Live Variables Analysis

Uday Khedker
(www.cse.iitb.ac.in/~uday)

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

Dec 2019
Part 1

About These Slides
Copyright

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

  (Indian edition published by Ane Books in 2013)

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


These slides are being made available under GNU FDL v1.2 or later purely for academic or research use.
Outline

- Live Variables Analysis
- Some Observations
- Strongly Live Variables Analysis
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 $v$. 

\[ \text{Start} \quad \begin{array}{c} \text{p} \\ v = a * b \end{array} \quad \text{End} \]

\[ \text{Start} \quad \begin{array}{c} \text{p} \\ v = a + 2 \end{array} \quad \text{End} \]

\[ \text{Start} \quad \begin{array}{c} \text{p} \\ v = v + 2 \end{array} \quad \text{End} \]
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 $p$

\[ v = a \times b \]

\[ a = v + 2 \]

End

\[ v = a + 2 \]

End

\[ v = v + 2 \]

End
Defining Live Variables Analysis

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

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

$$v = a \cdot b$$

$$a = v + 2$$

$$v = a + 2$$
Defining Live Variables Analysis

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

Path based specification

$v$ is live at $p$

\[ \begin{align*}
    &\text{Start} \\
    &v = a \ast b \\
    &a = v + 2 \\
    &\text{End}
\end{align*} \]

$v$ is not live at $p$

\[ \begin{align*}
    &\text{Start} \\
    &v = a \ast b \\
    &v = a + 2 \\
    &\text{End}
\end{align*} \]

$v$ is live at $p$

\[ \begin{align*}
    &\text{Start} \\
    &v = a \ast b \\
    &v = v + 2 \\
    &\text{End}
\end{align*} \]
Defining Data Flow Analysis for Live Variables Analysis

In $i$, $\text{Gen}_i$, $\text{Kill}_i$, $\text{Out}_i$

In $j$, $\text{Gen}_j$, $\text{Kill}_j$, $\text{Out}_j$

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

$\text{Gen}_k$, $\text{Kill}_k$, $\text{Out}_k$ = In $i$ $\cup$ In $j$
Defining Data Flow Analysis for Live Variables Analysis

Basic Blocks ≡ Single statements or Maximal groups of sequentially executed statements
Defining Data Flow Analysis for Live Variables Analysis

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

\[ \text{Gen}_k, \text{Kill}_k \]

\[ \text{Gen}_i, \text{Kill}_i \]

\[ \text{Gen}_j, \text{Kill}_j \]
Defining Data Flow Analysis for Live Variables Analysis

Local Data Flow Properties
Local Data Flow Properties for Live Variables Analysis

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

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

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

\[ \text{Gen}_n = \{ \, v \mid \text{variable } v \text{ is used in basic block } n \text{ and is not preceded by a definition of } v \, \} \]
\[ \text{Kill}_n = \{ \, v \mid \text{basic block } n \text{ contains a definition of } v \, \} \]
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
  \[ y = x.\text{lptr} \]

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

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

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

- **r-value occurrence**
  Value is only read, e.g. `x,y,z` in
  
  ```
  x.sum = y.data + z.data
  ```

- **l-value occurrence**
  Value is modified e.g. `y` in
  
  ```
  y = x.lptr
  ```

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

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

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

\[ \text{Out}_k = \text{In}_i \cup \text{In}_j \]
Defining Data Flow Analysis for Live Variables Analysis

Global Data Flow Properties

\[ In_k = Gen_k \cup (Out_k - Kill_k) \]

\[ Out_k = In_i \cup In_j \]

\[ \hat{In}_i \]

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

\[ \text{Global Data Flow Properties} \]

\[ \text{Edge based specifications} \]

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

\[ \text{Out}_k = \ln_i \cup \ln_j \]

\[ \ln_i \]

\[ \ln_j \]
Data Flow Equations For Live Variables Analysis

\[
\begin{align*}
    In_n &= (Out_n - Kill_n) \cup Gen_n \\
    Out_n &= \begin{cases} 
        BI & \text{n is End block} \\
        \bigcup_{s \in succ(n)} In_s & \text{otherwise}
    \end{cases}
\end{align*}
\]
Data Flow Equations For Live Variables Analysis

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

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

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

\[ Out_n = \begin{cases} 
\emptyset & \text{for local variables except for the values being returned} \\
\text{set of global variables used further in any calling context} & \text{(can be safely approximated by the set of all global variables)} \\
\bigcup_{s \in \text{succ}(n)} In_s & \text{otherwise} 
\end{cases} \]

- \( In_n \) and \( Out_n \) are sets of variables.
- \( BI \) is boundary information representing the effect of calling contexts.
Data Flow Equations for Our Example

\[ w = x \]

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

3. \(x = x.rptr\)

4. \(y = x.lptr\)

5. \(z = \text{New class} \_ \text{of}\_z\)

6. \(y = y.lptr\)

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

8. return \(z.\text{sum}\)

\[ \begin{align*}
In_1 &= (Out_1 - \text{Kill}_1) \cup Gen_1 \\
Out_1 &= In_2 \\
In_2 &= (Out_2 - \text{Kill}_2) \cup Gen_2 \\
Out_2 &= In_3 \cup In_4 \\
In_3 &= (Out_3 - \text{Kill}_3) \cup Gen_3 \\
Out_3 &= In_2 \\
In_4 &= (Out_4 - \text{Kill}_4) \cup Gen_4 \\
Out_4 &= In_5 \\
In_5 &= (Out_5 - \text{Kill}_5) \cup Gen_5 \\
Out_5 &= In_6 \\
In_6 &= (Out_6 - \text{Kill}_6) \cup Gen_6 \\
Out_6 &= In_7 \\
In_7 &= (Out_7 - \text{Kill}_7) \cup Gen_7 \\
Out_7 &= In_8 \\
In_8 &= (Out_8 - \text{Kill}_8) \cup Gen_8 \\
Out_8 &= \emptyset
\end{align*} \]
**Data Flow Equations for Our Example**

1. \( w = x \)

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

3. \( x = x\text{.rptr} \)

4. \( y = x\text{.lptr} \)

5. \( z = \text{New class\_of\_z} \)

6. \( y = y\text{.lptr} \)

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

8. \( \text{return } z\text{.sum} \)

\[
\begin{align*}
\text{In}_1 &= (\text{Out}_1 - \text{Kill}_1) \cup \text{Gen}_1 \\
\text{Out}_1 &= \text{In}_2 \\
\text{In}_2 &= (\text{Out}_2 - \text{Kill}_2) \cup \text{Gen}_2 \\
\text{Out}_2 &= \text{In}_3 \cup \text{In}_4 \\
\text{In}_3 &= (\text{Out}_3 - \text{Kill}_3) \cup \text{Gen}_3 \\
\text{Out}_3 &= \text{In}_2 \\
\text{In}_4 &= (\text{Out}_4 - \text{Kill}_4) \cup \text{Gen}_4 \\
\text{Out}_4 &= \text{In}_5 \\
\text{In}_5 &= (\text{Out}_5 - \text{Kill}_5) \cup \text{Gen}_5 \\
\text{Out}_5 &= \text{In}_6 \\
\text{In}_6 &= (\text{Out}_6 - \text{Kill}_6) \cup \text{Gen}_6 \\
\text{Out}_6 &= \text{In}_7 \\
\text{In}_7 &= (\text{Out}_7 - \text{Kill}_7) \cup \text{Gen}_7 \\
\text{Out}_7 &= \text{In}_8 \\
\text{In}_8 &= (\text{Out}_8 - \text{Kill}_8) \cup \text{Gen}_8 \\
\text{Out}_8 &= \emptyset
\end{align*}
\]
Data Flow Equations for Our Example

$In_1 = (Out_1 - Kill_1) \cup Gen_1$

$Out_1 = In_2$

$In_2 = (Out_2 - Kill_2) \cup Gen_2$

$Out_2 = In_3 \cup In_4$

$In_3 = (Out_3 - Kill_3) \cup Gen_3$

$Out_3 = In_2$

$In_4 = (Out_4 - Kill_4) \cup Gen_4$

$Out_4 = In_5$

$In_5 = (Out_5 - Kill_5) \cup Gen_5$

$Out_5 = In_6$

$In_6 = (Out_6 - Kill_6) \cup Gen_6$

$Out_6 = In_7$

$In_7 = (Out_7 - Kill_7) \cup Gen_7$

$Out_7 = In_8$

$In_8 = (Out_8 - Kill_8) \cup Gen_8$

$Out_8 = \emptyset$
Data Flow Equations for Our Example

$w = x$

$\text{while (} x.\text{data} < \text{MAX})$

$y = x.lptr$

$x = x.rptr$

$z = \text{New class of } z$

$y = y.lptr$

$z.\text{sum} = x.\text{data} + y.\text{data}$

return $z.\text{sum}$

$In_1 = (Out_1 - \text{Kill}_1) \cup \text{Gen}_1$

$Out_1 = In_2$

$In_2 = (Out_2 - \text{Kill}_2) \cup \text{Gen}_2$

$Out_2 = In_3 \cup In_4$

$In_3 = (Out_3 - \text{Kill}_3) \cup \text{Gen}_3$

$Out_3 = In_2$

$In_4 = (Out_4 - \text{Kill}_4) \cup \text{Gen}_4$

$Out_4 = In_5$

$In_5 = (Out_5 - \text{Kill}_5) \cup \text{Gen}_5$

$Out_5 = In_6$

$In_6 = (Out_6 - \text{Kill}_6) \cup \text{Gen}_6$

$Out_6 = In_7$

$In_7 = (Out_7 - \text{Kill}_7) \cup \text{Gen}_7$

$Out_7 = In_8$

$In_8 = (Out_8 - \text{Kill}_8) \cup \text{Gen}_8$

$Out_8 = \emptyset$
Data Flow Equations for Our Example

1. \( w = x \)

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

3. \( x = x . \text{rptr} \)

4. \( y = x . \text{lptr} \)

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

6. \( y = y . \text{lptr} \)

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

8. return \( z . \text{sum} \)

\[
\begin{align*}
\text{In}_1 &= (\text{Out}_1 - \text{Kill}_1) \cup \text{Gen}_1 \\
\text{Out}_1 &= \text{In}_2 \\
\text{In}_2 &= (\text{Out}_2 - \text{Kill}_2) \cup \text{Gen}_2 \\
\text{Out}_2 &= \text{In}_3 \cup \text{In}_4 \\
\text{In}_3 &= (\text{Out}_3 - \text{Kill}_3) \cup \text{Gen}_3 \\
\text{Out}_3 &= \text{In}_2 \\
\text{In}_4 &= (\text{Out}_4 - \text{Kill}_4) \cup \text{Gen}_4 \\
\text{Out}_4 &= \text{In}_5 \\
\text{In}_5 &= (\text{Out}_5 - \text{Kill}_5) \cup \text{Gen}_5 \\
\text{Out}_5 &= \text{In}_6 \\
\text{In}_6 &= (\text{Out}_6 - \text{Kill}_6) \cup \text{Gen}_6 \\
\text{Out}_6 &= \text{In}_7 \\
\text{In}_7 &= (\text{Out}_7 - \text{Kill}_7) \cup \text{Gen}_7 \\
\text{Out}_7 &= \text{In}_8 \\
\text{In}_8 &= (\text{Out}_8 - \text{Kill}_8) \cup \text{Gen}_8 \\
\text{Out}_8 &= \emptyset
\end{align*}
\]
Data Flow Equations for Our Example

\[
\text{\textbf{w} = \textbf{x}}
\]

\[
\text{\textbf{while (x.data < MAX)}
\]

\[
\text{\textbf{y} = x.lptr}
\]

\[
\text{\textbf{x} = x.rptr}
\]

\[
\text{\textbf{z} = New\ class\_of\_z}
\]

\[
\text{\textbf{y} = y.lptr}
\]

\[
\text{\textbf{z.sum} = x.data + y.data}
\]

\[
\text{\textbf{return} \textbf{z.sum}}
\]

\[
\text{In}_1 = (\text{Out}_1 - \text{Kill}_1) \cup \text{Gen}_1
\]

\[
\text{Out}_1 = \text{In}_2
\]

\[
\text{In}_2 = (\text{Out}_2 - \text{Kill}_2) \cup \text{Gen}_2
\]

\[
\text{Out}_2 = \text{In}_3 \cup \text{In}_4
\]

\[
\text{In}_3 = (\text{Out}_3 - \text{Kill}_3) \cup \text{Gen}_3
\]

\[
\text{Out}_3 = \text{In}_2
\]

\[
\text{In}_4 = (\text{Out}_4 - \text{Kill}_4) \cup \text{Gen}_4
\]

\[
\text{Out}_4 = \text{In}_5
\]

\[
\text{In}_5 = (\text{Out}_5 - \text{Kill}_5) \cup \text{Gen}_5
\]

\[
\text{Out}_5 = \text{In}_6
\]

\[
\text{In}_6 = (\text{Out}_6 - \text{Kill}_6) \cup \text{Gen}_6
\]

\[
\text{Out}_6 = \text{In}_7
\]

\[
\text{In}_7 = (\text{Out}_7 - \text{Kill}_7) \cup \text{Gen}_7
\]

\[
\text{Out}_7 = \text{In}_8
\]

\[
\text{In}_8 = (\text{Out}_8 - \text{Kill}_8) \cup \text{Gen}_8
\]

\[
\text{Out}_8 = \emptyset
\]
### 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. \( z.sum = x.data + y.data \)

8. return \( z.sum \)

\[
\begin{align*}
\text{In}_1 &= (\text{Out}_1 - \text{Kill}_1) \cup \text{Gen}_1 \\
\text{Out}_1 &= \text{In}_2 \\
\text{In}_2 &= (\text{Out}_2 - \text{Kill}_2) \cup \text{Gen}_2 \\
\text{Out}_2 &= \text{In}_3 \cup \text{In}_4 \\
\text{In}_3 &= (\text{Out}_3 - \text{Kill}_3) \cup \text{Gen}_3 \\
\text{Out}_3 &= \text{In}_2 \\
\text{In}_4 &= (\text{Out}_4 - \text{Kill}_4) \cup \text{Gen}_4 \\
\text{Out}_4 &= \text{In}_5 \\
\text{In}_5 &= (\text{Out}_5 - \text{Kill}_5) \cup \text{Gen}_5 \\
\text{Out}_5 &= \text{In}_6 \\
\text{In}_6 &= (\text{Out}_6 - \text{Kill}_6) \cup \text{Gen}_6 \\
\text{Out}_6 &= \text{In}_7 \\
\text{In}_7 &= (\text{Out}_7 - \text{Kill}_7) \cup \text{Gen}_7 \\
\text{Out}_7 &= \text{In}_8 \\
\text{In}_8 &= (\text{Out}_8 - \text{Kill}_8) \cup \text{Gen}_8 \\
\text{Out}_8 &= \emptyset
\end{align*}
\]
Performing Live Variables Analysis

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

w = x

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

\text{while} \ (x.\text{data} < \text{MAX})

Gen = \{x\}, \quad \text{Kill} = \{y\}

y = x.\text{rptr}

Gen = \emptyset, \quad \text{Kill} = \{z\}

z = \text{New \ class\_of\_z}

Gen = \{y\}, \quad \text{Kill} = \{y\}

y = y.\text{rptr}

Gen = \{x, y, z\}, \quad \text{Kill} = \emptyset

z.\text{sum} = x.\text{data} + y.\text{data}

Gen = \{z\}, \quad \text{Kill} = \emptyset

\text{return} \ z.\text{sum}

Dec 2019
Performing Live Variables Analysis

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

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

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

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

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

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

\[
\text{Gen} = \{z\}, \quad \text{Kill} = \emptyset
\]
\[
\text{return } z.\text{sum}
\]

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

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

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

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

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

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

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

\[ \text{Gen} = \{z\}, \text{Kill} = \emptyset \]
return \(z.\text{sum}\)

\(z\) is an r-value occurrence and not an l-value occurrence.
Performing Live Variables Analysis

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

Ignoring MAX because we are doing analysis for pointer variables w, x, y, z

```
Gen = {x}, Kill = {w}
w = x
Gen = {x}, Kill = {x}
while (x.data < MAX)
Gen = {x}, Kill = {x}
x = x.rptr
Gen = {x}
y = x.lptr
Gen = {}  
z = New class_of_z
Gen = {y}, Kill = {y}
y = y.lptr
Gen = {x, y, z}, Kill = {} 
z.sum = x.data + y.data
Gen = {z}, Kill = {} 
return z.sum
```

Traversal
Performing Live Variables Analysis

Ignoring MAX because we are doing analysis for pointer variables w, x, y, z

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

while (x.data < MAX)

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

\[
\text{Gen} = \{x\}, \quad \text{Kill} = \emptyset
\]

y = x.rptr

Gen = \{x, y\}

z = New class_of_z

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

z.sum = x.data + y.data

\[
\text{Gen} = \{x, y, z\}, \quad \text{Kill} = \emptyset
\]

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

return z.sum

Traversal

Iteration #1

Dec 2019 IIT Bombay
Performing Live Variables Analysis

Ignoring MAX because we are doing analysis for pointer variables w, x, y, z

Iteration #2

Dec 2019
Performing Live Variables Analysis

Local data flow properties when basic blocks contain multiple statements

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

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

\[ \text{Gen} = \{ x \}, \quad \text{Kill} = \{ y, z \} \]
\[ y = x.\text{lptr} \]
\[ z = \text{New class of} \ x \]
\[ y = y.\text{lptr} \]
\[ z.\text{sum} = x.\text{data} + y.\text{data} \]
return \(z.\text{sum}\)

\[ \text{Gen} = \{ x \}, \quad \text{Kill} = \{ x \} \]
\[ x = x.\text{rptr} \]
Local Data Flow Properties for Live Variables Analysis

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

- \text{Gen}_n : Use not preceded by definition

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

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

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

- \( \text{Kill}_n \) : Definition anywhere in a block
  - 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 basic block</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>
# Local Data Flow Properties for Live Variables Analysis

<table>
<thead>
<tr>
<th>Case</th>
<th>Local Information</th>
<th>Example basic block</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( v \notin \text{Gen}_n ) ( v \notin \text{Kill}_n )</td>
<td>( a = b + c ) ( b = c \times d )</td>
<td>liveness of ( v ) is unaffected by the basic block</td>
</tr>
<tr>
<td>2</td>
<td>( v \in \text{Gen}_n ) ( v \notin \text{Kill}_n )</td>
<td>( a = b + c ) ( b = v \times d )</td>
<td>( v ) becomes live before the basic block</td>
</tr>
<tr>
<td>3</td>
<td>( v \notin \text{Gen}_n ) ( v \in \text{Kill}_n )</td>
<td>( a = b + c ) ( v = c \times d ) OR ( v = a + b ) ( c = v \times d )</td>
<td>( v ) ceases to be live before the basic block</td>
</tr>
<tr>
<td>4</td>
<td>( v \in \text{Gen}_n ) ( v \in \text{Kill}_n )</td>
<td>( a = v + c ) ( v = c \times d )</td>
<td>liveness of ( v ) is killed but ( v ) becomes live before the basic block</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: Perform Dead Code Elimination**

```
a = 4
b = 2
c = 3
m = c*2
if (a ≤ m)
  a = a+1
if (a < 12)
  t1 = a+b
  a = t1+c
  print "Hi"
print "Hello"
```

**Local Data Flow Information**

<table>
<thead>
<tr>
<th></th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>∅</td>
<td>{a, b, c, m}</td>
</tr>
<tr>
<td>2</td>
<td>{a, m}</td>
<td>∅</td>
</tr>
<tr>
<td>3</td>
<td>{a}</td>
<td>{a}</td>
</tr>
<tr>
<td>4</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>{a, b, c}</td>
<td>{a, t1}</td>
</tr>
<tr>
<td>6</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>
Tutorial Problem 1: Perform Dead Code Elimination

Local Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>∅</td>
<td>{a, b, c, m}</td>
</tr>
<tr>
<td>2</td>
<td>{a, m}</td>
<td>∅</td>
</tr>
<tr>
<td>3</td>
<td>{a}</td>
<td>{a}</td>
</tr>
<tr>
<td>4</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>{a, b, c}</td>
<td>{a, t1}</td>
</tr>
<tr>
<td>6</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>

Global Data Flow Information

<table>
<thead>
<tr>
<th>Iteration #1</th>
<th>Iteration #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>6</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>∅</td>
</tr>
<tr>
<td>4</td>
<td>{a, b, c}</td>
</tr>
<tr>
<td>3</td>
<td>∅</td>
</tr>
<tr>
<td>2</td>
<td>{a, b, c}</td>
</tr>
<tr>
<td>1</td>
<td>{a, b, c, m}</td>
</tr>
</tbody>
</table>
Tutorial Problem 1: Perform Dead Code Elimination

Local Data Flow Information

<table>
<thead>
<tr>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>∅</td>
</tr>
<tr>
<td>2</td>
<td>{a, m}</td>
</tr>
<tr>
<td>3</td>
<td>{a}</td>
</tr>
<tr>
<td>4</td>
<td>{a}</td>
</tr>
<tr>
<td>5</td>
<td>{a, b, c}</td>
</tr>
<tr>
<td>6</td>
<td>∅</td>
</tr>
</tbody>
</table>

Global Data Flow Information

<table>
<thead>
<tr>
<th>Iteration #1</th>
<th>Iteration #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>6</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>∅</td>
</tr>
<tr>
<td>4</td>
<td>{a, b, c}</td>
</tr>
<tr>
<td>3</td>
<td>∅</td>
</tr>
<tr>
<td>2</td>
<td>{a, b, c}</td>
</tr>
<tr>
<td>1</td>
<td>{a, b, c, m}</td>
</tr>
</tbody>
</table>
Tutorial Problem 1: Perform Dead Code Elimination

Local Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>∅</td>
<td>{a, b, c, m}</td>
</tr>
<tr>
<td>2</td>
<td>{a, m}</td>
<td>∅</td>
</tr>
<tr>
<td>3</td>
<td>{a}</td>
<td>{a}</td>
</tr>
<tr>
<td>4</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>{a, b, c}</td>
<td>{a, t1}</td>
</tr>
<tr>
<td>6</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>

Global Data Flow Information

<table>
<thead>
<tr>
<th>Iteration #1</th>
<th>Iteration #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>6</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>∅</td>
</tr>
<tr>
<td>4</td>
<td>{a, b, c}</td>
</tr>
<tr>
<td>3</td>
<td>∅</td>
</tr>
<tr>
<td>2</td>
<td>{a, b, c}</td>
</tr>
<tr>
<td>1</td>
<td>{a, b, c, m}</td>
</tr>
</tbody>
</table>
Tutorial Problem 1: Observations About Round #1 of Dead Code Elimination

• We can repeat liveness analysis on the optimized code and then optimize it further.
  This can continue as long code continues to change.

• A better approach would be to perform strong liveness analysis.
  The code needs to be optimized only once.

• Here we show the repeated application only to show the scope of further optimizations.
Tutorial Problem 1: Round #2 of Dead Code Elimination

Local Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>∅</td>
<td>{a, b, c, m}</td>
</tr>
<tr>
<td>2</td>
<td>{a, m}</td>
<td>∅</td>
</tr>
<tr>
<td>3</td>
<td>{a}</td>
<td>{a}</td>
</tr>
<tr>
<td>4</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>{a, b}</td>
<td>{t1}</td>
</tr>
<tr>
<td>6</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>

FP School, PCI Pune Liveness Analysis: Live Variables Analysis

a = 4
b = 2
c = 3
m = c*2

if (a ≤ m)
    a = a+1

if (a < 12)
    t1 = a+b
    print "Hi"

print "Hello"
### Tutorial Problem 1: Round #2 of Dead Code Elimination

#### Local Data Flow Information

<table>
<thead>
<tr>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\emptyset$ ${a, b, c, m}$</td>
</tr>
<tr>
<td>2</td>
<td>${a, m}$ $\emptyset$</td>
</tr>
<tr>
<td>3</td>
<td>${a}$ ${a}$</td>
</tr>
<tr>
<td>4</td>
<td>${a}$ $\emptyset$</td>
</tr>
<tr>
<td>5</td>
<td>${a, b}$ ${t_1}$</td>
</tr>
<tr>
<td>6</td>
<td>$\emptyset$ $\emptyset$</td>
</tr>
</tbody>
</table>

#### Global Data Flow Information

<table>
<thead>
<tr>
<th>Iteration #1</th>
<th>Iteration #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>6</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>5</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>4</td>
<td>${a, b}$</td>
</tr>
<tr>
<td>3</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>2</td>
<td>${a, b}$</td>
</tr>
<tr>
<td>1</td>
<td>${a, b, m}$</td>
</tr>
</tbody>
</table>
Tutorial Problem 1: Round #2 of Dead Code Elimination

Local Data Flow Information

<table>
<thead>
<tr>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>∅</td>
</tr>
<tr>
<td>2</td>
<td>∅</td>
</tr>
<tr>
<td>3</td>
<td>∅</td>
</tr>
<tr>
<td>4</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>∅</td>
</tr>
<tr>
<td>6</td>
<td>∅</td>
</tr>
</tbody>
</table>

Global Data Flow Information

<table>
<thead>
<tr>
<th>Iteration #1</th>
<th>Iteration #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>6 ∅</td>
<td>∅</td>
</tr>
<tr>
<td>5 ∅</td>
<td>{a, b}</td>
</tr>
<tr>
<td>4 {a, b}</td>
<td>{a, b}</td>
</tr>
<tr>
<td>3 ∅</td>
<td>{a}</td>
</tr>
<tr>
<td>2 {a, b}</td>
<td>{a, b, m}</td>
</tr>
<tr>
<td>1 {a, b, m}</td>
<td>∅</td>
</tr>
</tbody>
</table>
Tutorial Problem 1: Round #2 of Dead Code Elimination

Local Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>∅</td>
<td>{a, b, c, m}</td>
</tr>
<tr>
<td>2</td>
<td>{a, m}</td>
<td>∅</td>
</tr>
<tr>
<td>3</td>
<td>{a}</td>
<td>{a}</td>
</tr>
<tr>
<td>4</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>{a, b}</td>
<td>{t1}</td>
</tr>
<tr>
<td>6</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>Iteration</td>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>#1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>∅</td>
<td>{a, b}</td>
</tr>
<tr>
<td>4</td>
<td>{a, b}</td>
<td>{a, b}</td>
</tr>
<tr>
<td>3</td>
<td>∅</td>
<td>{a}</td>
</tr>
<tr>
<td>2</td>
<td>{a, b}</td>
<td>{a, b, m}</td>
</tr>
<tr>
<td>1</td>
<td>{a, b, m}</td>
<td>∅</td>
</tr>
</tbody>
</table>
Tutorial Problem 1: Round #3 of Dead Code Elimination

Local Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>∅</td>
<td>{a, b, c, m}</td>
</tr>
<tr>
<td>2</td>
<td>{a, m}</td>
<td>∅</td>
</tr>
<tr>
<td>3</td>
<td>{a}</td>
<td>{a}</td>
</tr>
<tr>
<td>4</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>6</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>
**Tutorial Problem 1: Round #3 of Dead Code Elimination**

### Local Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\emptyset$</td>
<td>${a, b, c, m}$</td>
</tr>
<tr>
<td>2</td>
<td>${a, m}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>3</td>
<td>${a}$</td>
<td>${a}$</td>
</tr>
<tr>
<td>4</td>
<td>${a}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>5</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>6</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

### Global Data Flow Information

<table>
<thead>
<tr>
<th>Iteration #1</th>
<th>Iteration #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>6</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>5</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>4</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>3</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>2</td>
<td>${a}$</td>
</tr>
<tr>
<td>1</td>
<td>${a, m}$</td>
</tr>
</tbody>
</table>
Tutorial Problem 1: Round #3 of Dead Code Elimination

Local Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>∅</td>
<td>{a, b, c, m}</td>
</tr>
<tr>
<td>2</td>
<td>{a, m}</td>
<td>∅</td>
</tr>
<tr>
<td>3</td>
<td>{a}</td>
<td>{a}</td>
</tr>
<tr>
<td>4</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>6</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>6</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>4</td>
<td>∅</td>
<td>{a}</td>
</tr>
<tr>
<td>3</td>
<td>∅</td>
<td>{a}</td>
</tr>
<tr>
<td>2</td>
<td>{a}</td>
<td>{a, m}</td>
</tr>
<tr>
<td>1</td>
<td>{a, m}</td>
<td>∅</td>
</tr>
</tbody>
</table>
Tutorial Problem 1: Round #3 of Dead Code Elimination

```
a = 4
b = 2
c = 3
m = c*2

if (a <= m)
    a = a+1

if (a < 12)
    t1 = a+b
    a = t1+c

print "Hi"
print "Hello"
```

Local Data Flow Information

<table>
<thead>
<tr>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>∅</td>
</tr>
<tr>
<td>2</td>
<td>{a, m}</td>
</tr>
<tr>
<td>3</td>
<td>{a}</td>
</tr>
<tr>
<td>4</td>
<td>{a}</td>
</tr>
<tr>
<td>5</td>
<td>∅</td>
</tr>
<tr>
<td>6</td>
<td>∅</td>
</tr>
</tbody>
</table>

Global Data Flow Information

<table>
<thead>
<tr>
<th>Iteration #1</th>
<th>Iteration #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>6</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>∅</td>
</tr>
<tr>
<td>4</td>
<td>∅</td>
</tr>
<tr>
<td>3</td>
<td>∅</td>
</tr>
<tr>
<td>2</td>
<td>{a}</td>
</tr>
<tr>
<td>1</td>
<td>{a, m}</td>
</tr>
</tbody>
</table>
Part 3

Some Observations
What Does Data Flow Analysis Involve?

- Defining the analysis.
- Formulating the analysis.
- Performing the analysis.
What Does Data Flow Analysis Involve?

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

- **Performing the analysis.**
What Does Data Flow Analysis Involve?

- **Defining the analysis.** Define the properties of execution paths
- **Formulating the analysis.** Define data flow equations
  - Linear simultaneous equations on sets rather than numbers
  - Later we will generalize the domain of values
- **Performing the analysis.**
What Does Data Flow Analysis Involve?

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

- **Defining the analysis.** Define the properties of execution paths

- **Formulating the analysis.** Define data flow equations
  - Linear simultaneous equations on sets rather than numbers
  - Later we will generalize the domain of values

- **Performing the analysis.** Solve data flow equations for the given program flow graph

- **Many unanswered questions**
Iterative Solution of Linear Simultaneous Equations

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

\[ Ax = b \]

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

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

over the control flow graph
Tutorial Problem 2 for Liveness Analysis

Draw the control flow graph and perform live variables analysis

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

    for (i=m-1; i<k; i++)
    {
        if (i>=n)
        {
            a = n;
            a = a+i;
        }
        return a;
    }
}
```
Tutorial Problem 2 for Liveness Analysis

Draw the control flow graph and perform live variables analysis

```
int f(int m, int n, int k)
{
    int a,i;
    for (i=m-1; i<k; i++)
    {
        if (i>=n)
        {
            a = n;
            a = a+i;
        }
    }
    return a;
}
```
## Solution of Tutorial Problem 2

<table>
<thead>
<tr>
<th>Block</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Iteration # 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gen</td>
</tr>
<tr>
<td>6</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>{a, i}</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>{n}</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>{i, n}</td>
<td>∅</td>
</tr>
<tr>
<td>2</td>
<td>{i, k}</td>
<td>∅</td>
</tr>
<tr>
<td>1</td>
<td>{m}</td>
<td></td>
</tr>
</tbody>
</table>
## Solution of Tutorial Problem 2

<table>
<thead>
<tr>
<th>Block</th>
<th>Local Information</th>
<th>Global Information</th>
<th>Iteration # 1</th>
<th>Change in iteration # 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Gen</td>
<td>Kill</td>
</tr>
<tr>
<td>6</td>
<td>{a}</td>
<td></td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>{a, i}</td>
<td>{a, i}</td>
<td>∅</td>
<td>{a, i}</td>
</tr>
<tr>
<td>4</td>
<td>{n}</td>
<td>{a}</td>
<td>{a, i}</td>
<td>{i, n}</td>
</tr>
<tr>
<td>3</td>
<td>{i, n}</td>
<td>∅</td>
<td>{a, i, n}</td>
<td>{a, i, n}</td>
</tr>
<tr>
<td>2</td>
<td>{i, k}</td>
<td>∅</td>
<td>{a, i, n}</td>
<td>{a, i, k, n}</td>
</tr>
<tr>
<td>1</td>
<td>{m}</td>
<td>{i}</td>
<td>{a, i, k, n}</td>
<td>{a, k, m, n}</td>
</tr>
</tbody>
</table>

Dec 2019 IIT Bombay
# Solution of Tutorial Problem 2

<table>
<thead>
<tr>
<th>Block</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gen</td>
<td>Kill</td>
</tr>
<tr>
<td></td>
<td>Iteration # 1</td>
<td>Gen</td>
</tr>
<tr>
<td>6</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>{a, i}</td>
<td>{a, i}</td>
</tr>
<tr>
<td>4</td>
<td>{n}</td>
<td>{a}</td>
</tr>
<tr>
<td>3</td>
<td>{i, n}</td>
<td>∅</td>
</tr>
<tr>
<td>2</td>
<td>{i, k}</td>
<td>∅</td>
</tr>
<tr>
<td>1</td>
<td>{m}</td>
<td>{i}</td>
</tr>
</tbody>
</table>
**Interpreting the Result of Liveness Analysis for Tutorial Problem 2**

1. \( i = m - 1 \)
2. \( \text{if}(i < k) \)
3. \( \text{if}(i \geq n) \)
4. \( a = n \)
5. \( a = a + i \)  
   \( i = i + 1 \)
6. \( \text{return } a \)

- Is a live at the exit of node 5 at the end of iteration 1? Why?
  (We have used post order traversal)
Interpreting the Result of Liveness Analysis for Tutorial Problem 2

- Is a live at the exit of node 5 at the end of iteration 1? Why?
  (We have used post order traversal)
- Is a live at the exit of node 5 at the end of iteration 2? Why?
  (We have used post order traversal)
Interpreting the Result of Liveness Analysis for Tutorial Problem 2

• Is a live at the exit of node 5 at the end of iteration 1? Why?
  (We have used post order traversal)
• Is a live at the exit of node 5 at the end of iteration 2? Why?
  (We have used post order traversal)
• Show an execution path along which a is live at the exit of node 5
Interpreting the Result of Liveness Analysis for Tutorial Problem 2

• Is a live at the exit of node 5 at the end of iteration 1? Why?
  (We have used post order traversal)
• Is a live at the exit of node 5 at the end of iteration 2? Why?
  (We have used post order traversal)
• Show an execution path along which a is live at the exit of node 5
• Show an execution path along which a is live at the exit of node 3
Interpreting the Result of Liveness Analysis for Tutorial Problem 2

- Is a live at the exit of node 5 at the end of iteration 1? Why?
  (We have used post order traversal)

- Is a live at the exit of node 5 at the end of iteration 2? Why?
  (We have used post order traversal)

- Show an execution path along which a is live at the exit of node 5

- Show an execution path along which a is live at the exit of node 3
  $1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 2 \rightarrow \ldots$
Interpreting the Result of Liveness Analysis for Tutorial Problem 2

• Is a live at the exit of node 5 at the end of iteration 1? Why?
  (We have used post order traversal)

• Is a live at the exit of node 5 at the end of iteration 2? Why?
  (We have used post order traversal)

• Show an execution path along which a is live at the exit of node 5

• Show an execution path along which a is live at the exit of node 3
  \[1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 2 \rightarrow \ldots\]

• Show an execution path along which a is not live at the exit of node 3
Interpreting the Result of Liveness Analysis for Tutorial Problem 2

- Is a live at the exit of node 5 at the end of iteration 1? Why? (We have used post order traversal)
- Is a live at the exit of node 5 at the end of iteration 2? Why? (We have used post order traversal)
- Show an execution path along which a is live at the exit of node 5
- Show an execution path along which a is live at the exit of node 3
  \[1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 2 \rightarrow \ldots\]
- Show an execution path along which a is not live at the exit of node 3
  \[1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow \ldots\]
Tutorial Problem 3 for Liveness Analysis

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

```
x = 1
y = 2
if (c)
  x = y + 1
  y = 2 * z
  if (d)
```

```
  n3
  F
  n4
  x = y + z
  T
  n5
  if (c < 20)
  z = 1
  T
  n6
  z = x
  F
```
Tutorial Problem 3 for Liveness Analysis

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

```c
void f()
{
    int x, y, z;
    int c, d;
    x = 1;
    y = 2;
    if (c)
    {
        do
        {
            x = y + 1;
            y = 2 * z;
            if (d)
            {
                x = y + z;
                z = 1;
            }
        } while (c < 20);
    }
    z = x;
}
```
## Solution of Tutorial Problem 3

<table>
<thead>
<tr>
<th>Block</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Iteration # 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gen</td>
</tr>
<tr>
<td>$n_6$</td>
<td>${x}$</td>
<td></td>
</tr>
<tr>
<td>$n_5$</td>
<td>${c}$</td>
<td></td>
</tr>
<tr>
<td>$n_4$</td>
<td>${y, z}$</td>
<td></td>
</tr>
<tr>
<td>$n_3$</td>
<td>${y, z, d}$</td>
<td></td>
</tr>
<tr>
<td>$n_2$</td>
<td>${c}$</td>
<td></td>
</tr>
<tr>
<td>$n_1$</td>
<td>$\emptyset$</td>
<td></td>
</tr>
</tbody>
</table>

Dec 2019 IIT Bombay
### Solution of Tutorial Problem 3

<table>
<thead>
<tr>
<th>Block</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Iteration # 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gen</td>
</tr>
<tr>
<td>$n_6$</td>
<td>${x}$</td>
<td>${z}$</td>
</tr>
<tr>
<td>$n_5$</td>
<td>${c}$</td>
<td>${z}$</td>
</tr>
<tr>
<td>$n_4$</td>
<td>${y, z}$</td>
<td>${x}$</td>
</tr>
<tr>
<td>$n_3$</td>
<td>${y, z, d}$</td>
<td>${x, y}$</td>
</tr>
<tr>
<td>$n_2$</td>
<td>${c}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$n_1$</td>
<td>$\emptyset$</td>
<td>${x, y}$</td>
</tr>
</tbody>
</table>
Solution of Tutorial Problem 3

<table>
<thead>
<tr>
<th>Block</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gen</td>
<td>Kill</td>
</tr>
<tr>
<td>$n_6$</td>
<td>${x}$</td>
<td>${z}$</td>
</tr>
<tr>
<td>$n_5$</td>
<td>${c}$</td>
<td>${z}$</td>
</tr>
<tr>
<td>$n_4$</td>
<td>${y, z}$</td>
<td>${x}$</td>
</tr>
<tr>
<td>$n_3$</td>
<td>${y, z, d}$</td>
<td>${x, y}$</td>
</tr>
<tr>
<td>$n_2$</td>
<td>${c}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$n_1$</td>
<td>$\emptyset$</td>
<td>${x, y}$</td>
</tr>
</tbody>
</table>
Interpreting the Result of Liveness Analysis for Tutorial Problem 3

- Why is $z$ live at the exit of $n_5$?
Interpreting the Result of Liveness Analysis for Tutorial Problem 3

• Why is $z$ live at the exit of $n_5$?
• Why is $z$ not live at the entry of $n_5$?
Interpreting the Result of Liveness Analysis for Tutorial Problem 3

- Why is \( z \) live at the exit of \( n_5 \)?
- Why is \( z \) not live at the entry of \( n_5 \)?
- Why is \( x \) live at the exit of \( n_3 \) inspite of being killed in \( n_4 \)?
Interpreting the Result of Liveness Analysis for Tutorial Problem 3

- Why is \( z \) live at the exit of \( n_5 \)?
- Why is \( z \) not live at the entry of \( n_5 \)?
- Why is \( x \) live at the exit of \( n_3 \) inspite of being killed in \( n_4 \)?
- Identify the instance of dead code elimination
Interpreting the Result of Liveness Analysis for Tutorial Problem 3

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

```
\[
x = 1 \\
y = 2
\]

\[
\text{if (c)}
\]

\[
x = y + 1 \\
y = 2 \times z \\
\text{if (d)}
\]

\[
x = y + z
\]

\[
z = 1 \\
\text{if (c < 20)}
\]

\[
z = x
\]

Dec 2019
Interpreting the Result of Liveness Analysis for Tutorial Problem 3

- Why is z live at the exit of n_5?
- Why is z not live at the entry of n_5?
- Why is x live at the exit of n_3 inspite of being killed in n_4?
- Identify the instance of dead code elimination $z = x$ in n_6
- Would the first round of dead code elimination cause liveness information to change?
Interpreting the Result of Liveness Analysis for Tutorial Problem 3

- Why is z live at the exit of $n_5$?
- Why is z not live at the entry of $n_5$?
- Why is x live at the exit of $n_3$ inspite of being killed in $n_4$?
- Identify the instance of dead code elimination $z = x$ in $n_6$
- Would the first round of dead code elimination cause liveness information to change? Yes
Interpreting the Result of Liveness Analysis for Tutorial Problem 3

- Why is \( z \) live at the exit of \( n_5 \)?
- Why is \( z \) not live at the entry of \( n_5 \)?
- Why is \( x \) live at the exit of \( n_3 \) inspite of being killed in \( n_4 \)?
- Identify the instance of dead code elimination \( z = x \) in \( n_6 \)
- Would the first round of dead code elimination cause liveness information to change? Yes
- Would the second round of liveness analysis lead to further dead code elimination?
Interpreting the Result of Liveness Analysis for Tutorial Problem 3

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

What should be the initial value of internal nodes?
Choice of Initialization

What should be the initial value of internal nodes?

- Confluence is $\bigcup$
- Identity of $\bigcup$ is $\emptyset$
Choice of Initialization

What should be the initial value of internal nodes?

- Confluence is $\cup$
- Identity of $\cup$ is $\emptyset$
- We begin with $\emptyset$ and let the sets at each program point grow
  A revisit to a program point
  - may consider a new execution path
  - more variables may be found to be live
  - a variable found to be live earlier does not become dead
How Does the Initialization Affect the Solution?

\[ a = b = 5 \]

print \( b \)
How Does the Initialization Affect the Solution?

\[ a = b = 5 \]

Init.

\[ \emptyset \]

\[ \emptyset \]

\[ \emptyset \]

\[ \emptyset \]

\[ \emptyset \]
How Does the Initialization Affect the Solution?

\[ a = b = 5 \]

Init. \quad \text{Iter.}

\#1

\[
\begin{array}{l}
\emptyset \\
\emptyset \\
\emptyset \\
\emptyset \\
\emptyset \\
\emptyset \\
\emptyset
\end{array}
\]
How Does the Initialization Affect the Solution?

\[ a = b = 5 \]

\[
\begin{array}{c}
\text{Init.} \\
\emptyset \\
\emptyset \\
\emptyset \\
\emptyset \\
\emptyset \\
\emptyset \\
\emptyset \\
\end{array}
\]

\[
\begin{array}{c}
\text{Iter.} \\
\#1 \\
\emptyset \\
\emptyset \\
\emptyset \\
\emptyset \\
\emptyset \\
\emptyset \\
\end{array}
\]
How Does the Initialization Affect the Solution?

\[ a = b = 5 \]

```
print b
```

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td></td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>

Dec 2019 IIT Bombay
How Does the Initialization Affect the Solution?

\[ a = b = 5 \]

```
print b
```

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter. #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \emptyset )</td>
<td>( {b} )</td>
</tr>
<tr>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
How Does the Initialization Affect the Solution?

\[ a = b = 5 \]

Init. | Iter. #1
--- | ---
\emptyset | \emptyset
\emptyset | \{b\}
\emptyset | \{b\}
\emptyset | \emptyset
\emptyset | \emptyset
\emptyset | \emptyset
How Does the Initialization Affect the Solution?

\[ a = b = 5 \]

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter. #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{∅}</td>
<td>{∅}</td>
</tr>
<tr>
<td>{∅}</td>
<td>{{b}}</td>
</tr>
<tr>
<td>{∅}</td>
<td>{{b}}</td>
</tr>
<tr>
<td>{∅}</td>
<td>{∅}</td>
</tr>
<tr>
<td>{∅}</td>
<td>{∅}</td>
</tr>
</tbody>
</table>

Dec 2019 IIT Bombay
How Does the Initialization Affect the Solution?

\[ a = b = 5 \]

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter. #1</th>
<th>Iter. #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>∅</td>
<td>{b}</td>
<td>{b}</td>
</tr>
<tr>
<td>∅</td>
<td>{b}</td>
<td>{b}</td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
<td>{b}</td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>
How Does the Initialization Affect the Solution?

$\begin{align*}
\text{Init.} & \quad \text{Iter.} \ #1 & \quad \text{Iter.} \ #2 \\
\emptyset & \quad \emptyset & \quad \emptyset \\
\emptyset & \quad \{b\} & \quad \{b\} \\
\emptyset & \quad \emptyset & \quad \{b\} \\
\emptyset & \quad \emptyset & \quad \emptyset
\end{align*}$

$\begin{align*}
\text{Init.} & \quad \text{Iter.} \\
\{a, b\} & \quad \{a, b\} \\
\{a, b\} & \quad \{a, b\} \\
\{a, b\} & \quad \emptyset
\end{align*}$
How Does the Initialization Affect the Solution?

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter. #1</th>
<th>Iter. #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a = b = 5$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td></td>
<td>$\emptyset$</td>
<td>${b}$</td>
</tr>
<tr>
<td>${a, b}$</td>
<td>$\emptyset$</td>
<td>${b}$</td>
</tr>
<tr>
<td>${a, b}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter. #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a = b = 5$</td>
<td>${a, b}$</td>
</tr>
<tr>
<td>${a, b}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>${a, b}$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
How Does the Initialization Affect the Solution?

\[ a = b = 5 \]

Init. | Iter. #1 | Iter. #2
--- | --- | ---
∅ | ∅ | ∅
∅ | \{b\} | \{b\}
∅ | ∅ | ∅
∅ | ∅ | ∅

Init. | Iter. #1
--- | ---
\{a, b\} | \{a, b\}
\{a, b\} | \{a, b\}
\{a, b\} | ∅
∅ | ∅

Dec 2019 IIT Bombay
How Does the Initialization Affect the Solution?

\[
\begin{align*}
\text{Init.} & \quad \text{Iter.} \ #1 & \quad \text{Iter.} \ #2 \\
\emptyset & \quad \emptyset & \quad \emptyset \\
\emptyset & \quad \{b\} & \quad \{b\} \\
\emptyset & \quad \emptyset & \quad \{b\} \\
\emptyset & \quad \emptyset & \quad \emptyset \\
\end{align*}
\]

\[
\begin{align*}
\text{Init.} & \quad \text{Iter.} \ #1 \\
\{a, b\} & \quad \{a, b\} \\
\{a, b\} & \quad \{a, b\} \\
\{a, b\} & \quad \emptyset \\
\emptyset & \quad \emptyset \\
\end{align*}
\]
How Does the Initialization Affect the Solution?

\[ a = b = 5 \]

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter. #1</th>
<th>Iter. #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \emptyset )</td>
<td>{b}</td>
<td>{b}</td>
</tr>
<tr>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>{b}</td>
</tr>
<tr>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter. #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( {a, b} )</td>
<td>{a, b}</td>
</tr>
<tr>
<td>( {a, b} )</td>
<td>{a, b}</td>
</tr>
<tr>
<td>( {a, b} )</td>
<td>{a, b}</td>
</tr>
<tr>
<td>( {a, b} )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
How Does the Initialization Affect the Solution?

\[ a = b = 5 \]

<table>
<thead>
<tr>
<th></th>
<th>Init.</th>
<th>Iter. #1</th>
<th>Iter. #2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td></td>
<td>( \emptyset )</td>
<td>{( b )}</td>
<td>{( b )}</td>
</tr>
<tr>
<td></td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>{( b )}</td>
</tr>
<tr>
<td></td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td></td>
<td>( \emptyset )</td>
<td>{( b )}</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Init.</th>
<th>Iter. #1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{( a, b )}</td>
<td>{( a, b )}</td>
</tr>
<tr>
<td></td>
<td>{( a, b )}</td>
<td>{( a, b )}</td>
</tr>
<tr>
<td></td>
<td>{( a, b )}</td>
<td>{( a, b )}</td>
</tr>
<tr>
<td></td>
<td>{( a, b )}</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>

Dec 2019 IIT Bombay
How Does the Initialization Affect the Solution?

\[
a = b = 5
\]

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter. #1</th>
<th>Iter. #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>(\emptyset)</td>
<td>({b})</td>
<td>({b})</td>
</tr>
<tr>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>({b})</td>
</tr>
<tr>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
</tr>
</tbody>
</table>

\[
a = b = 5
\]

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter. #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>({a, b})</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>({a, b})</td>
<td>({a, b})</td>
</tr>
<tr>
<td>({a, b})</td>
<td>({a, b})</td>
</tr>
<tr>
<td>({a, b})</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
</tr>
</tbody>
</table>
How Does the Initialization Affect the Solution?

\[ a = b = 5 \]

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter. #1</th>
<th>Iter. #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>\emptyset</td>
<td>{b}</td>
<td>{b}</td>
</tr>
<tr>
<td>\emptyset</td>
<td>{b}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>\emptyset</td>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
<tr>
<td>\emptyset</td>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>

\[ a = b = 5 \]

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter. #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{a, b}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>{a, b}</td>
<td>{a, b}</td>
</tr>
<tr>
<td>{a, b}</td>
<td>{a, b}</td>
</tr>
<tr>
<td>{a, b}</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>

\[ a \text{ is spuriously marked live} \]
Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information
Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

- Spurious inclusion of a non-live variable

```
x = y + 10
```

```
Out_i = \{ x, y \}
```

```
print y
```

```
End
```
Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

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

\[
x = y + 10
\]

\[
\text{print } y
\]

\[
\text{Out}_i = \{x, y\}
\]

\[
\text{End}
\]
Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

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

- Spurious exclusion of a live variable

\[
\begin{align*}
  x &= y + 10 & i \\
  \text{print } y & & j \\
  \text{End}
\end{align*}
\]

\[
\begin{align*}
  x &= z + 10 & i \\
  \text{print } x, y & & j \\
  \text{End}
\end{align*}
\]
Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

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

- Spurious exclusion of a live variable
  - A useful assignment may be eliminated
  - Solution is unsound
Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

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

- Spurious exclusion of a live variable
  - A useful assignment may be eliminated
  - Solution is unsound

- Given $L_2 \supseteq L_1$ representing liveness information
  - Using $L_2$ in place of $L_1$ is sound
  - Using $L_1$ in place of $L_2$ may not be sound
Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

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

- Spurious exclusion of a live variable
  - A useful assignment may be eliminated
  - Solution is unsound

- Given $L_2 \supseteq L_1$ representing liveness information
  - Using $L_2$ in place of $L_1$ is sound
  - Using $L_1$ in place of $L_2$ may not be sound

- The smallest set of all live variables is most precise
  - Since liveness sets grow (confluence is $\cup$), we choose $\emptyset$ as the initial conservative value
Termination, Convergence, and Complexity

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

⇒ Termination is guaranteed
Termination, Convergence, and Complexity

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

→ Termination is guaranteed

• Since initial value is $\emptyset$, live variables analysis converges on the smallest set
Termination, Convergence, and Complexity

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

⇒ Termination is guaranteed

• Since initial value is ∅, live variables analysis converges on the smallest set

• How many iterations do we need for reaching the convergence?
Termination, Convergence, and Complexity

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

⇒ Termination is guaranteed

• Since initial value is $\emptyset$, live variables analysis converges on the smallest set

• How many iterations do we need for reaching the convergence?

• Going beyond live variables analysis
  - Do the sets always grow for other data flow frameworks?
  - What is the complexity of round robin analysis for other analyses?
Termination, Convergence, and Complexity

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

  ⇒ Termination is guaranteed

• Since initial value is $\emptyset$, live variables analysis converges on the smallest set

• How many iterations do we need for reaching the convergence?

• Going beyond live variables analysis
  - Do the sets always grow for other data flow frameworks?
  - What is the complexity of round robin analysis for other analyses?

Answered formally in module 2 (Theoretical Abstractions)
Conservative Nature of Analysis (1)

\[
\begin{align*}
x &= \text{abs}(x) \\
\text{if } (x < 0) \\
T & \quad F \\
x &= a + y \\
x &= a + z \\
\text{print } (x)
\end{align*}
\]
Conservative Nature of Analysis (1)

- \(\text{abs}(n)\) returns the absolute value of \(n\)
Conservative Nature of Analysis (1)

- \( \text{abs}(n) \) returns the absolute value of \( n \)
- Is \( y \) live on entry to block \( b2 \)?
Conservative Nature of Analysis (1)

- abs(n) returns the absolute value of n
- Is y live on entry to block b2?
  - By execution semantics, NO
  - Path b1→b2→b3 is an infeasible execution path
Conservative Nature of Analysis (1)

- $\text{abs}(n)$ returns the absolute value of $n$
- Is $y$ live on entry to block $b2$?
- By execution semantics, NO
  Path $b1 \rightarrow b2 \rightarrow b3$ is an infeasible execution path
- A compiler makes conservative assumptions:
  *All branch outcomes are possible*
  $\Rightarrow$ Consider every path in CFG as a potential execution path
Conservative Nature of Analysis (1)

- abs(n) returns the absolute value of n
- Is y live on entry to block b2?
- By execution semantics, NO
  Path b1 \(\rightarrow\) b2 \(\rightarrow\) b3 is an infeasible execution path
- A compiler makes conservative assumptions:
  *All branch outcomes are possible*
  \(\Rightarrow\) Consider every path in CFG as a potential execution path
- Our analysis concludes that y is live on entry to block b2
Conservative Nature of Analysis (2)

```
if (x < 0) { b1 }
  if (x < 0) { b4 }
    a = a + y, x = a + z { b2, b3 }
  x = c + 1, x = d + 1 { b5, b6 }
print x { b7 }
```
Conservative Nature of Analysis (2)

- Is d live on entry to block b2?
Conservative Nature of Analysis (2)

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

Path b₁→b₂→b₄→b₆ is an infeasible execution path
Conservative Nature of Analysis (2)

- Is \( d \) live on entry to block \( b2 \)?
  - By execution semantics, NO
  - Path \( b1 \rightarrow b2 \rightarrow b4 \rightarrow b6 \) is an infeasible execution path

- Is \( c \) live on entry to block \( b3 \)?
  - Path \( b1 \rightarrow b3 \rightarrow b4 \rightarrow b6 \) is a feasible execution path
Conservative Nature of Analysis (2)

- Is d live on entry to block b2?
  - By execution semantics, NO
  - Path b1→b2→b4→b6 is an infeasible execution path

- Is c live on entry to block b3?
  - Path b1→b3→b4→b6 is a feasible execution path

- A compiler makes conservative assumptions ⇒ our analysis is *path insensitive*
  - Note: It is *flow sensitive* (i.e. information is computed for every control flow point)
Conservative Nature of Analysis (2)

- Is d live on entry to block b2?
  - By execution semantics, NO
  - Path b1→b2→b4→b6 is an infeasible execution path

- Is c live on entry to block b3?
  - Path b1→b3→b4→b6 is a feasible execution path

- A compiler make conservative assumptions ⇒ our analysis is *path insensitive*
  - Note: It is *flow sensitive* (i.e. information is computed for every control flow points)

- Our analysis concludes that d is live at the entry of b2
### Conservative Nature of Analysis (2)

- **Is d live on entry to block b2?**
  - By execution semantics, NO
  - Path $b1 \rightarrow b2 \rightarrow b4 \rightarrow b6$ is an infeasible execution path

- **Is c live on entry to block b3?**
  - Path $b1 \rightarrow b3 \rightarrow b4 \rightarrow b6$ is a feasible execution path

- A compiler makes conservative assumptions $\Rightarrow$ our analysis is *path insensitive*
  - Note: It is *flow sensitive* (i.e. information is computed for every control flow points)

- Our analysis concludes that d is live at the entry of b2

- **Is c live at the entry of b3?**
Conservative Nature of Analysis at Intraprocedural Level

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

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

More about it later
Part 4

**Strongly Live Variables Analysis**
Strongly Live Variables Analysis

• A variable is strongly live if
  ○ it is used in a statement other than assignment statement, or
    (same as simple liveness)
  ○ it is used in an assignment statement defining a variable that is
    strongly live
    (different from simple liveness)

• Killing: An assignment statement, an input statement, or BI
  (this is same as killing in simple liveness)

• Generation: A direct use or a use for defining values that are strongly live
  (this is different from generation in simple liveness)
Understanding Strong Liveness

\[
\begin{align*}
y &= x \\
\text{print (}x\text{)}
\end{align*}
\]

\[
\begin{align*}
y &= x \\
\text{print (}y\text{)}
\end{align*}
\]

\[
\begin{align*}
y &= x \\
\text{print (}z\text{)}
\end{align*}
\]
Understanding Strong Liveness

\[ y = x \]

\[ \text{print (} x \text{)} \]

\[ y = x \]

\[ \text{print (} x \text{)} \]

\[ y = x \]

\[ \text{print (} y \text{)} \]

\[ y = x \]

\[ \text{print (} z \text{)} \]
Understanding Strong Liveness

\[
y = x
\]

print (x)

\[
\emptyset
\]
Understanding Strong Liveness

```plaintext
Strong Liveness

\[
\begin{align*}
  y &= x \\
  \{x\} \\
  \text{print (x)} \\
  \emptyset
\end{align*}
\]

\[
\begin{align*}
  y &= x \\
  \text{print (y)}
\end{align*}
\]

\[
\begin{align*}
  y &= x \\
  \text{print (z)}
\end{align*}
\]

Dec 2019
Understanding Strong Liveness

\[ y = x \]
\[ \{x\} \]
\[ \text{print}(x) \]
\[ \emptyset \]

\[ y = x \]
\[ \{x\} \]
\[ \text{print}(y) \]

\[ y = x \]
\[ \text{print}(z) \]
Understanding Strong Liveness

Simple Liveness

\{x\} \rightarrow \{x\}

\begin{align*}
    y &= x \\
    \{x\} &\rightarrow \{x\} \\
    \text{print } (x)
\end{align*}

\emptyset \rightarrow \emptyset

Strong Liveness

\begin{align*}
    y &= x \\
    \{x\} &\rightarrow \{x\} \\
    \text{print } (y)
\end{align*}

\begin{align*}
    y &= x \\
    \{x\} &\rightarrow \{x\} \\
    \text{print } (z)
\end{align*}
Understanding Strong Liveness

Simple Liveness

\[ \{x\} \]
\[ y = x \]
\[ \{x\} \]
\[ \{x\} \]
\[ \text{print(} x \text{)} \]
\[ \emptyset \]
\[ \emptyset \]

Strong Liveness

\[ \{x\} \]
\[ y = x \]
\[ \{x\} \]
\[ \{x\} \]
\[ \text{print(} y \text{)} \]
\[ \emptyset \]
\[ \emptyset \]

\[ y = x \]
\[ \text{print(} z \text{)} \]

Same
Understanding Strong Liveness

**Simple Liveness**

- \(\{x\}\)
  - \(y = x\)
  - \(\{x\}\)
  - \(\text{print} (x)\)
  - \(\emptyset\)

**Strong Liveness**

- \(\{x\}\)
  - \(y = x\)
  - \(\{x\}\)
  - \(\text{print} (x)\)
  - \(\emptyset\)

**Same Dec 2019 IIT Bombay**
Understanding Strong Liveness

Simple Liveness: $\{x\} \rightarrow y = x \rightarrow \{x\} \rightarrow \text{print}(x) \rightarrow \emptyset$

Strong Liveness: $\{x\} \rightarrow y = x \rightarrow \{x\} \rightarrow \text{print}(x) \rightarrow \emptyset$

Same
Understanding Strong Liveness

Simple Liveness
\[ \{x\} \]
\[ y = x \]
\[ \{x\} \]
\[ \text{print}(x) \]
\[ \emptyset \]

Strong Liveness
\[ \{x\} \]
\[ y = x \]
\[ \{x\} \]
\[ \text{print}(x) \]
\[ \emptyset \]

Same

Strong Liveness
\[ y = x \]
\[ \{y\} \]
\[ \text{print}(y) \]
\[ \emptyset \]

Strong Liveness
\[ y = x \]
\[ \{x\} \]
\[ \text{print}(z) \]
Understanding Strong Liveness

Simple Liveness

\[ \{x\} \]

\[ y = x \]

\[ \{x\} \]

\[ \text{print } (x) \]

\[ \emptyset \]

Strong Liveness

\[ \{x\} \]

\[ y = x \]

\[ \{x\} \]

\[ \text{print } (x) \]

\[ \emptyset \]

\[ \emptyset \]

Same

Strong Liveness

\[ \{x\} \]

\[ y = x \]

\[ \{x\} \]

\[ \text{print } (y) \]

\[ \emptyset \]

\[ \emptyset \]

\[ \emptyset \]

Strong Liveness

\[ \{x\} \]

\[ y = x \]

\[ \{y\} \]

\[ \text{print } (z) \]

\[ \emptyset \]

\[ \emptyset \]
Understanding Strong Liveness

Simple Liveness | Strong Liveness
\{x\} \downarrow \{x\}
\{x\} \downarrow \{x\}
\text{print}(x) \downarrow \emptyset \downarrow \emptyset
Same

Simple Liveness | Strong Liveness
\{x\} \downarrow \{x\}
\{y\} \downarrow \{y\}
\text{print}(y) \downarrow \emptyset \downarrow \emptyset

\text{print}(z)
Understanding Strong Liveness

Simple Liveness

\[
\{ x \} \xrightarrow{y = x} \{ x \} \xrightarrow{\text{print}(x)} \emptyset
\]

Strong Liveness

\[
\{ x \} \xrightarrow{y = x} \{ x \} \xrightarrow{\text{print}(x)} \emptyset
\]

Simple Liveness

\[
\{ x \} \xrightarrow{y = x} \{ x \} \xrightarrow{\text{print}(y)} \emptyset
\]

Strong Liveness

\[
\{ x \} \xrightarrow{y = x} \{ x \} \xrightarrow{\text{print}(y)} \emptyset
\]

\[
\{ y \} \xrightarrow{y = x} \{ y \} \xrightarrow{\text{print}(z)} \emptyset
\]

Same
Understanding Strong Liveness

Simple Liveness

<table>
<thead>
<tr>
<th>x</th>
<th>y = x</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x}</td>
<td>{x}</td>
<td></td>
</tr>
</tbody>
</table>

Strong Liveness

<table>
<thead>
<tr>
<th>x</th>
<th>y = x</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x}</td>
<td>{x}</td>
<td></td>
</tr>
</tbody>
</table>

Simple Liveness

<table>
<thead>
<tr>
<th>x</th>
<th>y = x</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>{y}</td>
<td>y</td>
</tr>
</tbody>
</table>

Strong Liveness

<table>
<thead>
<tr>
<th>y = x</th>
<th>x</th>
</tr>
</thead>
</table>

Dec 2019
Understanding Strong Liveness

Simple Liveness | Strong Liveness | Simple Liveness | Strong Liveness | Strong Liveness
\[
\{x\} \quad \{x\} \\
\{x\} \quad \{x\} \\
\text{print}(x) \\
\emptyset \quad \emptyset
\]

\[
\{y\} \quad \{x\} \\
\{y\} \quad \{y\} \\
\text{print}(y) \\
\emptyset \quad \emptyset
\]

\[
y = x \\
\emptyset \quad \emptyset
\]

Same | Same |
Understanding Strong Liveness

Simple Liveness | Strong Liveness | Simple Liveness | Strong Liveness | Strong Liveness

\[ \{x\} \xrightarrow{y = x} \{x\} \]  
\[ \{x\} \xrightarrow{\text{print } (x)} \{x\} \]  
\[ \{0\} \]  

\[ \{y\} \xrightarrow{y = x} \{y\} \]  
\[ \{y\} \xrightarrow{\text{print } (y)} \{y\} \]  
\[ \{0\} \]  

\[ y = x \]  
\[ \{z\} \xrightarrow{\text{print } (z)} \{\} \]  

Same  
Same  

Dec 2019
Understanding Strong Liveness

Simple Liveness
\{x\} → \{x\} → print(x) → ∅

Strong Liveness
\{x\} → \{x\} → y = x → \{x\} → ∅

Simple Liveness
\{x\} → \{x\} → print(x) → ∅

Strong Liveness
\{x\} → \{x\} → y = x → \{x\} → print(y) → ∅

Strong Liveness
y = x → \{z\} → print(z) → ∅
Understanding Strong Liveness

Simple Liveness | Strong Liveness
\[
\{ x \} \quad \rightarrow \quad \{ x \} \\
\{ x \} \quad \downarrow \quad \{ x \} \\
\text{print (}x\text{)} \quad \downarrow \quad \downarrow \\
\emptyset \quad \emptyset
\]

Simple Liveness | Strong Liveness
\[
\{ x \} \quad \rightarrow \quad \{ x \} \\
\{ y \} \quad \downarrow \quad \{ y \} \\
\text{print (}y\text{)} \quad \downarrow \quad \downarrow \\
\emptyset \quad \emptyset
\]

Simple Liveness | Strong Liveness
\[
\{ z, x \} \quad \rightarrow \quad \{ z \} \\
\{ z \} \quad \downarrow \quad \{ z \} \\
\text{print (}z\text{)} \quad \downarrow \quad \downarrow \\
\emptyset \quad \emptyset
\]
Understanding Strong Liveness

Simple Liveness | Strong Liveness
---|---
$$\{x\}$$ | $$\{x\}$$
$$\{x\}$$ | $$\{x\}$$
print ($$x$$) | print ($$x$$)
$$\emptyset$$ | $$\emptyset$$
Same | Same

Simple Liveness | Strong Liveness
---|---
$$\{x\}$$ | $$\{x\}$$
$$\{y\}$$ | $$\{y\}$$
print ($$y$$) | print ($$y$$)
$$\emptyset$$ | $$\emptyset$$
Same | Same

Simple Liveness | Strong Liveness
---|---
$$\{z, x\}$$ | $$\{z\}$$
$$\{z\}$$ | $$\{z\}$$
print ($$z$$) | print ($$z$$)
$$\emptyset$$ | $$\emptyset$$
Different | Different
Live Variables Analysis: Simple and Strong Liveness

• A variable is live at a program point if its current value is likely to be used later
Live Variables Analysis: Simple and Strong Liveness

- A variable is live at a program point if its current value is likely to be used later
- We want to compute the smallest set of variables that are live
Live Variables Analysis: Simple and Strong Liveness

- A variable is live at a program point if its current value is likely to be used later.
- We want to compute the smallest set of variables that are live.

```
a = 1; b = 2
c = 3; n = 6
B1

if a <= n
B2

a = a + 1
B3

if a <= 11
B4

t1 = a + b
a = t1 + c
print "Hello"
B5

print "Hi"
B6
```
Live Variables Analysis: Simple and Strong Liveness

- A variable is live at a program point if its current value is likely to be used later.
- We want to compute the smallest set of variables that are live.
- Simple liveness considers every use of a variable as useful.

```
a = 1; b = 2
c = 3; n = 6
B1
```
```
if a \leq n
B2
```
```
a = a + 1
```
```
if a \leq 11
B4
```
```
t1 = a + b
a = t1 + c
print "Hello"
```
```
print "Hi"
B6
```

Live Variables Analysis: Simple and Strong Liveness

- A variable is live at a program point if its current value is likely to be used later.
- We want to compute the smallest set of variables that are live.
- Simple liveness considers every use of a variable as useful.
- Strong liveness checks the liveness of the result before declaring the operands to be live.

```
a = 1; b = 2
c = 3; n = 6
```

```
a = a + 1
```

```
if a ≤ n
```

```
t1 = a + b
a = t1 + c
print "Hello"
```

```
print "Hi"
```
Live Variables Analysis: Simple and Strong Liveness

• A variable is live at a program point if its current value is likely to be used later

• We want to compute the smallest set of variables that are live

• Simple liveness considers every use of a variable as useful

• Strong liveness checks the liveness of the result before declaring the operands to be live

• Strong liveness is more precise than simple liveness
**Data Flow Equations for Strongly Live Variables Analysis**

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

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

where,

\[ f_n(X) = \begin{cases} (X - \{y\}) \cup (\text{Opd}(e) \cap \text{Var}) & n \text{ is } y = e, e \in \text{Expr}, \ y \in X \\ X - \{y\} & n \text{ is input}(y) \\ X \cup \{y\} & n \text{ is use}(y) \\ X & \text{otherwise} \end{cases} \]
Data Flow Equations for Strongly Live Variables Analysis

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

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

where,

\[ f_n(X) = \begin{cases} (X - \{y\}) \cup (\text{Opd}(e) \cap \text{Var}) & \text{n is } y = e, e \in \text{Expr}, y \in X \\ X - \{y\} & \text{n is input}(y) \\ X \cup \{y\} & \text{n is use}(y) \\ X & \text{otherwise} \end{cases} \]

If \( y \) is not strongly live, the assignment is skipped using the "otherwise" clause.
Tutorial Problem for strongly Live Variables Analysis

```
a = 0

if a ≥ 2

a = b
b = c

a = a + 1

Dec 2019 IIT Bombay
```
### Result of Strongly Live Variables Analysis

<table>
<thead>
<tr>
<th>Node</th>
<th>Iteration #1</th>
<th>Iteration #2</th>
<th>Iteration #3</th>
<th>Iteration #4</th>
</tr>
</thead>
<tbody>
<tr>
<td>n_6</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>n_5</td>
<td>∅</td>
<td>{a}</td>
<td>{a, b}</td>
<td>{a, b, c}</td>
</tr>
<tr>
<td>n_4</td>
<td>∅</td>
<td>{a}</td>
<td>{a, b}</td>
<td>{a, b, c}</td>
</tr>
<tr>
<td>n_3</td>
<td>∅</td>
<td>{a}</td>
<td>{a, b}</td>
<td>{a, b, c}</td>
</tr>
<tr>
<td>n_2</td>
<td>{a}</td>
<td>{a, b}</td>
<td>{a, b, c}</td>
<td>{a, b, c}</td>
</tr>
<tr>
<td>n_1</td>
<td>{a}</td>
<td>∅</td>
<td>{a, b}</td>
<td>{a, b, c}</td>
</tr>
</tbody>
</table>

Node labels: n_6, n_5, n_4, n_3, n_2, n_1.