

CS738: Advanced Compiler Optimizations

Data Flow Analysis

Amey Karkare

karkare@cse.iitk.ac.in

<http://www.cse.iitk.ac.in/~karkare/cs738>

Department of CSE, IIT Kanpur



Agenda

- ▶ *Intraprocedural* Data Flow Analysis: Classical Examples

Agenda

- ▶ *Intraprocedural* Data Flow Analysis: Classical Examples
 - ▶ Last lecture: Reaching Definitions

Agenda

- ▶ *Intraprocedural* Data Flow Analysis: Classical Examples
 - ▶ Last lecture: Reaching Definitions
 - ▶ Today: Available Expressions

Agenda

- ▶ *Intraprocedural* Data Flow Analysis: Classical Examples
 - ▶ Last lecture: Reaching Definitions
 - ▶ Today: Available Expressions
 - ▶ Discussion about the similarities/differences

Available Expressions Analysis

- ▶ An expression e is available at a point p if

Available Expressions Analysis

- ▶ An expression e is available at a point p if
 - ▶ **Every** path from the *Entry* to p has at least one evaluation of e

Available Expressions Analysis

- ▶ An expression e is available at a point p if
 - ▶ **Every** path from the *Entry* to p has at least one evaluation of e
 - ▶ There is no assignment to any component variable of e **after the last evaluation** of e prior to p

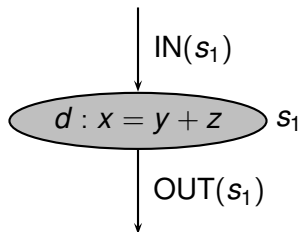
Available Expressions Analysis

- ▶ An expression e is available at a point p if
 - ▶ **Every** path from the *Entry* to p has at least one evaluation of e
 - ▶ There is no assignment to any component variable of e **after the last evaluation** of e prior to p
- ▶ Expression e is *generated* by its evaluation

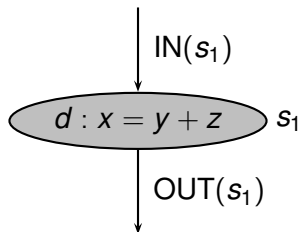
Available Expressions Analysis

- ▶ An expression e is available at a point p if
 - ▶ **Every** path from the *Entry* to p has at least one evaluation of e
 - ▶ There is no assignment to any component variable of e **after the last evaluation** of e prior to p
- ▶ Expression e is *generated* by its evaluation
- ▶ Expression e is *killed* by assignment to its component variables

AvE Analysis of a Structured Program

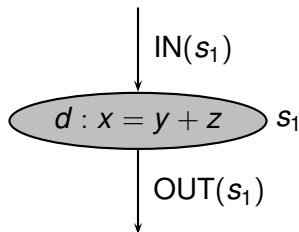


AvE Analysis of a Structured Program



$$OUT(s_1) = IN(s_1) - KILL(s_1) \cup GEN(s_1)$$

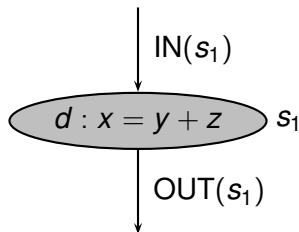
AvE Analysis of a Structured Program



$$OUT(s_1) = IN(s_1) - KILL(s_1) \cup GEN(s_1)$$

$$GEN(s_1) =$$

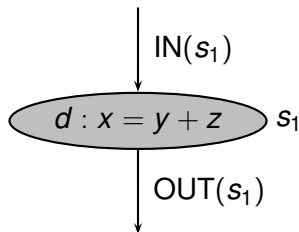
AvE Analysis of a Structured Program



$$OUT(s_1) = IN(s_1) - KILL(s_1) \cup GEN(s_1)$$

$$GEN(s_1) = \{y + z\}$$

AvE Analysis of a Structured Program

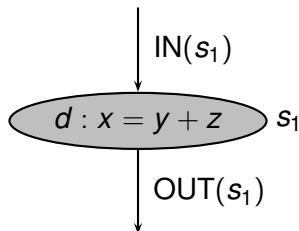


$$OUT(s_1) = IN(s_1) - KILL(s_1) \cup GEN(s_1)$$

$$GEN(s_1) = \{y + z\}$$

$$KILL(s_1) =$$

AvE Analysis of a Structured Program



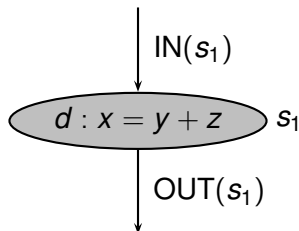
$$OUT(s_1) = IN(s_1) - KILL(s_1) \cup GEN(s_1)$$

$$GEN(s_1) = \{y + z\}$$

$$KILL(s_1) = E_x$$

where E_x : set of all expression having x as a component

AvE Analysis of a Structured Program



$$OUT(s_1) = IN(s_1) - KILL(s_1) \cup GEN(s_1)$$

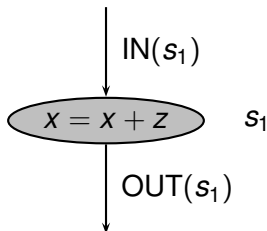
$$GEN(s_1) = \{y + z\}$$

$$KILL(s_1) = E_x$$

where E_x : set of all expression having x as a component

This may not work in general – WHY?

AvE Analysis of a Structured Program



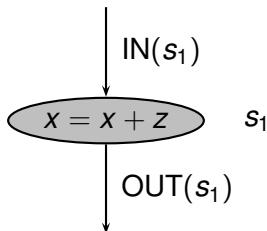
$$OUT(s_1) = IN(s_1) - KILL(s_1) \cup GEN(s_1)$$

$$GEN(s_1) = \{x + z\}$$

$$KILL(s_1) = E_x$$

Incorrectly marks $x + z$ as available after s_1

AvE Analysis of a Structured Program



$$OUT(s_1) = IN(s_1) - KILL(s_1) \cup GEN(s_1)$$

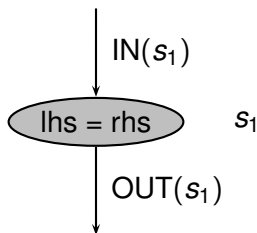
$$GEN(s_1) = \{x + z\}$$

$$KILL(s_1) = E_x$$

Incorrectly marks $x + z$ as available after s_1

$$GEN(s_1) = \emptyset \text{ for this case}$$

AvE Analysis of a Structured Program

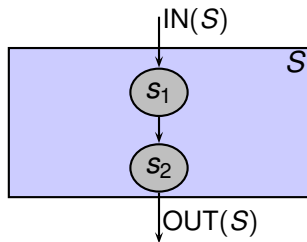


$$OUT(s_1) = IN(s_1) - KILL(s_1) \cup GEN(s_1)$$

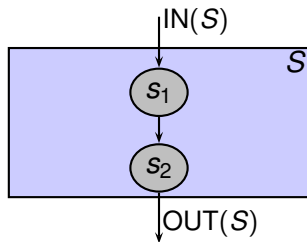
$$GEN(s_1) = \{rhs \mid lhs \text{ is not part of } rhs\}$$

$$KILL(s_1) = E_{lhs}$$

AvE Analysis of a Structured Program

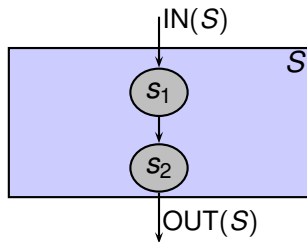


AvE Analysis of a Structured Program



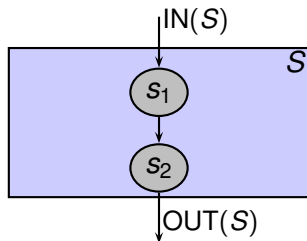
$GEN(S) =$

AvE Analysis of a Structured Program



$$GEN(S) = GEN(s_1) - KILL(s_2) \cup GEN(s_2)$$

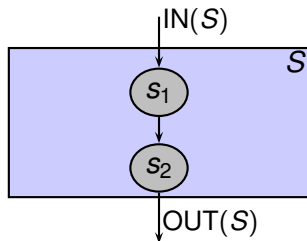
AvE Analysis of a Structured Program



$$GEN(S) = GEN(s_1) - KILL(s_2) \cup GEN(s_2)$$

$$KILL(S) =$$

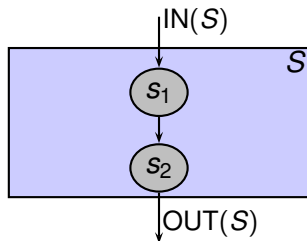
AvE Analysis of a Structured Program



$$GEN(S) = GEN(s_1) - KILL(s_2) \cup GEN(s_2)$$

$$KILL(S) = KILL(s_1) - GEN(s_2) \cup KILL(s_2)$$

AvE Analysis of a Structured Program

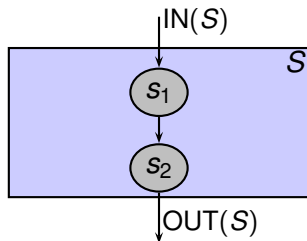


$$GEN(S) = GEN(s_1) - KILL(s_2) \cup GEN(s_2)$$

$$KILL(S) = KILL(s_1) - GEN(s_2) \cup KILL(s_2)$$

$$IN(s_1) =$$

AvE Analysis of a Structured Program

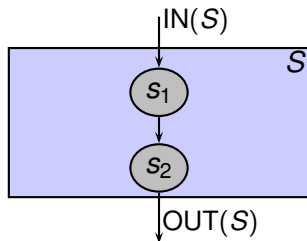


$$GEN(S) = GEN(s_1) - KILL(s_2) \cup GEN(s_2)$$

$$KILL(S) = KILL(s_1) - GEN(s_2) \cup KILL(s_2)$$

$$IN(s_1) = IN(S)$$

AvE Analysis of a Structured Program



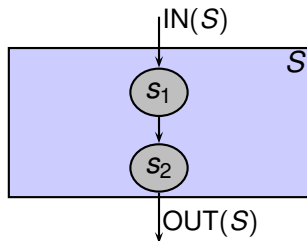
$$GEN(S) = GEN(s_1) - KILL(s_2) \cup GEN(s_2)$$

$$KILL(S) = KILL(s_1) - GEN(s_2) \cup KILL(s_2)$$

$$IN(s_1) = IN(S)$$

$$IN(s_2) =$$

AvE Analysis of a Structured Program



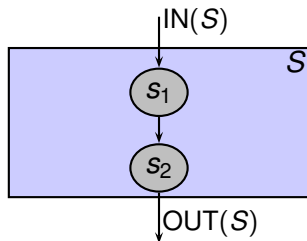
$$GEN(S) = GEN(s_1) - KILL(s_2) \cup GEN(s_2)$$

$$KILL(S) = KILL(s_1) - GEN(s_2) \cup KILL(s_2)$$

$$IN(s_1) = IN(S)$$

$$IN(s_2) = OUT(s_1)$$

AvE Analysis of a Structured Program



$$GEN(S) = GEN(s_1) - KILL(s_2) \cup GEN(s_2)$$

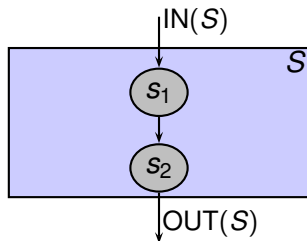
$$KILL(S) = KILL(s_1) - GEN(s_2) \cup KILL(s_2)$$

$$IN(s_1) = IN(S)$$

$$IN(s_2) = OUT(s_1)$$

$$OUT(S) =$$

AvE Analysis of a Structured Program



$$GEN(S) = GEN(s_1) - KILL(s_2) \cup GEN(s_2)$$

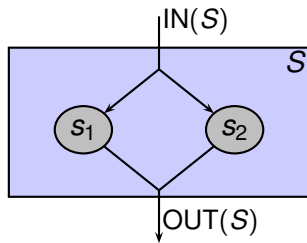
$$KILL(S) = KILL(s_1) - GEN(s_2) \cup KILL(s_2)$$

$$IN(s_1) = IN(S)$$

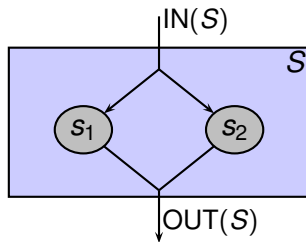
$$IN(s_2) = OUT(s_1)$$

$$OUT(S) = OUT(s_2)$$

AvE Analysis of a Structured Program

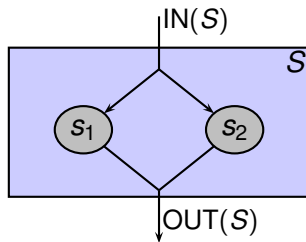


AvE Analysis of a Structured Program



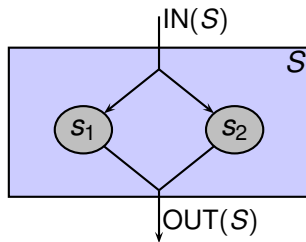
$GEN(S) =$

AvE Analysis of a Structured Program



$$GEN(S) = GEN(s_1) \cap GEN(s_2)$$

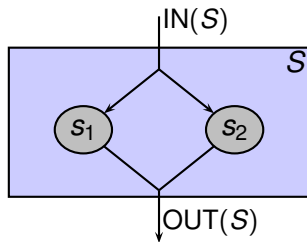
AvE Analysis of a Structured Program



$$GEN(S) = GEN(s_1) \cap GEN(s_2)$$

$$KILL(S) =$$

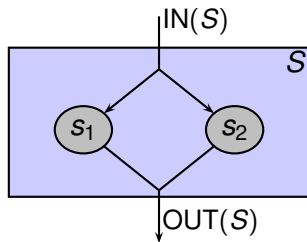
AvE Analysis of a Structured Program



$$GEN(S) = GEN(s_1) \cap GEN(s_2)$$

$$KILL(S) = KILL(s_1) \cup KILL(s_2)$$

AvE Analysis of a Structured Program

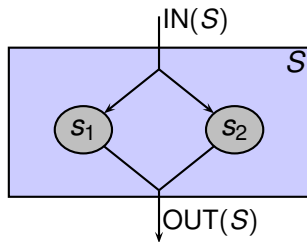


$$GEN(S) = GEN(s_1) \cap GEN(s_2)$$

$$KILL(S) = KILL(s_1) \cup KILL(s_2)$$

$$IN(s_1) =$$

AvE Analysis of a Structured Program

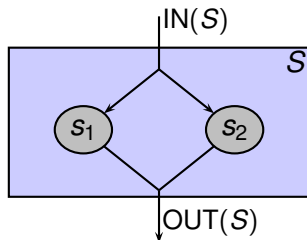


$$GEN(S) = GEN(s_1) \cap GEN(s_2)$$

$$KILL(S) = KILL(s_1) \cup KILL(s_2)$$

$$IN(s_1) = IN(s_2) = IN(S)$$

AvE Analysis of a Structured Program



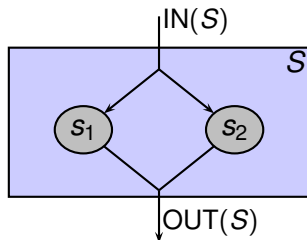
$$GEN(S) = GEN(s_1) \cap GEN(s_2)$$

$$KILL(S) = KILL(s_1) \cup KILL(s_2)$$

$$IN(s_1) = IN(s_2) = IN(S)$$

$$OUT(S) =$$

AvE Analysis of a Structured Program



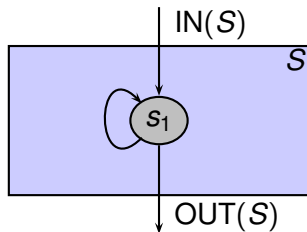
$$GEN(S) = GEN(s_1) \cap GEN(s_2)$$

$$KILL(S) = KILL(s_1) \cup KILL(s_2)$$

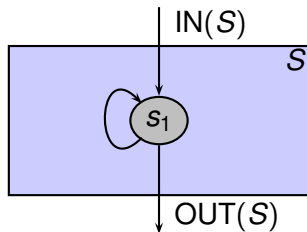
$$IN(s_1) = IN(s_2) = IN(S)$$

$$OUT(S) = OUT(s_1) \cap OUT(s_2)$$

AvE Analysis of a Structured Program

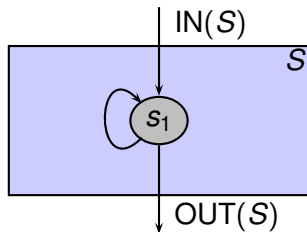


AvE Analysis of a Structured Program



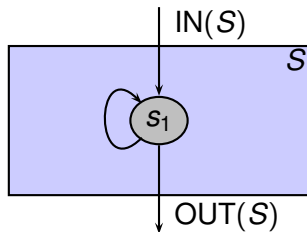
$GEN(S) =$

AvE Analysis of a Structured Program



$$GEN(S) = GEN(s_1)$$

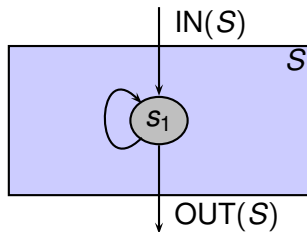
AvE Analysis of a Structured Program



$$GEN(S) = GEN(s_1)$$

$$KILL(S) =$$

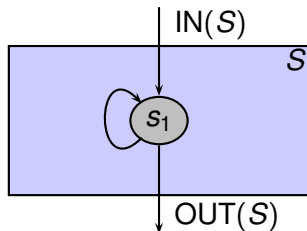
AvE Analysis of a Structured Program



$$GEN(S) = GEN(s_1)$$

$$KILL(S) = KILL(s_1)$$

AvE Analysis of a Structured Program

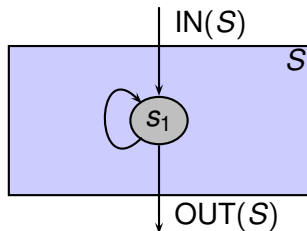


$$GEN(S) = GEN(s_1)$$

$$KILL(S) = KILL(s_1)$$

$$OUT(S) =$$

AvE Analysis of a Structured Program

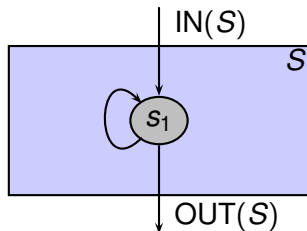


$$GEN(S) = GEN(s_1)$$

$$KILL(S) = KILL(s_1)$$

$$OUT(S) = OUT(s_1)$$

AvE Analysis of a Structured Program



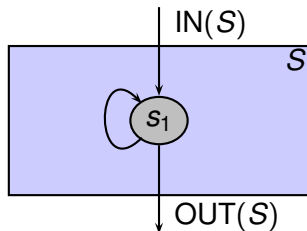
$$GEN(S) = GEN(s_1)$$

$$KILL(S) = KILL(s_1)$$

$$OUT(S) = OUT(s_1)$$

$$IN(s_1) =$$

AvE Analysis of a Structured Program



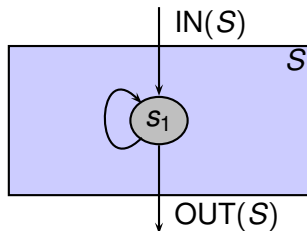
$$GEN(S) = GEN(s_1)$$

$$KILL(S) = KILL(s_1)$$

$$OUT(S) = OUT(s_1)$$

$$IN(s_1) = IN(S) \cap GEN(s_1)$$

AvE Analysis of a Structured Program



$$GEN(S) = GEN(s_1)$$

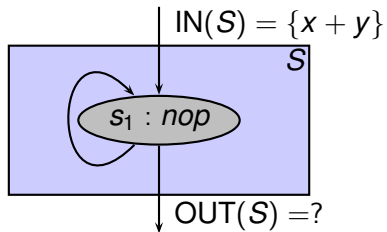
$$KILL(S) = KILL(s_1)$$

$$OUT(S) = OUT(s_1)$$

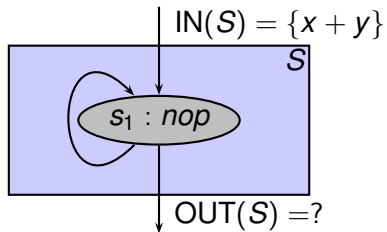
$$IN(s_1) = IN(S) \cap GEN(s_1) \text{ ?}$$

$$IN(s_1) = IN(S) \cap OUT(s_1) \text{ ??}$$

AvE Analysis of a Structured Program

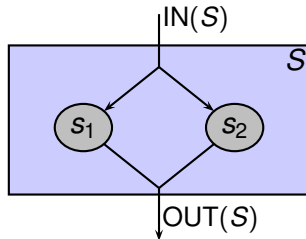


AvE Analysis of a Structured Program



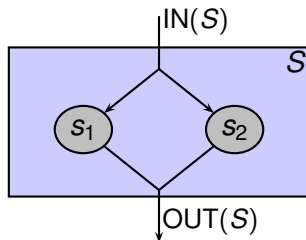
Is $x + y$ available at $\text{OUT}(S)$?

AvE Analysis is Approximate



- Assumption: All paths are feasible.

AvE Analysis is Approximate

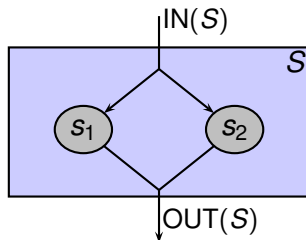


► Assumption: All paths are feasible.

► Example:

```
if (true) s1;  
else      s2;
```

AvE Analysis is Approximate

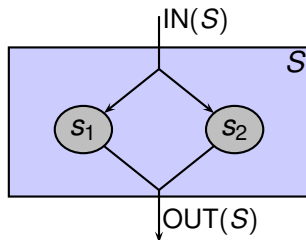


► Assumption: All paths are feasible.

► Example:

```
if (true) s1;  
else      s2;
```

AvE Analysis is Approximate

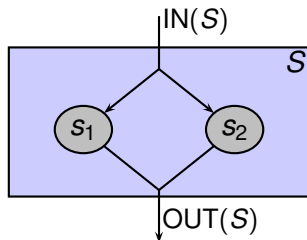


- ▶ Assumption: All paths are feasible.
- ▶ Example:

```
if (true) s1;  
else      s2;
```

$$\begin{array}{ccccc} \textbf{Fact} & & \textbf{Computed} & & \textbf{Actual} \\ \text{GEN}(S) & = & \text{GEN}(s_1) \cap \text{GEN}(s_2) & \subseteq & \text{GEN}(s_1) \end{array}$$

AvE Analysis is Approximate

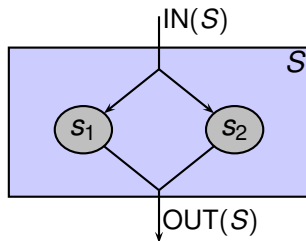


- Assumption: All paths are feasible.
- Example:

```
if (true) s1;  
else      s2;
```

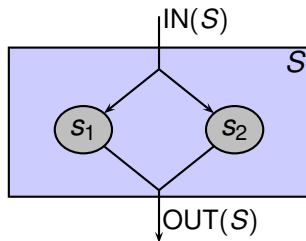
Fact	Computed	Actual
$GEN(S)$	$GEN(s_1) \cap GEN(s_2)$	$GEN(s_1)$
$KILL(S)$	$KILL(s_1) \cup KILL(s_2)$	$KILL(s_1)$

AvE Analysis is Approximate



► Thus,

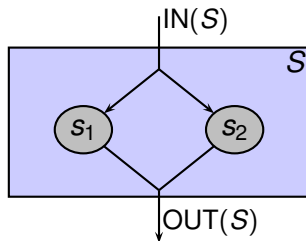
AvE Analysis is Approximate



► Thus,

$$\text{true GEN}(S) \supseteq \text{analysis GEN}(S)$$

AvE Analysis is Approximate

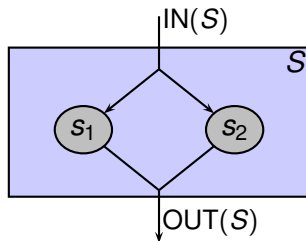


► Thus,

$\text{true GEN}(S) \supseteq \text{analysis GEN}(S)$

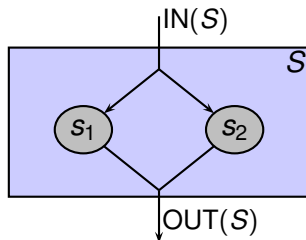
$\text{true KILL}(S) \subseteq \text{analysis KILL}(S)$

AvE Analysis is Approximate



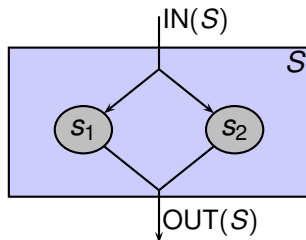
- ▶ Thus,
 $\text{true GEN}(S) \supseteq \text{analysis GEN}(S)$
 $\text{true KILL}(S) \subseteq \text{analysis KILL}(S)$
- ▶ Fewer expressions marked available than actually do!

AvE Analysis is Approximate



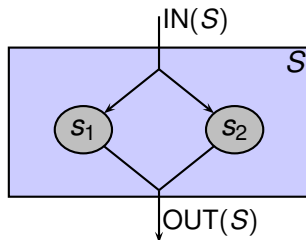
- ▶ Thus,
 - $\text{true GEN}(S) \supseteq \text{analysis GEN}(S)$
 - $\text{true KILL}(S) \subseteq \text{analysis KILL}(S)$
- ▶ Fewer expressions marked available than actually do!
- ▶ Later we shall see that this is **SAFE** approximation

AvE Analysis is Approximate



- ▶ Thus,
 - $\text{true GEN}(S) \supseteq \text{analysis GEN}(S)$
 - $\text{true KILL}(S) \subseteq \text{analysis KILL}(S)$
- ▶ Fewer expressions marked available than actually do!
- ▶ Later we shall see that this is **SAFE** approximation
 - ▶ prevents optimizations

AvE Analysis is Approximate



- ▶ Thus,
 - $\text{true GEN}(S) \supseteq \text{analysis GEN}(S)$
 - $\text{true KILL}(S) \subseteq \text{analysis KILL}(S)$
- ▶ Fewer expressions marked available than actually do!
- ▶ Later we shall see that this is **SAFE** approximation
 - ▶ prevents optimizations
 - ▶ but NO wrong optimization

AvE for Basic Blocks

- ▶ Expr e is available at the start of a block if

$$\text{IN}(B) = \bigcap_{P \in \text{PRED}(B)} \text{OUT}(P)$$

AvE for Basic Blocks

- ▶ Expr e is available at the start of a block if
 - ▶ It is available at the end of all predecessors

$$\text{IN}(B) = \bigcap_{P \in \text{PRED}(B)} \text{OUT}(P)$$

AvE for Basic Blocks

- ▶ Expr e is available at the start of a block if
 - ▶ It is available at the end of all predecessors

$$\text{IN}(B) = \bigcap_{P \in \text{PRED}(B)} \text{OUT}(P)$$

- ▶ Expr e is available at the end of a block if

$$\text{OUT}(B) = \text{IN}(B) - \text{KILL}(B) \cup \text{GEN}(B)$$

AvE for Basic Blocks

- ▶ Expr e is available at the start of a block if
 - ▶ It is available at the end of all predecessors

$$IN(B) = \bigcap_{P \in \text{PRED}(B)} OUT(P)$$

- ▶ Expr e is available at the end of a block if
 - ▶ Either it is generated by the block

$$OUT(B) = IN(B) - KILL(B) \cup GEN(B)$$

AvE for Basic Blocks

- ▶ Expr e is available at the start of a block if
 - ▶ It is available at the end of all predecessors

$$IN(B) = \bigcap_{P \in \text{PRED}(B)} OUT(P)$$

- ▶ Expr e is available at the end of a block if
 - ▶ Either it is generated by the block
 - ▶ Or it is available at the start of the block and not killed by the block

$$OUT(B) = IN(B) - KILL(B) \cup GEN(B)$$

Solving AvE Constraints

- ▶ KILL & GEN known for each BB.

Solving AvE Constraints

- ▶ KILL & GEN known for each BB.
- ▶ A program with N BBs has $2N$ equations with $2N$ unknowns.

Solving AvE Constraints

- ▶ KILL & GEN known for each BB.
- ▶ A program with N BBs has $2N$ equations with $2N$ unknowns.
 - ▶ Solution is possible.

Solving AvE Constraints

- ▶ KILL & GEN known for each BB.
- ▶ A program with N BBs has $2N$ equations with $2N$ unknowns.
 - ▶ Solution is possible.
 - ▶ Iterative approach (on the next slide).

for each block B {

```
for each block  $B$  {  
     $\text{OUT}(B) = \mathcal{U}$ ;  $\mathcal{U}$  = "universal" set of all exprs
```

```
for each block  $B$  {  
     $\text{OUT}(B) = \mathcal{U}$ ;  $\mathcal{U}$  = "universal" set of all exprs  
}  
 $\text{OUT}(\text{Entry}) = \emptyset$ ; // remember reaching defs?
```

```
for each block  $B$  {  
     $\text{OUT}(B) = \mathcal{U}$ ;  $\mathcal{U}$  = "universal" set of all exprs  
}  
 $\text{OUT}(\text{Entry}) = \emptyset$ ; // remember reaching defs?  
change = true;  
while (change) {  
    change = false;
```

```
for each block  $B$  {  
     $\text{OUT}(B) = \mathcal{U}$ ;  $\mathcal{U}$  = "universal" set of all exprs  
}  
 $\text{OUT}(\text{Entry}) = \emptyset$ ; // remember reaching defs?  
change = true;  
while (change) {  
    change = false;  
    for each block  $B$  other than  $\text{Entry}$  {
```

```
for each block  $B$  {  
     $\text{OUT}(B) = \mathcal{U}$ ;  $\mathcal{U}$  = "universal" set of all exprs  
}  
 $\text{OUT}(\text{Entry}) = \emptyset$ ; // remember reaching defs?  
change = true;  
while (change) {  
    change = false;  
    for each block  $B$  other than  $\text{Entry}$  {  
         $\text{IN}(B) = \bigcap_{P \in \text{PRED}(B)} \text{OUT}(P)$ ;
```

```

for each block  $B$  {
     $\text{OUT}(B) = \mathcal{U}$ ;  $\mathcal{U}$  = "universal" set of all exprs
}
 $\text{OUT}(\text{Entry}) = \emptyset$ ; // remember reaching defs?
change = true;
while (change) {
    change = false;
    for each block  $B$  other than  $\text{Entry}$  {
         $\text{IN}(B) = \bigcap_{P \in \text{PRED}(B)} \text{OUT}(P)$ ;
        oldOut =  $\text{OUT}(B)$ ;
         $\text{OUT}(B) = \text{IN}(B) - \text{KILL}(B) \cup \text{GEN}(B)$ ;
    }
}

```



```

for each block  $B$  {
     $\text{OUT}(B) = \mathcal{U}$ ;  $\mathcal{U}$  = "universal" set of all exprs
}
 $\text{OUT}(\text{Entry}) = \emptyset$ ; // remember reaching defs?
change = true;
while (change) {
    change = false;
    for each block  $B$  other than  $\text{Entry}$  {
         $\text{IN}(B) = \bigcap_{P \in \text{PRED}(B)} \text{OUT}(P)$ ;
        oldOut =  $\text{OUT}(B)$ ;
         $\text{OUT}(B) = \text{IN}(B) - \text{KILL}(B) \cup \text{GEN}(B)$ ;
        if ( $\text{OUT}(B) \neq \text{oldOut}$ ) then {
            change = true;
        }
    }
}

```

Some Issues

- ▶ What is \mathcal{U} – the set of *all* expressions?

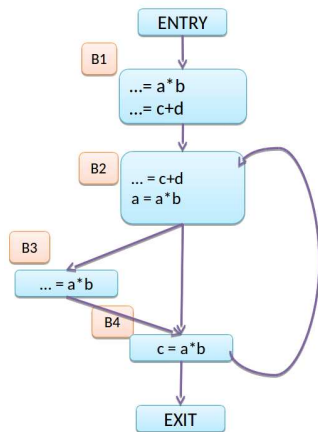
Some Issues

- ▶ What is \mathcal{U} – the set of *all* expressions?
- ▶ How to compute it efficiently?

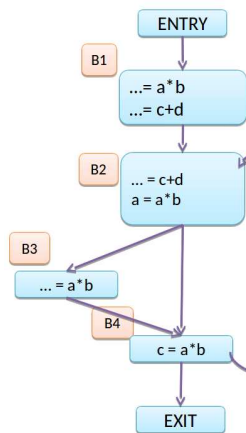
Some Issues

- ▶ What is \mathcal{U} – the set of *all* expressions?
- ▶ How to compute it efficiently?
- ▶ Why *Entry* block is initialized differently?

Available Expressions: Example



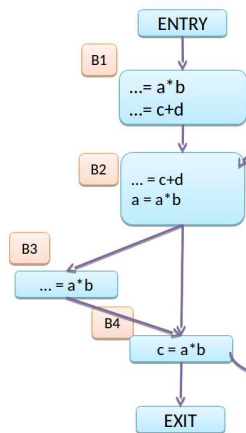
Available Expressions: Example



BB	GEN	KILL
B1		
B2		
B3		
B4		

$\mathcal{U} = \{a*b, c+d\}$

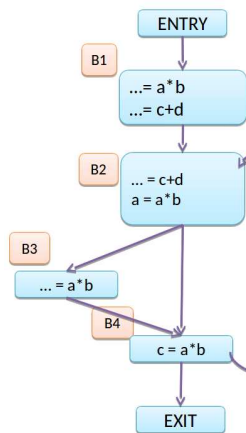
Available Expressions: Example



BB	GEN	KILL
B1	$\{a*b, c+d\}$	
B2		
B3		
B4		

$\mathcal{U} = \{a*b, c+d\}$

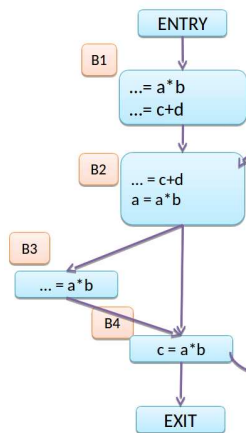
Available Expressions: Example



BB	GEN	KILL
B1	$\{a*b, c+d\}$	$\{\}$
B2		
B3		
B4		

$\mathcal{U} = \{a*b, c+d\}$

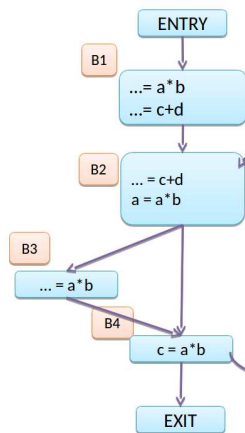
Available Expressions: Example



BB	GEN	KILL
B1	$\{a*b, c+d\}$	$\{\}$
B2	$\{c+d\}$	
B3		
B4		

$\mathcal{U} = \{a*b, c+d\}$

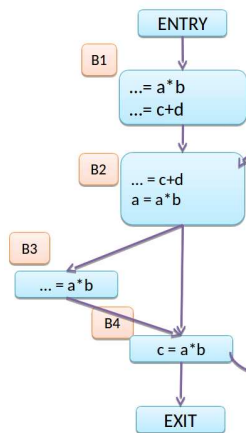
Available Expressions: Example



BB	GEN	KILL
B1	$\{a * b, c + d\}$	$\{\}$
B2	$\{c + d\}$	$\{a * b\}$
B3		
B4		

$\mathcal{U} = \{a * b, c + d\}$

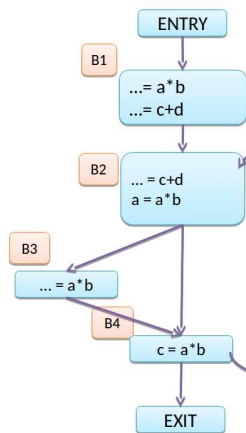
Available Expressions: Example



BB	GEN	KILL
B1	$\{a * b, c + d\}$	$\{\}$
B2	$\{c + d\}$	$\{a * b\}$
B3	$\{a * b\}$	
B4		

$\mathcal{U} = \{a * b, c + d\}$

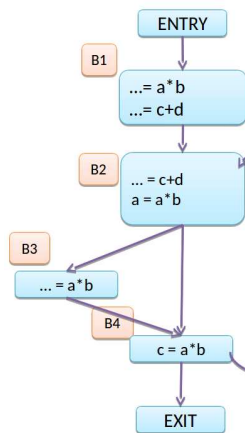
Available Expressions: Example



BB	GEN	KILL
B1	$\{a*b, c+d\}$	$\{\}$
B2	$\{c+d\}$	$\{a*b\}$
B3	$\{a*b\}$	$\{\}$
B4		

$\mathcal{U} = \{a*b, c+d\}$

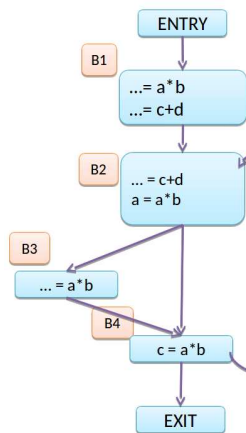
Available Expressions: Example



BB	GEN	KILL
B1	$\{a*b, c+d\}$	$\{\}$
B2	$\{c+d\}$	$\{a*b\}$
B3	$\{a*b\}$	$\{\}$
B4	$\{a*b\}$	

$\mathcal{U} = \{a*b, c+d\}$

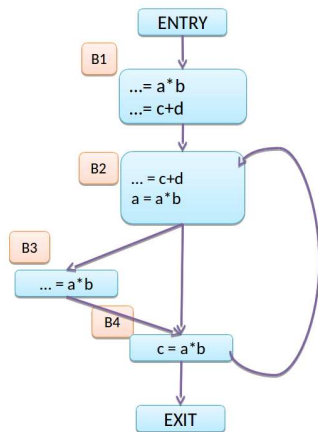
Available Expressions: Example



BB	GEN	KILL
B1	{a*b, c+d}	{ }
B2	{c+d}	{a*b}
B3	{a*b}	{ }
B4	{a*b}	{c+d}

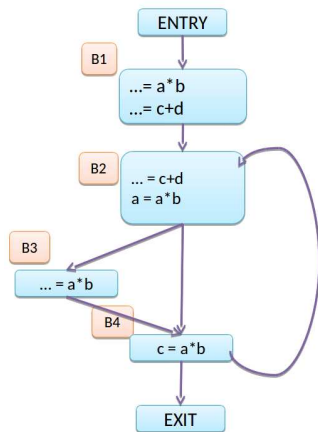
$\mathcal{U} = \{a*b, c+d\}$

Available Expressions: Example



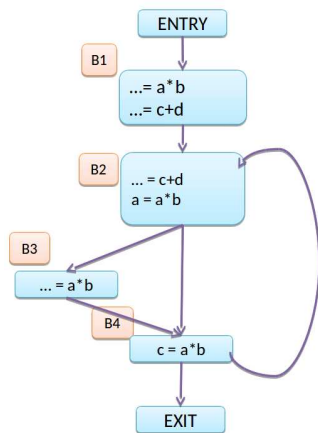
Pass#	Pt	B1	B2	B3	B4
Init	IN	-	-	-	-
	OUT	\mathcal{U}	\mathcal{U}	\mathcal{U}	\mathcal{U}

Available Expressions: Example



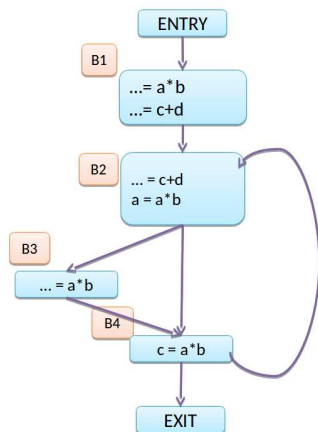
Pass#	Pt	B1	B2	B3	B4
Init	IN	-	-	-	-
	OUT	\mathcal{U}	\mathcal{U}	\mathcal{U}	\mathcal{U}
1	IN	\emptyset	$a*b, c+d$	$c+d$	$c+d$
	OUT	$a*b, c+d$	$c+d$	$a*b, c+d$	$a*b$

Available Expressions: Example



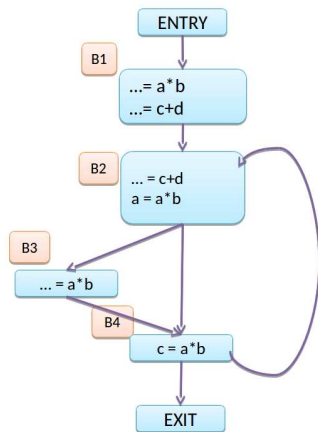
Pass#	Pt	B1	B2	B3	B4
Init	IN	-	-	-	-
	OUT	\mathcal{U}	\mathcal{U}	\mathcal{U}	\mathcal{U}
1	IN	\emptyset	a*b, c+d	c+d	c+d
	OUT	a*b, c+d	c+d	a*b, c+d	a*b
2	IN	\emptyset	a*b	c+d	c+d
	OUT	a*b, c+d	c+d	a*b, c+d	a*b

Available Expressions: Example



Pass#	Pt	B1	B2	B3	B4
Init	IN	-	-	-	-
	OUT	\mathcal{U}	\mathcal{U}	\mathcal{U}	\mathcal{U}
1	IN	\emptyset	a*b, c+d	c+d	c+d
	OUT	a*b, c+d	c+d	a*b, c+d	a*b
2	IN	\emptyset	a*b	c+d	c+d
	OUT	a*b, c+d	c+d	a*b, c+d	a*b
3	IN	\emptyset	a*b	c+d	c+d
	OUT	a*b, c+d	c+d	a*b, c+d	a*b

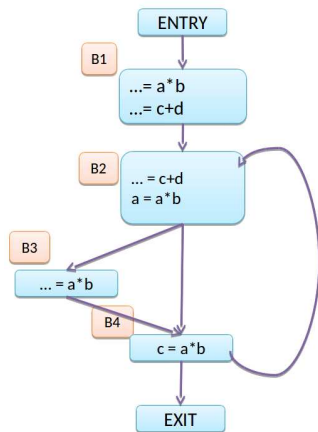
Available Expressions: Bitvectors



a bit for each expression:

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

Available Expressions: Bitvectors



a bit for each expression:

$a*b$ $c+d$

Pass#	Pt	B1	B2	B3	B4
Init	IN	-	-	-	-
	OUT	11	11	11	11
1	IN	00	11	01	01
	OUT	11	01	11	10
2	IN	00	10	01	01
	OUT	11	01	11	10
3	IN	00	10	01	01
	OUT	11	01	11	10

Available Expressions: Bitvectors

- Set-theoretic definitions:

$$\text{IN}(B) = \bigcap_{P \in \text{PRED}(B)} \text{OUT}(P)$$

$$\text{OUT}(B) = \text{IN}(B) - \text{KILL}(B) \cup \text{GEN}(B)$$

Available Expressions: Bitvectors

- ▶ Set-theoretic definitions:

$$\text{IN}(B) = \bigcap_{P \in \text{PRED}(B)} \text{OUT}(P)$$

$$\text{OUT}(B) = \text{IN}(B) - \text{KILL}(B) \cup \text{GEN}(B)$$

- ▶ Bitvector definitions:

$$\text{IN}(B) = \bigwedge_{P \in \text{PRED}(B)} \text{OUT}(P)$$

$$\text{OUT}(B) = \text{IN}(B) \wedge \neg \text{KILL}(B) \vee \text{GEN}(B)$$

Available Expressions: Bitvectors

- ▶ Set-theoretic definitions:

$$\text{IN}(B) = \bigcap_{P \in \text{PRED}(B)} \text{OUT}(P)$$

$$\text{OUT}(B) = \text{IN}(B) - \text{KILL}(B) \cup \text{GEN}(B)$$

- ▶ Bitvector definitions:

$$\text{IN}(B) = \bigwedge_{P \in \text{PRED}(B)} \text{OUT}(P)$$

$$\text{OUT}(B) = \text{IN}(B) \wedge \neg \text{KILL}(B) \vee \text{GEN}(B)$$

- ▶ Bitwise \vee, \wedge, \neg operators

Available Expressions: Application

- ▶ Common subexpression elimination in a block B

Available Expressions: Application

- ▶ Common subexpression elimination in a block B
 - ▶ Expression e available at the entry of B

Available Expressions: Application

- ▶ Common subexpression elimination in a block B
 - ▶ Expression e available at the entry of B
 - ▶ e is also computed at a point p in B

Available Expressions: Application

- ▶ Common subexpression elimination in a block B
 - ▶ Expression e available at the entry of B
 - ▶ e is also computed at a point p in B
 - ▶ Components of e are not modified from entry of B to p

Available Expressions: Application

- ▶ Common subexpression elimination in a block B
 - ▶ Expression e available at the entry of B
 - ▶ e is also computed at a point p in B
 - ▶ Components of e are not modified from entry of B to p
- ▶ e is “upward exposed” in B

Available Expressions: Application

- ▶ Common subexpression elimination in a block B
 - ▶ Expression e available at the entry of B
 - ▶ e is also computed at a point p in B
 - ▶ Components of e are not modified from entry of B to p
- ▶ e is “upward exposed” in B
- ▶ Expressions generated in B are “downward exposed”

Comparison of RD and AvE

- ▶ *Some* vs. *All* path property

Comparison of RD and AvE

- ▶ *Some* vs. *All* path property
- ▶ Meet operator: \cup vs. \cap

Comparison of RD and AvE

- ▶ *Some* vs. *All* path property
- ▶ Meet operator: \cup vs. \cap
- ▶ Initialization of *Entry*: \emptyset

Comparison of RD and AvE

- ▶ *Some* vs. *All* path property
- ▶ Meet operator: \cup vs. \cap
- ▶ Initialization of *Entry*: \emptyset
- ▶ Initialization of other BBs: \emptyset vs. \mathcal{U}

Comparison of RD and AvE

- ▶ *Some* vs. *All* path property
- ▶ Meet operator: \cup vs. \cap
- ▶ Initialization of *Entry*: \emptyset
- ▶ Initialization of other BBs: \emptyset vs. \mathcal{U}
- ▶ Safety: “More” RD vs. “Fewer” AvE

AvE: alternate Initialization

- ▶ What if we Initialize:

$$\text{OUT}(B) = \emptyset, \forall B \text{ including } \textit{Entry}$$

AvE: alternate Initialization

- ▶ What if we Initialize:

$$\text{OUT}(B) = \emptyset, \forall B \text{ including } \textit{Entry}$$

- ▶ Would we find “extra” available expressions?

AvE: alternate Initialization

- ▶ What if we Initialize:

$$\text{OUT}(B) = \emptyset, \forall B \text{ including } \textit{Entry}$$

- ▶ Would we find “extra” available expressions?
 - ▶ More opportunity to optimize?

AvE: alternate Initialization

- ▶ What if we Initialize:

$$\text{OUT}(B) = \emptyset, \forall B \text{ including } \textit{Entry}$$

- ▶ Would we find “extra” available expressions?
 - ▶ More opportunity to optimize?
- ▶ OR would we miss some expressions that are available?

AvE: alternate Initialization

- ▶ What if we Initialize:

$$\text{OUT}(B) = \emptyset, \forall B \text{ including } \textit{Entry}$$

- ▶ Would we find “extra” available expressions?
 - ▶ More opportunity to optimize?
- ▶ OR would we miss some expressions that are available?
 - ▶ Loose on opportunity to optimize?

Live Variables

- ▶ A variable x is live at a point p if

Live Variables

- ▶ A variable x is live at a point p if
 - ▶ There is a point p' along some path in the flow graph starting at p to the *Exit*

Live Variables

- ▶ A variable x is live at a point p if
 - ▶ There is a point p' along some path in the flow graph starting at p to the *Exit*
 - ▶ Value of x could be used at p'

Live Variables

- ▶ A variable x is live at a point p if
 - ▶ There is a point p' along some path in the flow graph starting at p to the *Exit*
 - ▶ Value of x could be used at p'
 - ▶ There is no definition of x between p and p' along this path

Live Variables

- ▶ A variable x is live at a point p if
 - ▶ There is a point p' along some path in the flow graph starting at p to the *Exit*
 - ▶ Value of x could be used at p'
 - ▶ There is no definition of x between p and p' along this path
- ▶ Otherwise x is dead at p

Live Variables: GEN

- ▶ $\text{GEN}(B)$: Set of variables whose values may be used in block B prior to any definition
 - ▶ Also called “ $\text{use}(B)$ ”
- ▶ “upward exposed use” of a variable in B

Live Variables: KILL

- ▶ $KILL(B)$: Set of variables defined in block B prior to any use
 - ▶ Also called “ $def(B)$ ”
- ▶ “upward exposed definition” of a variable in B

Live Variables: Equations

- Set-theoretic definitions:

$$\text{OUT}(B) = \bigcup_{S \in \text{SUCC}(B)} \text{IN}(S)$$

$$\text{IN}(B) = \text{OUT}(B) - \text{KILL}(B) \cup \text{GEN}(B)$$

Live Variables: Equations

- ▶ Set-theoretic definitions:

$$\text{OUT}(B) = \bigcup_{S \in \text{SUCC}(B)} \text{IN}(S)$$

$$\text{IN}(B) = \text{OUT}(B) - \text{KILL}(B) \cup \text{GEN}(B)$$

- ▶ Bitvector definitions:

$$\text{OUT}(B) = \bigvee_{S \in \text{SUCC}(B)} \text{OUT}(S)$$

$$\text{IN}(B) = \text{OUT}(B) \wedge \neg \text{KILL}(B) \vee \text{GEN}(B)$$

Live Variables: Equations

- Set-theoretic definitions:

$$\text{OUT}(B) = \bigcup_{S \in \text{SUCC}(B)} \text{IN}(S)$$

$$\text{IN}(B) = \text{OUT}(B) - \text{KILL}(B) \cup \text{GEN}(B)$$

- Bitvector definitions:

$$\text{OUT}(B) = \bigvee_{S \in \text{SUCC}(B)} \text{OUT}(S)$$

$$\text{IN}(B) = \text{OUT}(B) \wedge \neg \text{KILL}(B) \vee \text{GEN}(B)$$

- Bitwise \vee, \wedge, \neg operators

Very Busy Expressions

- ▶ Expression e is very busy at a point p if

Very Busy Expressions

- ▶ Expression e is very busy at a point p if
 - ▶ Every path from p to *Exit* has at least one evaluation of e

Very Busy Expressions

- ▶ Expression e is very busy at a point p if
 - ▶ Every path from p to *Exit* has at least one evaluation of e
 - ▶ On every path, there is no assignment to any component variable of e before the first evaluation of e following p

Very Busy Expressions

- ▶ Expression e is very busy at a point p if
 - ▶ Every path from p to *Exit* has at least one evaluation of e
 - ▶ On every path, there is no assignment to any component variable of e before the first evaluation of e following p
- ▶ Also called *Anticipable expression*

- ▶ Expression e is very busy at a point p if
 - ▶ **Every** path from p to *Exit* has at least one evaluation of e and there is no assignment to any component variable of e before the first evaluation of e following p on these paths.
- ▶ Set up the data flow equations for Very Busy Expressions (VBE). You have to give equations for GEN, KILL, IN, and OUT.
- ▶ Think of an optimization/transformation that uses VBE analysis. Briefly describe it (2-3 lines only)
- ▶ Will your optimization be *safe* if we replace “*Every*” by “*Some*” in the definition of VBE?