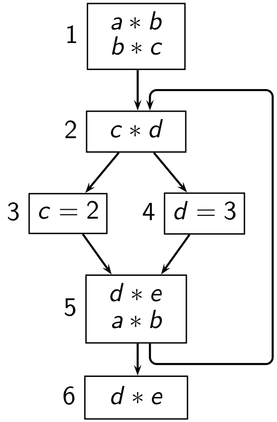
  Then the expression is redundant
- Redundant expression must be upwards exposed

Using Data Flow Information of Available Expressions Analysis

- Common subsexpression 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
- Redundant expression must be upwards exposed
- Expressions in Gen_n are downwards exposed

An Example of Available Expressions Analysis

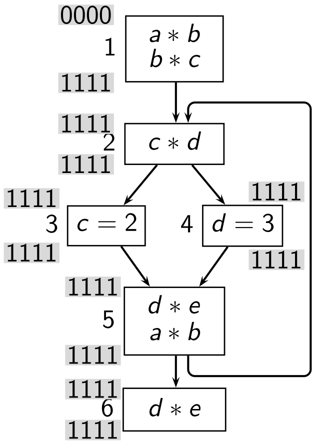


Let $e_1 \equiv a * b$, $e_2 \equiv b * c$, $e_3 \equiv c * d$, $e_4 \equiv d * e$

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

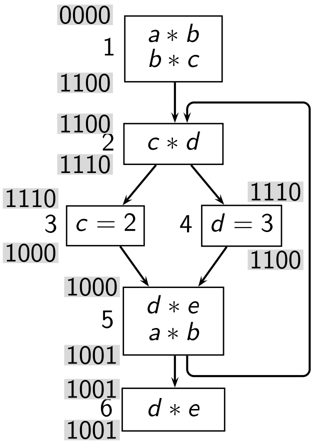


Let $e_1 \equiv a * b$, $e_2 \equiv b * c$, $e_3 \equiv c * d$, $e_4 \equiv d * e$

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

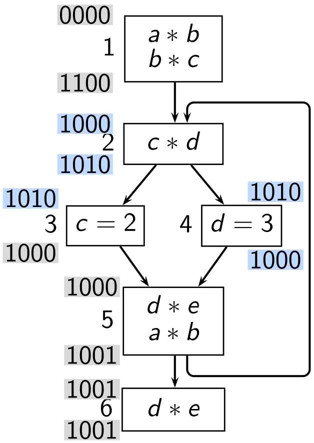


Let $e_1 \equiv a * b$, $e_2 \equiv b * c$, $e_3 \equiv c * d$, $e_4 \equiv d * e$

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

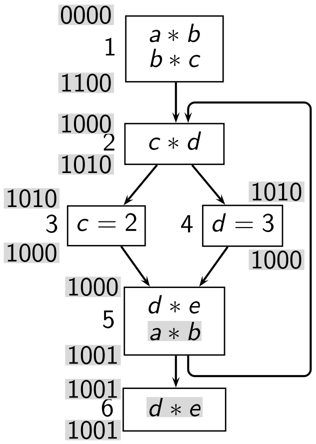


Let $e_1 \equiv a * b$, $e_2 \equiv b * c$, $e_3 \equiv c * d$, $e_4 \equiv d * e$

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