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

These slides constitute the lecture notes for

- MACS L111 Advanced Data Flow Analysis course at Cambridge University, and
- CS 618 Program Analysis course at IIT Bombay.

They have been made available under GNU FDL v1.2 or later (purely for academic or research use) as teaching material accompanying the book:


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

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 $v$. 

\[
\begin{align*}
    v &= a \ast b \\
    a &= v + 2 \\
    \text{End}
\end{align*}
\]
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 \( v \).
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 $p$

$v$ is not live at $p$

$v$ 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 \( v \).

Path based specification:

\[ v = a \times b \]
\[ a = v + 2 \]
\[ v = a + 2 \]

\( v \) 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 $\equiv$ Single statements or Maximal groups of sequentially executed statements

May 2011 Uday Khedker
Defining Data Flow Analysis for Live Variables Analysis

Basic Blocks \equiv \text{Single statements or Maximal groups of sequentially executed statements}

Control Transfer

\text{Basic Blocks} \rightarrow \text{Single statements or Maximal groups of sequentially executed statements}
Defining Data Flow Analysis for Live Variables Analysis

\[
\begin{align*}
\text{Gen}_k, \text{Kill}_k \\
\text{Gen}_i, \text{Kill}_i \\
\text{Gen}_j, \text{Kill}_j
\end{align*}
\]
Defining Data Flow Analysis for Live Variables Analysis

Local Data Flow Properties

$Gen_k, Kill_k$

$Gen_i, Kill_i$

$Gen_j, Kill_j$
Defining Data Flow Analysis for Live Variables Analysis

For basic blocks consisting of single statements, $Gen_n$ is same as $Ref(n)$ and $Kill_n$ is same as $Def(n)$.

$Gen_k, Kill_k$

$Gen_i, Kill_i$

$Gen_j, Kill_j$
Local Data Flow Properties for Live Variables Analysis

\[ Gen_n = \{ \nu | \text{ variable } \nu \text{ is used in basic block } n \text{ and is not preceded by a definition of } \nu \} \]

\[ Kill_n = \{ \nu | \text{ basic block } n \text{ contains a definition of } \nu \} \]
Local Data Flow Properties for Live Variables Analysis

Gen_n = \{ \nu | \text{variable } \nu \text{ is used in basic block } n \text{ and is not preceded by a definition of } \nu \} \\
Kill_n = \{ \nu | \text{basic block } n \text{ contains a definition of } \nu \}
Local Data Flow Properties for Live Variables Analysis

- **r-value occurrence**
  - Value is only read, e.g. \(x, y, z\) in
    \[x.\text{sum} = y.\text{data} + z.\text{data}\]

- **l-value occurrence**
  - Value is modified e.g. \(y\) in
    \[y = x.\text{lptr}\]

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

May 2011  Uday Khedker
Local Data Flow Properties for Live Variables Analysis

Gen\textsubscript{n} = \{ \nu | \text{variable } \nu \text{ is used in basic block } n \text{ and is not preceded by a definition of } \nu \} \\
Kill\textsubscript{n} = \{ \nu | \text{basic block } n \text{ contains a definition of } \nu \}
Local Data Flow Properties for Live Variables Analysis

\[ \text{Gen}_n = \{ v \mid \text{variable } v \text{ is used in basic block } n \text{ and is not preceded by a definition of } v \} \]

\[ \text{Kill}_n = \{ v \mid \text{basic block } n \text{ contains a definition of } v \} \]

r-value occurrence
Value is only read, e.g. \( x,y,z \) in
\[ x.\text{sum} = y.\text{data} + z.\text{data} \]

l-value occurrence
Value is modified e.g. \( y \) in
\[ y = x.lptr \]

within \( n \)

anywhere in \( n \)
Defining Data Flow Analysis for Live Variables Analysis

\[ \text{In}_k = \text{Gen}_k \cup (\text{Out}_k - \text{Kill}_k) \]

\[ \text{Out}_k = \text{In}_i \cup \text{In}_j \]

\[ \text{In}_i \]

\[ \text{In}_j \]
Defining Data Flow Analysis for Live Variables Analysis

Global Data Flow Properties

\[ \text{In}_k = \text{Gen}_k \cup (\text{Out}_k - \text{Kill}_k) \]

\[ \text{Out}_k = \text{In}_i \cup \text{In}_j \]

\[ \text{In}_i \]

\[ \text{In}_j \]
Defining Data Flow Analysis for Live Variables Analysis

Global Data Flow Properties

- $\text{In}_k = \text{Gen}_k \cup (\text{Out}_k - \text{Kill}_k)$
- $\text{Out}_k = \text{In}_i \cup \text{In}_j$

$\text{In}_i$

$\text{In}_j$
Data Flow Equations For Live Variables Analysis

\[ \text{In}_n = (\text{Out}_n - \text{Kill}_n) \cup \text{Gen}_n \]

\[ \text{Out}_n = \begin{cases} \text{BL} & \text{if } n \text{ is End block} \\ \bigcup_{s \in \text{succ}(n)} \text{In}_s & \text{otherwise} \end{cases} \]
Data Flow Equations For Live Variables Analysis

\[ \text{In}_n = (\text{Out}_n - \text{Kill}_n) \cup \text{Gen}_n \]

\[ \text{Out}_n = \begin{cases} 
    \text{BL} & \text{if } n \text{ is End block} \\
    \bigcup_{s \in \text{succ}(n)} \text{In}_s & \text{otherwise}
\end{cases} \]

- \( \text{In}_n \) and \( \text{Out}_n \) are sets of variables
Data Flow Equations For Live Variables Analysis

\[ In_n = (Out_n - Kill_n) \cup Gen_n \]

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

- \( In_n \) and \( Out_n \) are sets of variables
- \( Bl \) is boundary information representing 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

1. \( w = x \)

2. while \((x.data < \text{max})\)

3. \( x = x.rptr \)

4. \( y = x.lptr \)

5. \( z = \text{New class_of_z} \)

6. \( y = y.lptr \)

7. \( z.sum = x.data + y.data \)

\[
\begin{align*}
In_1 &= (Out_1 - Kill_1) \cup Gen_1 \\
Out_1 &= In_2 \\
In_2 &= (Out_2 - Kill_2) \cup Gen_2 \\
Out_2 &= In_3 \cup In_4 \\
In_3 &= (Out_3 - Kill_3) \cup Gen_3 \\
Out_3 &= In_2 \\
In_4 &= (Out_4 - Kill_4) \cup Gen_4 \\
Out_4 &= In_5 \\
In_5 &= (Out_5 - Kill_5) \cup Gen_5 \\
Out_5 &= In_6 \\
In_6 &= (Out_6 - Kill_6) \cup Gen_6 \\
Out_6 &= In_7 \\
In_7 &= (Out_7 - Kill_7) \cup Gen_7 \\
Out_7 &= \emptyset
\end{align*}
\]
Data Flow Equations for Our Example

1. \( w = x \)
2. while \((x.data < \text{max})\)
4. \( y = x.lptr \) \( x = x.rptr \)
5. \( z = \text{New class of z} \)
6. \( y = y.lptr \)
7. \( z.sum = x.data + y.data \)

\[
\begin{align*}
\text{In}_1 &= (\text{Out}_1 - \text{Kill}_1) \cup \text{Gen}_1 \\
\text{Out}_1 &= \text{In}_2 \\
\text{In}_2 &= (\text{Out}_2 - \text{Kill}_2) \cup \text{Gen}_2 \\
\text{Out}_2 &= \text{In}_3 \cup \text{In}_4 \\
\text{In}_3 &= (\text{Out}_3 - \text{Kill}_3) \cup \text{Gen}_3 \\
\text{Out}_3 &= \text{In}_4 \\
\text{In}_4 &= (\text{Out}_4 - \text{Kill}_4) \cup \text{Gen}_4 \\
\text{Out}_4 &= \text{In}_5 \\
\text{In}_5 &= (\text{Out}_5 - \text{Kill}_5) \cup \text{Gen}_5 \\
\text{Out}_5 &= \text{In}_6 \\
\text{In}_6 &= (\text{Out}_6 - \text{Kill}_6) \cup \text{Gen}_6 \\
\text{Out}_6 &= \text{In}_7 \\
\text{In}_7 &= (\text{Out}_7 - \text{Kill}_7) \cup \text{Gen}_7 \\
\text{Out}_7 &= \emptyset
\end{align*}
\]
Performing Live Variables Analysis

Gen = \{x\}, Kill = \{w\}

Gen = \{x\}, Kill = \emptyset
while (x.data < max)

Gen = \{x\}, Kill = \{y\}
y = x.lptr

Gen = \emptyset, Kill = \{z\}
z = \text{New class_of_z}

Gen = \{y\}, Kill = \{y\}
y = y.lptr

Gen = \{x, y, z\}, Kill = \emptyset
z.sum = x.data + y.data
Performing Live Variables Analysis

Gen = \{x\}, \ Kill = \{w\}
\[ w = x \]

Gen = \{x\}, \ Kill = \emptyset
while (x.data < max)

Gen = \{x\}, \ Kill = \{y\}
\[ y = x.lptr \]

Gen = \emptyset, \ Kill = \{z\}
\[ z = \text{New class of } z \]

Gen = \{y\}, \ Kill = \{y\}
\[ y = y.lptr \]

Gen = \{x, y, z\}, \ Kill = \emptyset
\[ z.sum = x.data + y.data \]

Gen and Kill need not be mutually exclusive
Performing Live Variables Analysis

\[
Gen = \{x\}, \quad Kill = \{w\} \\
w = x
\]

\[
Gen = \{x\}, \quad Kill = \emptyset \\
while \ (x.data < \text{max})
\]

\[
Gen = \{x\}, \quad Kill = \{y\} \\
y = x.lptr
\]

\[
Gen = \emptyset, \quad Kill = \{z\} \\
z = \text{New class of } z
\]

\[
Gen = \{y\}, \quad Kill = \{y\} \\
y = y.lptr
\]

\[
Gen = \{x, y, z\}, \quad Kill = \emptyset \\
z.sum = x.data + y.data
\]

*z is an r-value occurrence and not an l-value occurrence*
Performing Live Variables Analysis

\[
\begin{align*}
\text{Gen} &= \{x\}, \quad \text{Kill} = \{w\} \\
\text{w} &= x \\
\text{Gen} &= \{x\}, \quad \text{Kill} = \emptyset \\
\text{while } (x.\text{data} < \text{max}) \\
\text{Gen} &= \{x\}, \quad \text{Kill} = \{y\} \\
\text{y} &= x.\text{lptr} \\
\text{Gen} &= \emptyset, \quad \text{Kill} = \{z\} \\
\text{z} &= \text{New class of } z \\
\text{Gen} &= \{y\}, \quad \text{Kill} = \{y\} \\
\text{y} &= y.\text{lptr} \\
\text{Gen} &= \{x, y, z\}, \quad \text{Kill} = \emptyset \\
\text{z.sum} &= x.\text{data} + y.\text{data}
\end{align*}
\]

\(x, y, z\) are considered to be used based purely on local use even if the value of \(z\) is not use later. A different analysis called faint variables analysis improves on this.
Performing Live Variables Analysis

\[
\begin{align*}
\emptyset & \quad \text{Gen} = \{x\}, \quad \text{Kill} = \{w\} \\
\emptyset & \quad w = x \\
\emptyset & \quad \text{Gen} = \{x\}, \quad \text{Kill} = \emptyset \\
\emptyset & \quad \text{while (x.data < max)} \\
\emptyset & \quad \text{Gen} = \{x\}, \quad \text{Kill} = \emptyset \\
\emptyset & \quad y = x.lptr \\
\emptyset & \quad \text{Gen} = \emptyset, \quad \text{Kill} = \{z\} \\
\emptyset & \quad z = \text{New class of } z \\
\emptyset & \quad \text{Gen} = \{y\}, \quad \text{Kill} = \{y\} \\
\emptyset & \quad y = y.lptr \\
\emptyset & \quad \text{Gen} = \{x, y, z\}, \quad \text{Kill} = \emptyset \\
\emptyset & \quad z.sum = x.data + y.data
\end{align*}
\]
Performing Live Variables Analysis

Ignoring max because we are doing analysis for pointer variables \( w, x, y, z \)

\[
\begin{align*}
\text{Gen} &= \{ x \}, \quad \text{Kill} = \{ w \} \\
\{ x \} &\quad w = x
\end{align*}
\]

\[
\begin{align*}
\text{Gen} &= \{ x \}, \quad \text{Kill} = \emptyset \\
\{ x \} &\quad \text{while} \ (x.\text{data} \leq \text{max})
\end{align*}
\]

\[
\begin{align*}
\text{Gen} &= \{ x \}, \quad \text{Kill} = \{ y \} \\
\{ x \} &\quad y = x.\text{lptr}
\end{align*}
\]

\[
\begin{align*}
\text{Gen} &= \emptyset, \quad \text{Kill} = \{ z \} \\
\{ x, y, z \} &\quad z = \text{New class of z}
\end{align*}
\]

\[
\begin{align*}
\text{Gen} &= \{ y \}, \quad \text{Kill} = \{ y \} \\
\{ x, y, z \} &\quad y = y.\text{lptr}
\end{align*}
\]

\[
\begin{align*}
\text{Gen} &= \{ x, y, z \}, \quad \text{Kill} = \emptyset \\
\{ x, y, z \} &\quad z.\text{sum} = x.\text{data} + y.\text{data}
\end{align*}
\]

Traversal

Iteration #1
Performing Live Variables Analysis

Ignoring max because we are doing analysis for pointer variables \(w, x, y, z\)

\[\text{Gen} = \{x\}, \text{Kill} = \{w\}\]
\[w = x\]
\[\text{Gen} = \{x\}, \text{Kill} = \emptyset\]
\[\text{while} \ (x.\text{data} < \text{max})\]

\[\text{Gen} = \{x\}, \text{Kill} = \{y\}\]
\[y = x.\text{lptr}\]
\[\text{Gen} = \emptyset, \text{Kill} = \{z\}\]
\[z = \text{New class of z}\]

\[\text{Gen} = \{y\}, \text{Kill} = \{y\}\]
\[y = y.\text{lptr}\]

\[\text{Gen} = \{x, y, z\}, \text{Kill} = \emptyset\]
\[z.\text{sum} = x.\text{data} + y.\text{data}\]

Iteration #2

Traversal

May 2011  Uday Khedker
Performing Live Variables Analysis

Gen = \{x\}, Kill = \{w\}

w = x

Gen = \{x\}, Kill = \emptyset

while (x.data < max)

Gen = \{x\}, Kill = \{y, z\}

y = x.lptr
z = New class_of_z
y = y.lptr
z.sum = x.data + y.data

Gen = \{x\}, Kill = \{x\}

x = x.rptr
Local Data Flow Properties for Live Variables Analysis

\[ \text{In}_n = (\text{Out}_n - \text{Kill}_n) \cup \text{Gen}_n \]

- \text{Gen}_n : Use not preceded by definition (Ref for a statement)

- \text{Kill}_n : Definition anywhere in a block (Def for a statement)
Local Data Flow Properties for Live Variables Analysis

\[ \text{In}_n = (\text{Out}_n - \text{Kill}_n) \cup \text{Gen}_n \]

- \( \text{Gen}_n \): Use not preceded by definition (Ref for a statement)
  
  Upwards exposed use

- \( \text{Kill}_n \): 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

<table>
<thead>
<tr>
<th>Case</th>
<th>Local Information</th>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$v \notin Gen_n$</td>
<td>$v \notin Kill_n$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$v \in Gen_n$</td>
<td>$v \notin Kill_n$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$v \notin Gen_n$</td>
<td>$v \in Kill_n$</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$v \in Gen_n$</td>
<td>$v \in Kill_n$</td>
<td></td>
</tr>
</tbody>
</table>

May 2011

Uday Khedker
## Local Data Flow Properties for Live Variables Analysis

<table>
<thead>
<tr>
<th>Case</th>
<th>Local Information</th>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$v \notin \text{Gen}_n$</td>
<td>$v \notin \text{Kill}_n$</td>
<td>$a = b + c$&lt;br&gt;$b = c \times d$</td>
</tr>
<tr>
<td>2</td>
<td>$v \in \text{Gen}_n$</td>
<td>$v \notin \text{Kill}_n$</td>
<td>$a = b + c$&lt;br&gt;$b = v \times d$</td>
</tr>
<tr>
<td>3</td>
<td>$v \notin \text{Gen}_n$</td>
<td>$v \in \text{Kill}_n$</td>
<td>$a = b + c$&lt;br&gt;$v = c \times d$</td>
</tr>
<tr>
<td>4</td>
<td>$v \in \text{Gen}_n$</td>
<td>$v \in \text{Kill}_n$</td>
<td>$a = v + c$&lt;br&gt;$v = c \times d$</td>
</tr>
</tbody>
</table>
• Used for register allocation.
  If variable $x$ is live in a basic block $b$, it is a potential candidate for register allocation.
Using Data Flow Information of Live Variables Analysis

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

• Used for dead code elimination.
  If variable $x$ is not live after an assignment $x = \ldots$, then the assignment is redundant and can be deleted as dead code.
Tutorial Problem 1 for Liveness Analysis

Local Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td>∅</td>
<td>{a, b, c, n}</td>
</tr>
<tr>
<td>n2</td>
<td>{a, n}</td>
<td>∅</td>
</tr>
<tr>
<td>n3</td>
<td>{a}</td>
<td>{a}</td>
</tr>
<tr>
<td>n4</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>n5</td>
<td>{a, b, c}</td>
<td>{a, t1}</td>
</tr>
<tr>
<td>n6</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>
### Local Data Flow Information

<table>
<thead>
<tr>
<th>Block</th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td>∅</td>
<td>{a, b, c, n}</td>
</tr>
<tr>
<td>n2</td>
<td>{a, n}</td>
<td>∅</td>
</tr>
<tr>
<td>n3</td>
<td>{a}</td>
<td>{a}</td>
</tr>
<tr>
<td>n4</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>n5</td>
<td>{a, b, c}</td>
<td>{a, t1}</td>
</tr>
<tr>
<td>n6</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>

### Global Data Flow Information

<table>
<thead>
<tr>
<th>Block</th>
<th>Iteration #1</th>
<th>Iteration #2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>n6</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>n5</td>
<td>∅</td>
<td>{a, b, c}</td>
</tr>
<tr>
<td>n4</td>
<td>{a, b, c}</td>
<td>{a, b, c}</td>
</tr>
<tr>
<td>n3</td>
<td>∅</td>
<td>{a}</td>
</tr>
<tr>
<td>n2</td>
<td>{a, b, c}</td>
<td>{a, b, c, n}</td>
</tr>
<tr>
<td>n1</td>
<td>{a, b, c, n}</td>
<td>∅</td>
</tr>
</tbody>
</table>
Tutorial Problem 1 for Liveness Analysis

Local Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td>∅</td>
<td>{a, b, c, n}</td>
</tr>
<tr>
<td>n2</td>
<td>{a, n}</td>
<td>∅</td>
</tr>
<tr>
<td>n3</td>
<td>{a}</td>
<td>{a}</td>
</tr>
<tr>
<td>n4</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>n5</td>
<td>{a, b, c}</td>
<td>{a, t1}</td>
</tr>
<tr>
<td>n6</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>

Global Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Iteration #1</th>
<th>Iteration #2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>n6</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>n5</td>
<td>∅</td>
<td>{a, b, c}</td>
</tr>
<tr>
<td>n4</td>
<td>{a, b, c}</td>
<td>{a, b, c}</td>
</tr>
<tr>
<td>n3</td>
<td>∅</td>
<td>{a}</td>
</tr>
<tr>
<td>n2</td>
<td>{a, b, c}</td>
<td>{a, b, c, n}</td>
</tr>
<tr>
<td>n1</td>
<td>{a, b, c, n}</td>
<td>∅</td>
</tr>
</tbody>
</table>
Tutorial Problem 2 for Liveness Analysis: C Program

```c
1 int x, y, z;
2 int exmp(void)
3 {
4     int a, b, c, d;
5     b = 4;
6     a = b + c;
7     d = a * b;
8     if (x < y)
9         b = a - c;
10    else
11        do
12            c = b + c;
13            if (y > x)
14                do
15                    d = a + b;
16                    f(b + c);
17                } while(y > x);
18            } while(z > x);
19        } else
20            c = a * b;
21            f(a - b);
22        g (a + b);
23    } while(z > x);
24 }
25 h(a-c);
26 f(b+c);
27 }
```
Tutorial Problem 2 for Liveness Analysis: Control Flow Graph

\[ \text{Var} = \{a, b, c, d\} \]

1. \[ b = 4; \]
2. \[ a = b + c; \]
3. \[ d = a \ast b; \]
4. \[ n_2 \quad b = a - c; \]
5. \[ n_3 \quad c = b + c; \]
6. \[ n_4 \quad c = a \ast b; \quad f(a - b); \]
7. \[ n_5 \quad d = a + b; \]
8. \[ n_6 \quad f(b + c); \]
9. \[ n_7 \quad g(a + b); \]
10. \[ n_8 \quad h(a - c); \quad f(b + c); \]
Tutorial Problem 2 for Liveness Analysis: Control Flow Graph

\( n_5 \) and \( n_6 \) have been artificially separated. gcc combines them.

\[ \text{\texttt{\textbackslash Var}} = \{a, b, c, d\} \]

Graph:

- \( n_1 \): \( b = 4; \ a = b + c; \ d = a \ast b; \)
- \( n_2 \): \( b = a - c; \)
- \( n_3 \): \( c = b + c; \)
- \( n_4 \): \( c = a \ast b; \ f(a - b); \)
- \( n_5 \): \( d = a + b; \)
- \( n_6 \): \( f(b + c); \)
- \( n_7 \): \( g(a + b); \)
- \( n_8 \): \( h(a - c); \ f(b + c); \)

May 2011 Uday Khedker
## Solution of the Tutorial Problem

### Table of Local Information and Global Information

<table>
<thead>
<tr>
<th>Block</th>
<th>Local Information</th>
<th>Global Information</th>
<th>Gen&lt;sub&gt;n&lt;/sub&gt;</th>
<th>Kill&lt;sub&gt;n&lt;/sub&gt;</th>
<th>Out&lt;sub&gt;n&lt;/sub&gt;</th>
<th>In&lt;sub&gt;n&lt;/sub&gt;</th>
<th>Out&lt;sub&gt;n&lt;/sub&gt;</th>
<th>In&lt;sub&gt;n&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>n₈</td>
<td>{a, b, c}</td>
<td></td>
<td>Ŧ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n₇</td>
<td>{a, b}</td>
<td></td>
<td></td>
<td>Ŧ</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n₆</td>
<td>{b, c}</td>
<td></td>
<td></td>
<td></td>
<td>{a, b, c}</td>
<td></td>
<td>{a, b, c}</td>
<td></td>
</tr>
<tr>
<td>n₅</td>
<td>{a, b}</td>
<td>{d}</td>
<td>{a, b, c}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n₄</td>
<td>{a, b}</td>
<td>{c}</td>
<td>{a, b, c}</td>
<td></td>
<td>{a, b}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n₃</td>
<td>{b, c}</td>
<td>{c}</td>
<td>{a, b, c}</td>
<td></td>
<td>{a, b, c}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n₂</td>
<td>{a, c}</td>
<td>{b}</td>
<td>{a, b, c}</td>
<td></td>
<td>{a, c}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n₁</td>
<td>{c}</td>
<td>{a, b, d}</td>
<td>{a, b, c}</td>
<td></td>
<td>{c}</td>
<td></td>
<td>{a, b, c}</td>
<td></td>
</tr>
</tbody>
</table>
Tutorial Problems for Liveness Analysis

• Perform analysis with universal set $\forall \text{Var}$ 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 $e$. 

\[
\text{Start} \quad a \ast b \quad \text{End}
\]

\[
\text{Start} \quad a \ast b \quad a = \quad \text{End}
\]

\[
\text{Start} \quad a \ast b \quad \text{End}
\]
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 \( e \).

- \( a \ast b \) is available at \( p \)
- \( a \ast b \) 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 $e$. 

- $a \times b$ is available at $p$
- $a \times b$ is not available at $p$
- $a \times b$ is not available at $p$
Local Data Flow Properties for Available Expressions Analysis

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

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

<table>
<thead>
<tr>
<th>Entity</th>
<th>Manipulation</th>
<th>Exposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Gen_n$</td>
<td>Expression</td>
<td>Use</td>
</tr>
<tr>
<td>$Kill_n$</td>
<td>Expression</td>
<td>Modification</td>
</tr>
</tbody>
</table>
Data Flow Equations For Available Expressions Analysis

\[ \text{In}_n = \begin{cases} \text{BI} & \text{n is Start block} \\ \bigcap_{p \in \text{pred}(n)} \text{Out}_p & \text{otherwise} \end{cases} \]

\[ \text{Out}_n = \text{Gen}_n \cup (\text{In}_n - \text{Kill}_n) \]
Data Flow Equations For Available Expressions Analysis

\[
Ln_n = \begin{cases} 
  B1 & n \text{ is Start block} \\
  \bigcap_{p \in \text{pred}(n)} Out_p & \text{otherwise}
\end{cases}
\]

\[
Out_n = \text{Gen}_n \cup (Ln_n - \text{Kill}_n)
\]

Alternatively,

\[
Out_n = f_n(Ln_n), \quad \text{where}
\]

\[
f_n(X) = \text{Gen}_n \cup (X - \text{Kill}_n)
\]
Data Flow Equations For Available Expressions Analysis

\[ \text{In}_n = \begin{cases} \text{BI} & \text{n is Start block} \\ \bigcap_{p \in \text{pred}(n)} \text{Out}_p & \text{otherwise} \end{cases} \]

\[ \text{Out}_n = \text{Gen}_n \cup (\text{In}_n - \text{Kill}_n) \]

Alternatively,

\[ \text{Out}_n = f_n(\text{In}_n), \quad \text{where} \]

\[ f_n(X) = \text{Gen}_n \cup (X - \text{Kill}_n) \]

- \text{In}_n and \text{Out}_n are sets of expressions
Data Flow Equations For Available Expressions Analysis

\[ \begin{align*}
\text{In}_n &= \begin{cases} 
  BI & \text{n is Start block} \\
  \bigcap_{p \in \text{pred}(n)} \text{Out}_p & \text{otherwise}
\end{cases} \\
\text{Out}_n &= \text{Gen}_n \cup (\text{In}_n - \text{Kill}_n)
\end{align*} \]

Alternatively,

\[ \text{Out}_n = f_n(\text{In}_n), \text{ where} \]

\[ f_n(X) = \text{Gen}_n \cup (X - \text{Kill}_n) \]

- \( \text{In}_n \) and \( \text{Out}_n \) 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 $e_1 \equiv a \ast b$, $e_2 \equiv b \ast c$, $e_3 \equiv c \ast d$, $e_4 \equiv d \ast e$

<table>
<thead>
<tr>
<th>Node</th>
<th>Computed</th>
<th>Killed</th>
<th>Available</th>
<th>Redund.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${e_1, e_2}$ 1100</td>
<td>$\emptyset$</td>
<td>0000</td>
<td>$\emptyset$ 0000</td>
</tr>
<tr>
<td>2</td>
<td>${e_3}$ 0010</td>
<td>$\emptyset$</td>
<td>0000</td>
<td>${e_1}$ 1000</td>
</tr>
<tr>
<td>3</td>
<td>$\emptyset$ 0000</td>
<td>${e_2, e_3}$ 0110</td>
<td>${e_1, e_3}$ 1010</td>
<td>$\emptyset$ 0000</td>
</tr>
<tr>
<td>4</td>
<td>$\emptyset$ 0000</td>
<td>${e_3, e_4}$ 0011</td>
<td>${e_1, e_3}$ 1010</td>
<td>$\emptyset$ 0000</td>
</tr>
<tr>
<td>5</td>
<td>${e_1, e_4}$ 1001</td>
<td>$\emptyset$</td>
<td>0000</td>
<td>${e_1}$ 1000</td>
</tr>
<tr>
<td>6</td>
<td>${e_4}$ 0001</td>
<td>$\emptyset$</td>
<td>0000</td>
<td>${e_1, e_4}$ 1001</td>
</tr>
</tbody>
</table>
An Example of Available Expressions Analysis

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

<table>
<thead>
<tr>
<th>Node</th>
<th>Computed</th>
<th>Killed</th>
<th>Available</th>
<th>Redund.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${e_1, e_2}$</td>
<td>1100</td>
<td>$\emptyset$</td>
<td>0000</td>
</tr>
<tr>
<td>2</td>
<td>${e_3}$</td>
<td>0010</td>
<td>$\emptyset$</td>
<td>0000</td>
</tr>
<tr>
<td>3</td>
<td>$\emptyset$</td>
<td>0000</td>
<td>${e_2, e_3}$</td>
<td>0110</td>
</tr>
<tr>
<td>4</td>
<td>$\emptyset$</td>
<td>0000</td>
<td>${e_3, e_4}$</td>
<td>0011</td>
</tr>
<tr>
<td>5</td>
<td>${e_1, e_4}$</td>
<td>1001</td>
<td>$\emptyset$</td>
<td>0000</td>
</tr>
<tr>
<td>6</td>
<td>${e_4}$</td>
<td>0001</td>
<td>$\emptyset$</td>
<td>0000</td>
</tr>
</tbody>
</table>
An Example of Available Expressions Analysis

Let \( e_1 \equiv a \times b, e_2 \equiv b \times c, e_3 \equiv c \times d, e_4 \equiv d \times e \)

### Node Computed Killed Available Redund.

<table>
<thead>
<tr>
<th>Node</th>
<th>Computed</th>
<th>Killed</th>
<th>Available</th>
<th>Redund.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{e_1, e_2}</td>
<td>1100</td>
<td>\emptyset</td>
<td>0000</td>
</tr>
<tr>
<td>2</td>
<td>{e_3}</td>
<td>0010</td>
<td>\emptyset</td>
<td>0000</td>
</tr>
<tr>
<td>3</td>
<td>\emptyset</td>
<td>0000</td>
<td>{e_2, e_3}</td>
<td>0110</td>
</tr>
<tr>
<td>4</td>
<td>\emptyset</td>
<td>0000</td>
<td>{e_3, e_4}</td>
<td>0011</td>
</tr>
<tr>
<td>5</td>
<td>{e_1, e_4}</td>
<td>1001</td>
<td>\emptyset</td>
<td>0000</td>
</tr>
<tr>
<td>6</td>
<td>{e_4}</td>
<td>0001</td>
<td>\emptyset</td>
<td>0000</td>
</tr>
</tbody>
</table>
An Example of Available Expressions Analysis

Iteration #2

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

<table>
<thead>
<tr>
<th>Node</th>
<th>Computed</th>
<th>Killed</th>
<th>Available</th>
<th>Redund.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${e_1, e_2}$</td>
<td>1100</td>
<td>$\emptyset$</td>
<td>0000</td>
</tr>
<tr>
<td>2</td>
<td>${e_3}$</td>
<td>0010</td>
<td>$\emptyset$</td>
<td>0000</td>
</tr>
<tr>
<td>3</td>
<td>$\emptyset$</td>
<td>0000</td>
<td>${e_2, e_3}$</td>
<td>0110</td>
</tr>
<tr>
<td>4</td>
<td>$\emptyset$</td>
<td>0000</td>
<td>${e_3, e_4}$</td>
<td>0011</td>
</tr>
<tr>
<td>5</td>
<td>${e_1, e_4}$</td>
<td>1001</td>
<td>$\emptyset$</td>
<td>0000</td>
</tr>
<tr>
<td>6</td>
<td>${e_4}$</td>
<td>0001</td>
<td>$\emptyset$</td>
<td>0000</td>
</tr>
</tbody>
</table>
An Example of Available Expressions Analysis

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

<table>
<thead>
<tr>
<th>Node</th>
<th>Computed</th>
<th>Killed</th>
<th>Available</th>
<th>Redund.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${e_1, e_2}$ 1100</td>
<td>$\emptyset$</td>
<td>0000</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>2</td>
<td>${e_3}$ 0010</td>
<td>$\emptyset$</td>
<td>0000</td>
<td>${e_1}$</td>
</tr>
<tr>
<td>3</td>
<td>$\emptyset$ 0000</td>
<td>${e_2, e_3}$ 0110</td>
<td>${e_1, e_3}$</td>
<td>1010</td>
</tr>
<tr>
<td>4</td>
<td>$\emptyset$ 0000</td>
<td>${e_3, e_4}$ 0011</td>
<td>${e_1, e_3}$</td>
<td>1010</td>
</tr>
<tr>
<td>5</td>
<td>${e_1, e_4}$ 1001</td>
<td>$\emptyset$</td>
<td>0000</td>
<td>${e_1}$</td>
</tr>
<tr>
<td>6</td>
<td>${e_4}$ 0001</td>
<td>$\emptyset$</td>
<td>0000</td>
<td>${e_1, e_4}$</td>
</tr>
</tbody>
</table>
Tutorial Problem for Available Expressions Analysis

\[ b = 4; \]
\[ a = b + c; \]
\[ d = a \times b; \]

\[ c = b + c; \]
\[ d = a + b; \]
\[ f(b + c); \]

\[ b = a - c; \]
\[ c = a \times b; \]
\[ f(a - b); \]

\[ b = a - c; \]
\[ c = a \times b; \]
\[ f(a - b); \]

\[ g(a + b); \]

\[ h(a - c); \]
\[ f(b + c); \]

\[ \text{Expr} = \{ a \times b, a + b, a - b, a - c, b + c \} \]
Bit vector: \[ \begin{align*} a \times b & | a + b & | a - b & | a - c & | b + c \end{align*} \]

### Solution of the Tutorial Problem

<table>
<thead>
<tr>
<th>Node</th>
<th>Local Information</th>
<th>Global Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( Gen_n )</td>
<td>( Kill_n )</td>
<td>( AntGen_n )</td>
</tr>
<tr>
<td>( n_1 )</td>
<td>10001</td>
<td>11111</td>
<td>00000</td>
</tr>
<tr>
<td>( n_2 )</td>
<td>00010</td>
<td>11101</td>
<td>00010</td>
</tr>
<tr>
<td>( n_3 )</td>
<td>00000</td>
<td>00011</td>
<td>00001</td>
</tr>
<tr>
<td>( n_4 )</td>
<td>10100</td>
<td>00011</td>
<td>10100</td>
</tr>
<tr>
<td>( n_5 )</td>
<td>01000</td>
<td>00000</td>
<td>01000</td>
</tr>
<tr>
<td>( n_6 )</td>
<td>00001</td>
<td>00000</td>
<td>00001</td>
</tr>
<tr>
<td>( n_7 )</td>
<td>01000</td>
<td>00000</td>
<td>01000</td>
</tr>
<tr>
<td>( n_8 )</td>
<td>00011</td>
<td>00000</td>
<td>00011</td>
</tr>
</tbody>
</table>
Further Tutorial Problems

1. $a + c$
2. $a \times b$
3. $b = 2$
4. 
5. $a \times c$
6. $a \times b$
Further Tutorial Problems

Bit Vector

| Node | Initialization $\cap$ | Initialization $\cup$
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 $a + c$</td>
<td>$1$</td>
<td>$1$</td>
</tr>
<tr>
<td>2 $a \times b$</td>
<td>$2$</td>
<td>$2$</td>
</tr>
<tr>
<td>3 $b = 2$</td>
<td>$3$</td>
<td>$3$</td>
</tr>
<tr>
<td>4</td>
<td>$4$</td>
<td>$4$</td>
</tr>
<tr>
<td>5 $a \times c$</td>
<td>$5$</td>
<td>$5$</td>
</tr>
<tr>
<td>6 $a \times b$</td>
<td>$6$</td>
<td>$6$</td>
</tr>
</tbody>
</table>
Further Tutorial Problems

Bit Vector Frameworks: Available Expressions Analysis

**Bit Vector**

\[
\begin{align*}
& a + c \\
& a \cdot b \\
& a \cdot c
\end{align*}
\]

**BI Node Initialization**

<table>
<thead>
<tr>
<th>BI</th>
<th>Node</th>
<th>Initialization ( \cup )</th>
<th>Initialization ( \emptyset )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</td>
<td>1</td>
<td>000 100</td>
<td></td>
</tr>
<tr>
<td>( \cup )</td>
<td>2</td>
<td>100 110</td>
<td></td>
</tr>
<tr>
<td>( \emptyset )</td>
<td>3</td>
<td>110 100</td>
<td></td>
</tr>
<tr>
<td>( \cup )</td>
<td>4</td>
<td>110 110</td>
<td></td>
</tr>
<tr>
<td>( \emptyset )</td>
<td>5</td>
<td>100 101</td>
<td></td>
</tr>
<tr>
<td>( \cup )</td>
<td>6</td>
<td>101 111</td>
<td></td>
</tr>
</tbody>
</table>

May 2011

Uday Khedker
Further Tutorial Problems

Bit Vector

| 1 | $a + c$ |
| 2 | $a \times b$ |
| 3 | $b = 2$ |
| 4 | $a \times c$ |
| 5 | $a \times b$ |

<table>
<thead>
<tr>
<th>$BI$</th>
<th>Node</th>
<th>$\text{Initialization } \bigcup$</th>
<th>$\text{Initialization } \emptyset$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$In_n$</td>
<td>$Out_n$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>000</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>100</td>
<td>110</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>110</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>100</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>101</td>
<td>111</td>
</tr>
</tbody>
</table>

May 2011
Further Tutorial Problems

**Bit Vector**

- \( a + c \)
- \( a \times b \)
- \( a \times c \)

**Node Diagram**

1. \( a + c \)
2. \( a \times b \)
3. \( b = 2 \)
4. \( a \times c \)
5. \( a \times b \)

**Initialization Table**

<table>
<thead>
<tr>
<th>BI</th>
<th>Node</th>
<th>Initialization ( \bigcup )</th>
<th>Initialization ( \emptyset )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>( In_n )</td>
<td>( Out_n )</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>000</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>100</td>
<td>110</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>110</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>100</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>101</td>
<td>111</td>
</tr>
</tbody>
</table>

**BB**

1. 111
2. 101
3. 111
4. 111
5. 101
6. 101
### Further Tutorial Problems

#### Bit Vector

<table>
<thead>
<tr>
<th></th>
<th>$a + c$</th>
<th>$a \times b$</th>
<th>$a \times c$</th>
</tr>
</thead>
</table>

**BI**

<table>
<thead>
<tr>
<th>Node</th>
<th>Initialization $\cup$</th>
<th>Initialization $\emptyset$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$In_n$</td>
<td>$Out_n$</td>
</tr>
<tr>
<td>1</td>
<td>000</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>110</td>
</tr>
<tr>
<td>3</td>
<td>110</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>101</td>
<td>111</td>
</tr>
</tbody>
</table>

**∅**

<table>
<thead>
<tr>
<th>Node</th>
<th>Initialization $\cup$</th>
<th>Initialization $\emptyset$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$In_n$</td>
<td>$Out_n$</td>
</tr>
<tr>
<td>1</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>2</td>
<td>101</td>
<td>111</td>
</tr>
<tr>
<td>3</td>
<td>111</td>
<td>101</td>
</tr>
<tr>
<td>4</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>101</td>
<td>111</td>
</tr>
</tbody>
</table>

**∪**

<table>
<thead>
<tr>
<th>Node</th>
<th>Initialization $\cup$</th>
<th>Initialization $\emptyset$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$In_n$</td>
<td>$Out_n$</td>
</tr>
<tr>
<td>1</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>2</td>
<td>101</td>
<td>111</td>
</tr>
<tr>
<td>3</td>
<td>111</td>
<td>101</td>
</tr>
<tr>
<td>4</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>101</td>
<td>111</td>
</tr>
</tbody>
</table>
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)

![Diagram]

<table>
<thead>
<tr>
<th>Traversal</th>
<th>Initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>$U$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$BI$</td>
<td>$BI$</td>
</tr>
<tr>
<td>$U$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$U$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

Forward

Backward
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)
More Tutorial Problems

Number of iterations assuming that the order of $\text{In}_i$ and $\text{Out}_i$ computation is fixed ($\text{In}_i$ is computed first and then $\text{Out}_i$ is computed)

\[
\begin{array}{c}
1 \\
\downarrow \\
2 \quad a \times b \\
\downarrow \\
3 \quad b = 2 \\
\downarrow \\
4 \\
\downarrow \\
5 \quad a \times c \\
\downarrow \\
6 \quad a \times b
\end{array}
\]

<table>
<thead>
<tr>
<th></th>
<th><strong>Initialization</strong></th>
<th><strong>Traversal</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td><strong>Forward</strong></td>
<td>$B\cap$</td>
<td>$B\cap$</td>
</tr>
<tr>
<td></td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td></td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td><strong>Backward</strong></td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td></td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
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 $a \times b \ | \ a + b \ | \ a - b \ | \ a - c \ | \ b + c$

<table>
<thead>
<tr>
<th>Node</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$Gen_n$</td>
<td>$Kill_n$</td>
</tr>
<tr>
<td>$n_1$</td>
<td>10001</td>
<td>11111</td>
</tr>
<tr>
<td>$n_2$</td>
<td>00010</td>
<td>11101</td>
</tr>
<tr>
<td>$n_3$</td>
<td>00000</td>
<td>00011</td>
</tr>
<tr>
<td>$n_4$</td>
<td>10100</td>
<td>00011</td>
</tr>
<tr>
<td>$n_5$</td>
<td>01000</td>
<td>00000</td>
</tr>
<tr>
<td>$n_6$</td>
<td>00001</td>
<td>00000</td>
</tr>
<tr>
<td>$n_7$</td>
<td>01000</td>
<td>00000</td>
</tr>
<tr>
<td>$n_8$</td>
<td>00011</td>
<td>00000</td>
</tr>
</tbody>
</table>
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 (≡ can change the behaviour of the optimized program)

May 2011
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

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

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

<table>
<thead>
<tr>
<th>Entity</th>
<th>Manipulation</th>
<th>Exposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Gen_n )</td>
<td>Expression</td>
<td>Use</td>
</tr>
<tr>
<td>( Kill_n )</td>
<td>Expression</td>
<td>Modification</td>
</tr>
</tbody>
</table>
Data Flow Equations for Anticipable Expressions Analysis

\[ \begin{align*}
    \text{In}_n &= \text{Gen}_n \cup (\text{Out}_n - \text{Kill}_n) \\
    \text{Out}_n &= \begin{cases} \\
        \text{BI} & \text{n is End block} \\
        \bigcap_{s \in \text{succ}(n)} \text{In}_s & \text{otherwise} \\
    \end{cases}
\end{align*} \]

\text{In}_n \text{ and } \text{Out}_n \text{ are sets of expressions}
Tutorial Problem for Anticipable Expressions Analysis

\[ n_1 \]
\[ b = 4; \]
\[ a = b + c; \]
\[ d = a \times b; \]

\[ n_2 \]
\[ b = a - c; \]

\[ n_3 \]
\[ c = b + c; \]

\[ n_4 \]
\[ c = a \times b; \]
\[ f(a - b); \]

\[ n_5 \]
\[ d = a + b; \]

\[ n_6 \]
\[ f(b + c); \]

\[ n_7 \]
\[ g(a + b); \]

\[ n_8 \]
\[ h(a - c); \]
\[ f(b + c); \]

\[ \text{Expr} = \{ a \times b, a + b, a - b, a - c, b + c \} \]
## Result of Anticipable Expressions Analysis

Bit vector

\[
\begin{align*}
& a \times b \\
& a + b \\
& a - b \\
& a - c \\
& b + c
\end{align*}
\]

<table>
<thead>
<tr>
<th>Block</th>
<th>Local Information</th>
<th>Global Information</th>
<th>Iteration # 1</th>
<th>Changes in iteration # 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$Gen_n$</td>
<td>$Kill_n$</td>
<td>$Out_n$</td>
<td>$In_n$</td>
</tr>
<tr>
<td>$n_8$</td>
<td>00011</td>
<td>00000</td>
<td>00000</td>
<td>00011</td>
</tr>
<tr>
<td>$n_7$</td>
<td>01000</td>
<td>00000</td>
<td>00011</td>
<td>01011</td>
</tr>
<tr>
<td>$n_6$</td>
<td>00001</td>
<td>00000</td>
<td>01011</td>
<td>01011</td>
</tr>
<tr>
<td>$n_5$</td>
<td>01000</td>
<td>00000</td>
<td>01011</td>
<td>01011</td>
</tr>
<tr>
<td>$n_4$</td>
<td>10100</td>
<td>00011</td>
<td>01011</td>
<td>11100</td>
</tr>
<tr>
<td>$n_3$</td>
<td>00001</td>
<td>00011</td>
<td>01000</td>
<td>01001</td>
</tr>
<tr>
<td>$n_2$</td>
<td>00010</td>
<td>11101</td>
<td>00011</td>
<td>00010</td>
</tr>
<tr>
<td>$n_1$</td>
<td>00000</td>
<td>11111</td>
<td>00000</td>
<td>00000</td>
</tr>
</tbody>
</table>
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 redefinition 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 $v$

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

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

<table>
<thead>
<tr>
<th></th>
<th>Entity</th>
<th>Manipulation</th>
<th>Exposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Gen_n$</td>
<td>Definition</td>
<td>Occurrence</td>
<td>Downwards</td>
</tr>
<tr>
<td>$Kill_n$</td>
<td>Definition</td>
<td>Occurrence</td>
<td>Anywhere</td>
</tr>
</tbody>
</table>
Data Flow Equations for Reaching Definitions Analysis

\[\begin{align*}
Ind_n &= \begin{cases} 
Bl & \text{$n$ is Start block} \\
\bigcup_{p \in \text{pred}(n)} Out_p & \text{otherwise}
\end{cases} \\
Out_n &= Gen_n \cup (In_n - \text{Kill}_n) \\
Bl &= \{d_x : x = \text{undef} \mid x \in \text{Var}\}
\end{align*}\]

*Ind$_n$ and Out$_n$ are sets of definitions*
Tutorial Problem for Reaching Definitions Analysis

\[ b_1 : b = 4; \]
\[ a_1 : a = b + c; \]
\[ d_1 : d = a \times b; \]

\[ n_1 \]

\[ b_2 : b = a - c; \]
\[ c_1 : c = b + c; \]

\[ n_2 \]

\[ n_3 \]

\[ c_2 : c = a \times b; \]
\[ f(a - b); \]

\[ n_4 \]

\[ n_5 \]

\[ d_2 : d = a + b; \]

\[ n_6 \]

\[ f(b + c); \]

\[ n_7 \]

\[ g(a + b); \]

\[ D_{ef} = \{ a_0, b_0, c_0, d_0, a_1, b_1, b_2, c_1, c_2, d_1, d_2 \} \]

\[ n_8 \]

\[ h(a - c); \]
\[ f(b + c); \]
# Result of Reaching Definitions Analysis

<table>
<thead>
<tr>
<th>Block</th>
<th>Local Information</th>
<th>Global Information</th>
<th>Changes in iteration # 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gen$_n$</td>
<td>Iteration # 1</td>
<td></td>
</tr>
<tr>
<td>$n_1$</td>
<td>${a_1, b_1, d_1}$</td>
<td>${a_0, b_0, c_0, d_0}$</td>
<td>${a_1, b_1, c_0, d_1}$</td>
</tr>
<tr>
<td>$n_2$</td>
<td>${b_2}$</td>
<td>${a_1, b_1, c_0, d_1}$</td>
<td>${a_1, b_2, c_0, d_1}$</td>
</tr>
<tr>
<td>$n_3$</td>
<td>${c_1}$</td>
<td>${a_1, b_1, c_0, d_1}$</td>
<td>${a_1, b_1, c_1, d_1}$</td>
</tr>
<tr>
<td>$n_4$</td>
<td>${c_2}$</td>
<td>${a_1, b_1, c_1, d_1}$</td>
<td>${a_1, b_1, c_2, d_1}$</td>
</tr>
<tr>
<td>$n_5$</td>
<td>${d_2}$</td>
<td>${a_1, b_1, c_1, d_1}$</td>
<td>${a_1, b_1, c_1, d_2}$</td>
</tr>
<tr>
<td>$n_6$</td>
<td>$\emptyset$</td>
<td>${a_1, b_1, c_1, d_2}$</td>
<td>${a_1, b_1, c_1, d_2}$</td>
</tr>
<tr>
<td>$n_7$</td>
<td>$\emptyset$</td>
<td>${a_1, b_1, c_1, c_2, d_1, d_2}$</td>
<td>${a_1, b_1, c_1, c_2, d_1, d_2}$</td>
</tr>
<tr>
<td>$n_8$</td>
<td>$\emptyset$</td>
<td>${a_1, b_1, b_2, c_0, c_1, c_2, d_1, d_2}$</td>
<td>${a_1, b_1, b_2, c_0, c_1, c_2, d_1, d_2}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kill$_n$</th>
<th>${a_0, a_1, b_0, b_1, b_2, d_0, d_1, d_2}$</th>
<th>${a_1, b_1, c_0, d_1}$</th>
<th>${a_1, b_1, c_0, c_1, c_2, d_1, d_2}$</th>
<th>${a_1, b_1, c_1, d_1, d_2}$</th>
<th>${a_1, b_1, c_2, d_1, d_2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_1$</td>
<td>${a_0, a_1, b_0, b_1, b_2, d_0, d_1, d_2}$</td>
<td>${a_1, b_1, c_0, d_1}$</td>
<td>${a_1, b_1, c_0, c_1, c_2, d_1, d_2}$</td>
<td>${a_1, b_1, c_1, d_1, d_2}$</td>
<td>${a_1, b_1, c_2, d_1, d_2}$</td>
</tr>
<tr>
<td>$n_2$</td>
<td>${b_0, b_1, b_2}$</td>
<td>${a_1, b_1, c_0, d_1}$</td>
<td>${a_1, b_2, c_0, d_1}$</td>
<td>${a_1, b_1, c_0, c_1, c_2, d_1, d_2}$</td>
<td>${a_1, b_1, c_1, d_1, d_2}$</td>
</tr>
<tr>
<td>$n_3$</td>
<td>${c_0, c_1, c_2}$</td>
<td>${a_1, b_1, c_0, d_1}$</td>
<td>${a_1, b_1, c_1, d_1}$</td>
<td>${a_1, b_1, c_1, c_2, d_1, d_2}$</td>
<td>${a_1, b_1, c_1, d_1, d_2}$</td>
</tr>
<tr>
<td>$n_4$</td>
<td>${c_0, c_1, c_2}$</td>
<td>${a_1, b_1, c_1, d_1}$</td>
<td>${a_1, b_1, c_2, d_1}$</td>
<td>${a_1, b_1, c_1, c_2, d_1, d_2}$</td>
<td>${a_1, b_1, c_2, d_1, d_2}$</td>
</tr>
<tr>
<td>$n_5$</td>
<td>${d_0, d_1, d_2}$</td>
<td>${a_1, b_1, c_1, d_1}$</td>
<td>${a_1, b_1, c_1, d_2}$</td>
<td>${a_1, b_1, c_1, c_2, d_1, d_2}$</td>
<td>${a_1, b_1, c_1, c_2, d_1, d_2}$</td>
</tr>
<tr>
<td>$n_6$</td>
<td>$\emptyset$</td>
<td>${a_1, b_1, c_1, d_2}$</td>
<td>${a_1, b_1, c_1, d_2}$</td>
<td>${a_1, b_1, c_1, d_2}$</td>
<td>${a_1, b_1, c_1, d_2}$</td>
</tr>
<tr>
<td>$n_7$</td>
<td>$\emptyset$</td>
<td>${a_1, b_1, c_1, c_2, d_1, d_2}$</td>
<td>${a_1, b_1, c_1, c_2, d_1, d_2}$</td>
<td>${a_1, b_1, c_1, c_2, d_1, d_2}$</td>
<td>${a_1, b_1, c_1, c_2, d_1, d_2}$</td>
</tr>
<tr>
<td>$n_8$</td>
<td>$\emptyset$</td>
<td>${a_1, b_1, b_2, c_0, c_1, c_2, d_1, d_2}$</td>
<td>${a_1, b_1, b_2, c_0, c_1, c_2, d_1, d_2}$</td>
<td>${a_1, b_1, b_2, c_0, c_1, c_2, d_1, d_2}$</td>
<td>${a_1, b_1, b_2, c_0, c_1, c_2, d_1, d_2}$</td>
</tr>
</tbody>
</table>
Part 6

Common Features of Bit Vector Data Flow Frameworks
Defining Local Data Flow Properties

- Live variables analysis

<table>
<thead>
<tr>
<th></th>
<th>Entity</th>
<th>Manipulation</th>
<th>Exposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Gen_n$</td>
<td>Variable</td>
<td>Use</td>
<td>Upwards</td>
</tr>
<tr>
<td>$Kill_n$</td>
<td>Variable</td>
<td>Modification</td>
<td>Anywhere</td>
</tr>
</tbody>
</table>

- Analysis of expressions

<table>
<thead>
<tr>
<th></th>
<th>Entity</th>
<th>Manipulation</th>
<th>Exposition</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Exposition</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Availability</td>
<td>Anticipability</td>
</tr>
<tr>
<td>$Gen_n$</td>
<td>Expression</td>
<td>Use</td>
<td>Downwards</td>
<td>Upwards</td>
</tr>
<tr>
<td>$Kill_n$</td>
<td>Expression</td>
<td>Modification</td>
<td>Anywhere</td>
<td>Anywhere</td>
</tr>
</tbody>
</table>
Common Form of Data Flow Equations

\[ X_i = f(Y_i) \]

\[ Y_i = \sqcap X_j \]
Common Form of Data Flow Equations

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

\[ X_i = f(Y_i) \]

\[ Y_i = \bigcap X_j \]
Common Form of Data Flow Equations

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

Data Flow Information

Flow Function

\[ X_i = f(Y_i) \]

\[ Y_i = \bigcap X_j \]

So far we have seen constant Gen and Kill. Could be dependent Gen and Kill.
Common Form of Data Flow Equations

Data Flow Information

$x_i = f(y_i)$

$y_i = \bigcap x_j$

Flow Function

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.

Confluence

So far we have seen $\cup$ and $\cap$. Could be other operations.
# A Taxonomy of Bit Vector Data Flow Frameworks

<table>
<thead>
<tr>
<th></th>
<th>Confluence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Union</td>
</tr>
<tr>
<td>Forward</td>
<td>Reaching Definitions</td>
</tr>
<tr>
<td>Backward</td>
<td>Live Variables</td>
</tr>
<tr>
<td>Bidirectional (limited)</td>
<td>Partial Redundancy Elimination (Original M-R Formulation)</td>
</tr>
</tbody>
</table>
### A Taxonomy of Bit Vector Data Flow Frameworks

<table>
<thead>
<tr>
<th></th>
<th>Confluence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Union</td>
</tr>
<tr>
<td>Forward</td>
<td>Reaching Definitions</td>
</tr>
<tr>
<td>Backward</td>
<td>Live Variables</td>
</tr>
<tr>
<td>Bidirectional (limited)</td>
<td>Partial Redundancy Elimination (Original M-R Formulation)</td>
</tr>
</tbody>
</table>
# A Taxonomy of Bit Vector Data Flow Frameworks

<table>
<thead>
<tr>
<th></th>
<th>Union</th>
<th>Intersection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward</td>
<td>Reaching Definitions</td>
<td>Available Expressions</td>
</tr>
<tr>
<td>Backward</td>
<td>Live Variables</td>
<td>Anticipable Expressions</td>
</tr>
<tr>
<td>Bidirectional (limited)</td>
<td></td>
<td>Partial Redundancy Elimination (Original M-R Formulation)</td>
</tr>
</tbody>
</table>

**Confluence**
- **Any Path**
- **All Paths**
# A Taxonomy of Bit Vector Data Flow Frameworks

<table>
<thead>
<tr>
<th></th>
<th>Confluence</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Union</td>
<td>Intersection</td>
</tr>
<tr>
<td>Forward</td>
<td>Reaching Definitions</td>
<td>Available Expressions</td>
</tr>
<tr>
<td>Backward</td>
<td>Live Variables</td>
<td>Anticipable Expressions</td>
</tr>
<tr>
<td>Bidirectional</td>
<td></td>
<td>Partial Redundancy Elimination (Original M-R Formulation)</td>
</tr>
</tbody>
</table>

- **Any Path**
- **All Paths**
A Taxonomy of Bit Vector Data Flow Frameworks

<table>
<thead>
<tr>
<th></th>
<th>Confluence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Union</td>
</tr>
<tr>
<td>Forward</td>
<td>Reaching Definitions</td>
</tr>
<tr>
<td>Backward</td>
<td>Live Variables</td>
</tr>
<tr>
<td>Bidirectional (limited)</td>
<td>Partial Redundancy Elimination (Original M-R Formulation)</td>
</tr>
</tbody>
</table>
## A Taxonomy of Bit Vector Data Flow Frameworks

<table>
<thead>
<tr>
<th></th>
<th>Confluence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Union</strong></td>
</tr>
<tr>
<td>Forward</td>
<td>Reaching Definitions</td>
</tr>
<tr>
<td>Backward</td>
<td>Live Variables</td>
</tr>
<tr>
<td>Bidirectional</td>
<td>Partial Redundancy Elimination (Original M-R Formulation)</td>
</tr>
<tr>
<td>(limited)</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Intersection</strong></td>
</tr>
<tr>
<td>Forward</td>
<td>Available Expressions</td>
</tr>
<tr>
<td>Backward</td>
<td>Anticipable Expressions</td>
</tr>
</tbody>
</table>

- **Any Path**
- **All Paths**
Data Flow Paths Discovered by Data Flow Analysis

Liveness

Anticipability

Availability

Partial Availability

\[ a \times b \]
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

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 \(a \ast b\), 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 \(a \ast b\).
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 \(a \times b\), 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 \(a \times b\).
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 \(a \ast b\), and
- no other block on the path contains an assignment to \(a\) or \(b\).
Data Flow Paths Discovered by Data Flow Analysis

Liveness

Anticipability

Availability

Partial Availability