Some Generalizations

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

- Partial Redundancy Elimination
- Introduction to Constant Propagation
- Theoretical Abstractions in Data Flow Analysis
  - The world of data flow values
  - The world of functions and operations that compute data values (Not today)
  - Results of data flow analysis (Not today)
  - Algorithms for performing data flow analysis (Not today)
Part 2

Partial Redundancy Elimination
Precursor: Common Subexpression Elimination

0 if ()

1 $a \ast b$

2 $a \ast b$

3 $a \ast b$
Precursor: Common Subexpression Elimination

- $a$ and $b$ are not modified along paths 1 $\rightarrow$ 3 and 2 $\rightarrow$ 3
Precursor: Common Subexpression Elimination

- $a$ and $b$ are not modified along paths $1 \rightarrow 3$ and $2 \rightarrow 3$
- Computation of $a \times b$ in 3 is redundant
Precursor: Common Subexpression Elimination

- \( t = a \times b \)
- \( t = a \times b \)
- \( t \)

0: if ()

1: \( t = a \times b \)

2: \( t = a \times b \)

3: \( t \)

- \( a \) and \( b \) are not modified along paths 1 \( \rightarrow \) 3 and 2 \( \rightarrow \) 3
- Computation of \( a \times b \) in 3 is redundant
- Previous value can be used
Partial Redundancy Elimination

0: if ()
1: $a \times b$
2: $a = 5$
3: $a \times b$
Partial Redundancy Elimination

- Computation of $a \times b$ in 3 is

![Diagram](attachment:image.png)
Partial Redundancy Elimination

- Computation of $a \times b$ in 3 is redundant along path 1 → 3, but ...
Partial Redundancy Elimination

- Computation of $a \ast b$ in 3 is
  - redundant along path $1 \rightarrow 3$, but . . .
  - not redundant along path $2 \rightarrow 3$
Code Hoisting for Partial Redundancy Elimination

```
0: if ()
1: a * b
2: a = 5
3: a * b
```
Code Hoisting for Partial Redundancy Elimination

- Computation of $a \times b$ in 3 becomes totally redundant
- Can be deleted
PRE Subsumes Loop Invariant Movement

1

2 \[ a = b \times c \]

3
PRE Subsumes Loop Invariant Movement

\[ a = b \times c \]
PRE Subsumes Loop Invariant Movement

1. $a = b \times c$
2. $a = t$
3. $t = b \times c$

1. $a = b \times c$
2. $a = b \times c$
3. $a = b \times c$

May 2011  
Uday Khedker
PRE Can be Used for Strength Reduction

\[ i = 0 \]
\[ t_0 = \text{base}(A) \]

\[ t_1 = t_0 + i \times 4 \]
\[ a = \text{mem}[t_1] \]
\[ i = i + 1 \]
PRE Can be Used for Strength Reduction

\[
i = 0 \\
t_0 = base(A)
\]

\[
t_1 = t_0 + i \times 4 \\
a = mem[t_1] \\
i = i + 1
\]

\[
\Rightarrow
\]

\[
i = 0 \\
t_0 = base(A) \\
t_1 = t_0 + i \times 4
\]

\[
t_1 = t_1 + 4 \\
a = mem[t_1] \\
i = i + 1
\]

- * and + in the loop have been replaced by +
- \( i = i + 1 \) in the loop has been eliminated
PRE Can be Used for Strength Reduction

\[ i = 0 \]
\[ t_0 = \text{base}(A) \]

\[ t_1 = t_0 + i \times 4 \]
\[ a = \text{mem}[t_1] \]
\[ i = i + 1 \]

- Delete \( i = i + 1 \)
PRE Can be Used for Strength Reduction

- $i = 0$
- $t_0 = \text{base}(A)$

- $t_1 = t_0 + i \times 4$
- $a = \text{mem}[t_1]$
- $i = i + 1$

- Delete $i = i + 1$
- Expression $t_0 + i \times 4$ becomes loop invariant
PRE Can be Used for Strength Reduction

\begin{align*}
  i &= 0 \\
  t0 &= base(A) \\
  t1 &= t0 + i \times 4 \\
  t1 &= t1 + 4 \\
  a &= mem[t1] \\
  i &= i + 1
\end{align*}

- Delete $i = i + 1$
- Expression $t0 + i \times 4$ becomes loop invariant
- Hoist it and increment $t1$ in the loop
PRE Can be Used for Strength Reduction

\[
i = 0 \\
t_0 = base(A) \\
t_1 = t_0 + i \times 4
\]

\[
t_1 = t_1 + 4 \\
a = mem[t_1] \\
i = i + 1
\]

\[
i = 0 \\
t_0 = base(A) \\
t_1 = t_0 + i \times 4
\]

\[
t_1 = t_1 + 4 \\
a = mem[t_1] \\
i = i + 1
\]

- \* and + in the loop have been replaced by +
- \(i = i + 1\) in the loop has been eliminated
Performing Partial Redundancy Elimination

1. Identify partial redundancies
2. Identify program points where computations can be inserted
3. Insert expressions
4. Partial redundancies become total redundancies
   ⇒ Delete them.

Morel-Renvoise Algorithm (CACM, 1979.)
Defining Hoisting Criteria

- An expression can be safely inserted at a program point $p$ if it is 

**Available at $p$**
Defining Hoisting Criteria

• An expression can be safely inserted at a program point $p$ if it is

Available at $p$  Anticipable at $p$

```
Start
\[ a \times b \]
Start
\[ a \times b \]
```

```
End
\[ a \times b \]
End
\[ a \times b \]
```
Defining Hoisting Criteria

- An expression can be safely inserted at a program point $p$ if it is

  **Available at $p$**

  ![Diagram showing available at $p$]

  **Anticipable at $p$**

  ![Diagram showing anticipable at $p$]

  - If it is available at $p$, then there is no need to insert it at $p$. 

  $a \times b$

  $a \times b$

  $a \times b$

  $a \times b$
Defining Hoisting Criteria

- An expression can be safely inserted at a program point $p$ if it is
  
  **Available at $p$**
  
  ![Diagram showing available at $p$]

  **Anticipable at $p$**
  
  ![Diagram showing anticipable at $p$]

  - If it is available at $p$, then there is no need to insert it at $p$.
  - If it is anticipable at $p$ then all such occurrence should be hoisted to $p$. 

  \[ a \times b \]

  \[ a \times b \]
Defining Hoisting Criteria

- An expression can be safely inserted at a program point $p$ if it is

**Available at $p$**

- If it is available at $p$, then there is no need to insert it at $p$.

**Anticipable at $p$**

- If it is anticipable at $p$ then all such occurrence should be hoisted to $p$.
- An expression should be hoisted to $p$ provided it can be hoisted to $p$ along all paths from $p$ to exit.
Hoisting Criteria

- Safety of hoisting to the exit of a block.

  S.1 Should be hoisted only if it can be hoisted to the entry of all successors
Hoisting Criteria

- Safety of hoisting to the exit of a block.

S.1 Should be hoisted only if it can be hoisted to the entry of all successors
Hoisting Criteria

• *Safety of hoisting to the exit of a block.*
  
  S.1 Should be hoisted only if it can be hoisted to the entry of all successors

• *Safety of hoisting to the entry of a block.*
  Should be hoisted only if
  
  S.2 it is upwards exposed, or
Hoisting Criteria

- **Safety of hoisting to the exit of a block.**
  
  S.1 Should be hoisted only if it can be hoisted to the entry of all successors

- **Safety of hoisting to the entry of a block.**
  Should be hoisted only if
  
  S.2 it is upwards exposed, or

\[
a \ast c = a
\]
Hoisting Criteria

- **Safety of hoisting to the exit of a block.**
  
  S.1 Should be hoisted only if it can be hoisted to the entry of all successors

- **Safety of hoisting to the entry of a block.**
  
  Should be hoisted only if
  
  S.2 it is upwards exposed, or
  S.3 it can be hoisted to its exit and is transparent in the block
Hoisting Criteria

- **Safety of hoisting to the exit of a block.**
  
  S.1 Should be hoisted only if it can be hoisted to the entry of all successors

- **Safety of hoisting to the entry of a block.**
  
  Should be hoisted only if
  
  S.2 it is upwards exposed, or
  
  S.3 it can be hoisted to its exit and is transparent in the block
Hoisting Criteria

- **Safety of hoisting to the exit of a block.**
  - S.1 Should be hoisted only if it can be hoisted to the entry of all successors

- **Safety of hoisting to the entry of a block.**
  Should be hoisted only if
  - S.2 it is upwards exposed, or
  - S.3 it can be hoisted to its exit and is transparent in the block

- **Desirability of hoisting to the entry of a block.**
  Should be hoisted only if
  - D.1 it is partially available, and
  - D.2 For each predecessor
Hoisting Criteria

- **Safety of hoisting to the exit of a block.**
  
  S.1 Should be hoisted only if it can be hoisted to the entry of all successors

- **Safety of hoisting to the entry of a block.**
  Should be hoisted only if
  
  S.2 it is upwards exposed, or
  S.3 it can be hoisted to its exit and is transparent in the block

- **Desirability of hoisting to the entry of a block.**
  Should be hoisted only if
  
  D.1 it is partially available, and
  D.2 For each predecessor
    
    D.2.a it is hoisted to its exit, or
Hoisting Criteria

- **Safety of hoisting to the exit of a block.**
  - **S.1** Should be hoisted only if it can be hoisted to the entry of all successors

- **Safety of hoisting to the entry of a block.**
  Should be hoisted only if
  - **S.2** it is upwards exposed, or
  - **S.3** it can be hoisted to its exit and is transparent in the block

- **Desirability of hoisting to the entry of a block.**
  Should be hoisted only if
  - **D.1** it is partially available, and
  - **D.2** For each predecessor
    - **D.2.a** it is hoisted to its exit, or
    - **D.2.b** is available at its exit.
Hoisting Criteria

- **Safety of hoisting to the exit of a block.**
  - S.1 Should be hoisted only if it can be hoisted to the entry of all successors

- **Safety of hoisting to the entry of a block.**
  Should be hoisted only if
  - S.2 it is upwards exposed, or
  - S.3 it can be hoisted to its exit and is transparent in the block

- **Desirability of hoisting to the entry of a block.**
  Should be hoisted only if
  - D.1 it is partially available, and
  - D.2 For each predecessor
    - D.2.a it is hoisted to its exit, or
    - D.2.b is available at its exit.
Applying the Hoisting Criteria

- **Safety of hoisting to the exit of a block.**
  
  S.1 Should be hoisted only if it can be hoisted to the entry of all successors

- **Safety of hoisting to the entry of a block.**
  
  Should be hoisted only if
  
  S.2 it is upwards exposed, or
  S.3 it can be hoisted to its exit and is transparent in the block

- **Desirability of hoisting to the entry of a block.**
  
  Should be hoisted only if
  
  D.1 it is partially available, and
  D.2 For each predecessor
    
    D.2.a it is hoisted to its exit, or
    D.2.b is available at its exit.

What does this slide show?

- Four examples
- For each example
  - statements in blue enable hoisting
  - statements in red prohibit hoisting
Applying the Hoisting Criteria

- **Safety of hoisting to the exit of a block.**
  
  S.1 Should be hoisted only if it can be hoisted to the entry of all successors

- **Safety of hoisting to the entry of a block.**
  Should be hoisted only if
  
  S.2 it is upwards exposed, or
  S.3 it can be hoisted to its exit and is transparent in the block

- **Desirability of hoisting to the entry of a block.**
  Should be hoisted only if
  
  D.1 it is partially available, and
  D.2 For each predecessor
    
    D.2.a it is hoisted to its exit, or
    D.2.b is available at its exit.

(Example 1)
Applying the Hoisting Criteria

- **Safety of hoisting to the exit of a block.**
  
  S.1 Should be hoisted only if it can be hoisted to the entry of all successors

- **Safety of hoisting to the entry of a block.**
  
  Should be hoisted only if
  
  S.2 it is upwards exposed, or
  
  S.3 it can be hoisted to its exit and is transparent in the block

- **Desirability of hoisting to the entry of a block.**
  
  Should be hoisted only if
  
  D.1 it is partially available, and
  
  D.2 For each predecessor
    
    D.2.a it is hoisted to its exit, or
    
    D.2.b is available at its exit.
Applying the Hoisting Criteria

- **Safety of hoisting to the exit of a block.**
  - S.1 Should be hoisted only if it can be hoisted to the entry of all successors

- **Safety of hoisting to the entry of a block.**
  - Should be hoisted only if
    - S.2 it is upwards exposed, or
    - S.3 it can be hoisted to its exit and is transparent in the block

- **Desirability of hoisting to the entry of a block.**
  - Should be hoisted only if
    - D.1 it is partially available, and
    - D.2 For each predecessor
      - D.2.a it is hoisted to its exit, or
      - D.2.b is available at its exit

What if we insert

- a = 2 in 3?
- a = 2 in 3 and a * b in 4?

(Example 1)
Applying the Hoisting Criteria

- **Safety of hoisting to the exit of a block.**
  
  S.1 Should be hoisted only if it can be hoisted to the entry of all successors

- **Safety of hoisting to the entry of a block.**
  Should be hoisted only if
  
  S.2 it is upwards exposed, or
  S.3 it can be hoisted to its exit and is transparent in the block

- **Desirability of hoisting to the entry of a block.**
  Should be hoisted only if
  
  D.1 it is partially available, and
  D.2 For each predecessor
    
    D.2.a it is hoisted to its exit, or
    D.2.b is available at its exit.
Applying the Hoisting Criteria

- **Safety of hoisting to the exit of a block.**
  - S.1 Should be hoisted only if it can be hoisted to the entry of all successors

- **Safety of hoisting to the entry of a block.**
  Should be hoisted only if
  - S.2 it is upwards exposed, or
  - S.3 it can be hoisted to its exit and is transparent in the block

- **Desirability of hoisting to the entry of a block.**
  Should be hoisted only if
  - D.1 it is partially available, and
  - D.2 For each predecessor
    - D.2.a it is hoisted to its exit, or
    - D.2.b is available at its exit.

(Example 2)
Applying the Hoisting Criteria

- **Safety of hoisting to the exit of a block.**
  - S.1 Should be hoisted only if it can be hoisted to the entry of all successors

- **Safety of hoisting to the entry of a block.**
  Should be hoisted only if
  - S.2 it is upwards exposed, or
  - S.3 it can be hoisted to its exit and is transparent in the block

- **Desirability of hoisting to the entry of a block.**
  Should be hoisted only if
  - D.1 it is partially available, and
  - D.2 For each predecessor
    - D.2.a it is hoisted to its exit, or
    - D.2.b is available at its exit.

(Example 2)
Applying the Hoisting Criteria

• **Safety of hoisting to the exit of a block.**
  
  S.1 Should be hoisted only if it can be hoisted to the entry of all successors

• **Safety of hoisting to the entry of a block.**
  Should be hoisted only if
  
  S.2 it is upwards exposed, or
  S.3 it can be hoisted to its exit and is transparent in the block

• **Desirability of hoisting to the entry of a block.**
  Should be hoisted only if
  
  D.1 it is partially available, and
  D.2 For each predecessor
    
    D.2.a it is hoisted to its exit, or
    D.2.b is available at its exit.

(Example 3)
Applying the Hoisting Criteria

- **Safety of hoisting to the exit of a block.**
  
  S.1 Should be hoisted only if it can be hoisted to the entry of all successors

- **Safety of hoisting to the entry of a block.**
  Should be hoisted only if
  
  S.2 it is upwards exposed, or
  S.3 it can be hoisted to its exit and is transparent in the block

- **Desirability of hoisting to the entry of a block.**
  Should be hoisted only if
  
  D.1 it is partially available, and
  D.2 For each predecessor
    
    D.2.a it is hoisted to its exit, or
    D.2.b is available at its exit.

(Example 3)
Applying the Hoisting Criteria

- **Safety of hoisting to the exit of a block.**
  
  S.1 Should be hoisted only if it can be hoisted to the entry of all successors

- **Safety of hoisting to the entry of a block.**
  Should be hoisted only if
  
  S.2 it is upwards exposed, or
  S.3 it can be hoisted to its exit and is transparent in the block

- **Desirability of hoisting to the entry of a block.**
  Should be hoisted only if
  
  D.1 it is partially available, and
  D.2 For each predecessor
    
    D.2.a it is hoisted to its exit, or
    D.2.b is available at its exit.

(Example 4)
Applying the Hoisting Criteria

- **Safety of hoisting to the exit of a block.**
  
  S.1 Should be hoisted only if it can be hoisted to the entry of all successors

- **Safety of hoisting to the entry of a block.**
  Should be hoisted only if
  
  S.2 it is upwards exposed, or
  S.3 it can be hoisted to its exit and is transparent in the block

- **Desirability of hoisting to the entry of a block.**
  Should be hoisted only if
  
  D.1 it is partially available, and
  D.2 For each predecessor
    
    D.2.a it is hoisted to its exit, or
    D.2.b is available at its exit.

(Example 4)
Applying the Hoisting Criteria

- **Safety of hoisting to the exit of a block.**
  - **S.1** Should be hoisted only if it can be hoisted to the entry of all successors

- **Safety of hoisting to the entry of a block.**
  - Should be hoisted only if
    - **S.2** it is upwards exposed, or
    - **S.3** it can be hoisted to its exit and is transparent in the block

- **Desirability of hoisting to the entry of a block.**
  - Should be hoisted only if
    - **D.1** it is partially available, and
    - **D.2** For each predecessor
      - **D.2.a** it is hoisted to its exit, or
      - **D.2.b** is available at its exit.

(Example 4)
Applying the Hoisting Criteria

- **S.1** Should be hoisted only if it can be hoisted to the entry of all successors

- Should be hoisted only if
  - **S.2** it is upwards exposed, or
  - **S.3** it can be hoisted to its exit and is transparent in the block

Should be hoisted only if

- **D.1** it is partially available, and
- **D.2** For each predecessor
  - **D.2.a** it is hoisted to its exit, or
  - **D.2.b** is available at its exit.

(Example 4)
First Level Global Data Flow Properties in PRE

- Partial Availability.

\[
PavIn_n = \begin{cases} 
  BI & \text{if } n \text{ is Start block} \\
  \bigcup_{p \in \text{pred}(n)} PavOut_p & \text{otherwise}
\end{cases}
\]

\[
PavOut_n = Gen_n \cup (PavIn_n - Kill_n)
\]

- Total Availability.

\[
AvIn_n = \begin{cases} 
  BI & \text{if } n \text{ is Start block} \\
  \bigcap_{p \in \text{pred}(n)} AvOut_p & \text{otherwise}
\end{cases}
\]

\[
AvOut_n = Gen_n \cup (AvIn_n - Kill_n)
\]
PRE Data Flow Equations

Desirability: D.1

\[ \text{In}_n = \text{PavIn}_n \cap \left( \text{AntGen}_n \cup \left( \text{Out}_n - \text{Kill}_n \right) \right) \]

\[ \bigcap_{p \in \text{pred}(n)} \left( \text{Out}_p \cup \text{AvOut}_p \right) \]

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

Expressions should be partially available, and
**PRE Data Flow Equations**

$$In_n = PavIn_n \cap \left( \text{AntGen}_n \cup (Out_n - Kill_n) \right)$$

$$\bigcap_{p \in \text{pred}(n)} \left( Out_p \cup \text{AvOut}_p \right)$$

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

Expressions should be upwards exposed, or
**PRE Data Flow Equations**

\[ \begin{align*}
    \text{In}_n &= P_{av}\text{In}_n \cap \left( \text{AntGen}_n \cup (\text{Out}_n - \text{Kill}_n) \right) \\
    \bigcap_{p \in \text{pred}(n)} &\left( \text{Out}_p \cup \text{AvOut}_p \right) \\
    \text{Out}_n &= \begin{cases} 
        \text{BI} & \text{if } n \text{ is End block} \\
        \bigcap_{s \in \text{succ}(n)} \text{In}_s & \text{otherwise}
    \end{cases}
\end{align*} \]

*Expressions can be hoisted to the exit and are transparent in the block*
PRE Data Flow Equations

\[ \text{Desirability: D.2.a} \]

\[ \begin{align*}
\text{In}_n &= PAV\text{In}_n \cap \left( \text{AntGen}_n \cup (\text{Out}_n - \text{Kill}_n) \right) \\
& \quad \bigcap_{p \in \text{pred}(n)} \left( \text{Out}_p \cup \text{AvOut}_p \right)
\end{align*} \]

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

For every predecessor, expressions can be hoisted to its exit, or
PRE Data Flow Equations

\[ \text{Desirability: } D.2.b \]

\[ \begin{align*}
\ln_n &= P_{av}\ln_n \cap \left( \text{AntGen}_n \cup (\text{Out}_n - \text{Kill}_n) \right) \\
&\quad \bigcap_{p \in \text{pred}(n)} \left( \text{Out}_p \cup \text{AvOut}_p \right) \\
\text{Out}_n &= \begin{cases} 
\text{BI} & \text{n is End block} \\
\bigcap_{s \in \text{succ}(n)} \ln_s & \text{otherwise}
\end{cases}
\end{align*} \]

...expressions are available at the exit of the same predecessor.
PRE Data Flow Equations

\[ In_n = PavIn_n \cap \left( \text{AntGen}_n \cup (Out_n - \text{Kill}_n) \right) \]

\[ \bigcap_{p \in \text{pred}(n)} \left( Out_p \cup \text{AvOut}_p \right) \]

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

Expressions should be hoisted to the exit of a block if they can be hoisted to the entry of all successors.
PRE Data Flow Equations

\[
\begin{align*}
\text{In}_n &= P_{\text{avIn}}_n \cap \left( \text{AntGen}_n \cup (\text{Out}_n - \text{Kill}_n) \right) \\
&\quad \cap \left( \text{Out}_p \cup \text{AvOut}_p \right) \\
&\quad \quad p \in \text{pred}(n) \\
\text{Out}_n &= \begin{cases} 
\text{BL} & n \text{ is End block} \\
\bigcap_{s \in \text{succ}(n)} \text{In}_s & \text{otherwise}
\end{cases}
\end{align*}
\]
Deletion Criteria in PRE

• An expression is redundant in node $n$ if
  ▶ it can be placed at the entry (i.e. can be “hoisted” out) of $n$, AND
  ▶ it is upwards exposed in node $n$.

$$\text{Redundant}_n = \text{In}_n \cap \text{AntGen}_n$$

• A hoisting path for an expression $e$ begins at $n$ if $e \in \text{Redundant}_n$
• This hoisting path extends against the control flow.
Insertion Criteria in PRE

• An expression is inserted at the exit of node $n$ is
  ▶ it can be placed at the exit of $n$, AND
  ▶ it is not available at the exit of $n$, AND
  ▶ it cannot be hoisted out of $n$, OR it is modified in $n$.

\[
\text{Insert}_n = \text{Out}_n \cap (\neg \text{AvOut}_n) \cap (\neg \text{In}_n \cup \text{Kill}_n)
\]

• A hoisting path for an expression $e$ ends at $n$ if $e \in \text{Insert}_n$
Performing PRE by Computing $In/Out$: Simple Cases

1. $c = a \cdot b$
2. $d = a \cdot b$
3. $a = 5$
4. $d = c$

$\Rightarrow$

1. $t = a \cdot b$
2. $c = t$
3. $a = 5$
4. $d = t$

---

1. $c = a \cdot b$
2. $a = b \cdot c$
3. $
\Rightarrow$

1. $t = b \cdot c$
2. $a = t$
3. $

Performing PRE by Computing In/Out: Simple Cases

\[ c = a \times b \]
\[ a = 5 \]
\[ d = a \times b \]

\[ t = a \times b \]
\[ c = t \]
\[ t = a \times b \]

\[ a = 5 \]
\[ d = t \]

\[ a = b \times c \]
\[ t = b \times c \]

\[ a = t \]
Performing PRE by Computing \textit{In/Out}: Simple Cases

\[ c = a \times b \]
\[ d = a \times b \]
\[ a = 5 \]
\[ t = a \times b \]
\[ c = t \]
\[ t = b \times c \]
\[ a = t \]
\[ d = t \]

May 2011 Uday Khedker
Tutorial Problems for PRE

(a)
Tutorial Problems for PRE

(a) $a \times b$

(b) $a = 5$

May 2011 Uday Khedker
Tutorial Problems for PRE

(a) $a \cdot b = 2$
(b) $a = 5$

May 2011

Uday Khedker
Tutorial Problems for PRE

(a) \( a \times b \)

(b) \( a = 5 \)

(c) \( a \times b \)
Tutorial Problems for PRE

(a) \( a \times b \)

(b) \( a = 5 \)

(c) \( a \times b \)

May 2011
**Tutorial Problems for PRE**

(a) 

(b) 

(c) 

(d)
Tutorial Problems for PRE

(a) $a \ast b$

1. $a \ast b$
2. $a = 5$
3. $a \ast b$
4. $a \ast b$

(b) $a = 5$

1. $a = 5$
2. $a \ast b$
3. $a \ast b$
4. $a \ast b$

(c) $a \ast b$

1. $a \ast b$
2. $a \ast b$
3. $a \ast b$
4. $a \ast b$

(d) $a \ast b$

1. $a \ast b$
2. $a \ast b$
3. $a \ast b$
4. $a \ast b$
Tutorial Problems for PRE

(a) \[a \cdot b\]
\[a \cdot b\]
\[a = 5\]
\[a \cdot b\]
\[a = 5\]

(b) \[a \cdot b\]
\[a = 5\]
\[a \cdot b\]
\[a = 5\]

(c) \[a \cdot b\]
\[a = 5\]
\[a \cdot b\]
\[a = 5\]

(d) \[a \cdot b\]
\[a = 5\]
\[a \cdot b\]
\[a = 5\]

(e) \[a \cdot b\]
\[a = 5\]
\[a \cdot b\]
\[a = 5\]
Tutorial Problems for PRE

(a) 

(b) 

(c) 

(d) 

(e)
Tutorial Problems for PRE

Redundancy

Insertion

May 2011 Uday Khedker
Let \( \{a \ast b, b \ast c\} \equiv \text{bit string 11} \)

<table>
<thead>
<tr>
<th>Node ( n )</th>
<th>( \text{Kill}_n )</th>
<th>( \text{AntGen}_n )</th>
<th>( \text{PavIn}_n )</th>
<th>( \text{AvOut}_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>2</td>
<td>00</td>
<td>10</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>00</td>
<td>11</td>
<td>00</td>
</tr>
<tr>
<td>4</td>
<td>00</td>
<td>00</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>00</td>
<td>01</td>
<td>11</td>
<td>01</td>
</tr>
<tr>
<td>6</td>
<td>00</td>
<td>00</td>
<td>11</td>
<td>01</td>
</tr>
</tbody>
</table>

- Compute \( \text{In}_n/\text{Out}_n/\text{Redundant}_n/\text{Insert}_n \)
- Identify hoisting paths
Result of PRE Data Flow Analysis of the Running Example

Bit vector $a \times b \ a + b \ a - b \ a - c \ b + c$

<table>
<thead>
<tr>
<th>Block</th>
<th>Constant information</th>
<th>Global Information</th>
<th>Changes in iteration # 2</th>
<th>Changes in iteration # 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$PavIn_n$</td>
<td>$AvOut_n$</td>
<td>$Out_n$</td>
<td>$In_n$</td>
</tr>
<tr>
<td>$n_8$</td>
<td>11111</td>
<td>00011</td>
<td>00000</td>
<td>00011</td>
</tr>
<tr>
<td>$n_7$</td>
<td>11101</td>
<td>11000</td>
<td>00011</td>
<td>01001</td>
</tr>
<tr>
<td>$n_6$</td>
<td>11101</td>
<td>11001</td>
<td>01001</td>
<td>01001</td>
</tr>
<tr>
<td>$n_5$</td>
<td>11101</td>
<td>11000</td>
<td>01001</td>
<td>01001</td>
</tr>
<tr>
<td>$n_4$</td>
<td>11100</td>
<td>10100</td>
<td>01001</td>
<td>11100</td>
</tr>
<tr>
<td>$n_3$</td>
<td>11101</td>
<td>10000</td>
<td>01000</td>
<td>01001</td>
</tr>
<tr>
<td>$n_2$</td>
<td>10001</td>
<td>00010</td>
<td>00011</td>
<td>00000</td>
</tr>
<tr>
<td>$n_1$</td>
<td>00000</td>
<td>10001</td>
<td>00000</td>
<td>00000</td>
</tr>
</tbody>
</table>

May 2011

Uday Khedker
Hoisting Paths for Some Expressions in the Running Example

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

May 2011

Uday Khedker
Hoisting Paths for Some Expressions in the Running Example

\[ n_1 \]
\[
\begin{align*}
  b &= 4; \\
  a &= b + c; \\
  d &= a \times b;
\end{align*}
\]

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

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

\[ n_4 \]
\[
\begin{align*}
  c &= a \times b; \\
  f(a - b);
\end{align*}
\]

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

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

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

\[ n_8 \]
\[
\begin{align*}
  h(a - c); \\
  f(b + c);
\end{align*}
\]
Hoisting Paths for Some Expressions in the Running Example

\[\begin{align*}
\text{n}_1 & : b = 4; \\
& a = b + c; \\
& d = a \times b;
\end{align*}\]

\[\begin{align*}
\text{n}_2 & : b = a - c;
\end{align*}\]

\[\begin{align*}
\text{n}_3 & : c = b + c;
\end{align*}\]

\[\begin{align*}
\text{n}_4 & : c = a \times b; \\
& f(a - b);
\end{align*}\]

\[\begin{align*}
\text{n}_5 & : d = a + b;
\end{align*}\]

\[\begin{align*}
\text{n}_6 & : f(b + c);
\end{align*}\]

\[\begin{align*}
\text{n}_7 & : g(a + b);
\end{align*}\]

\[\begin{align*}
\text{n}_8 & : h(a - c); \\
& f(b + c);
\end{align*}\]
Hoisting Paths for Some Expressions in the Running Example

\begin{align*}
  n_1: & \quad b = 4; \\
       & \quad a = b + c; \\
       & \quad d = a \times b; \\

  n_2: & \quad b = a - c; \\

  n_3: & \quad c = b + c; \\

  n_4: & \quad c = a \times b; \\
       & \quad f(a - b); \\

  n_5: & \quad d = a + b; \\

  n_6: & \quad f(b + c); \\

  n_7: & \quad g(a + b); \\

  n_8: & \quad h(a - c); \\
       & \quad f(b + c);
\end{align*}
Hoisting Paths for Some Expressions in the Running Example

\begin{align*}
n_1 & : b = 4; \\
 & a = b + c; \\
 & d = a * b;
\end{align*}

\begin{align*}
n_2 & : b = a - c;
\end{align*}

\begin{align*}
n_3 & : c = b + c;
\end{align*}

\begin{align*}
n_4 & : c = a * b; \\
 & f(a - b);
\end{align*}

\begin{align*}
n_5 & : d = a + b;
\end{align*}

\begin{align*}
n_6 & : f(b + c);
\end{align*}

\begin{align*}
n_7 & : g(a + b);
\end{align*}

\begin{align*}
n_8 & : h(a - c); \\
 & f(b + c);
\end{align*}
Hoisting Paths for Some Expressions in the Running Example

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

\[ n_1 \]

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

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

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

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

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

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

\[ n_8 \] \[ h(a - c); \]
\[ f(b + c); \]
Optimized Version of the Running Example

\[ b = 4; \]
\[ t_2 = b + c; \]
\[ a = t_2; \]
\[ t_0 = a \times b; \]
\[ d = t_0; \]

\[ c = t_2 \]
\[ t_1 = a + b; \]

\[ b = c; \]
\[ f(a - c); \]
\[ t_2 = b + c; \]

\[ c = t_0; \]
\[ f(a - b); \]
\[ t_2 = b + c; \]

\[ d = t_1; \]
\[ t_2 = b + c; \]

\[ f(t_2); \]

\[ h(a - c); \]
\[ f(t_2); \]

\[ g(t_1); \]
Part 3

The Need for a More General Setting
What We Have Seen So Far . . .

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Entity</th>
<th>Attribute at $p$</th>
<th>Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>Live variables</td>
<td>Variables</td>
<td>Use</td>
<td>Starting at $p$</td>
</tr>
<tr>
<td>Available expressions</td>
<td>Expressions</td>
<td>Availability</td>
<td>Reaching $p$</td>
</tr>
<tr>
<td>Partially available expressions</td>
<td>Expressions</td>
<td>Availability</td>
<td>Reaching $p$</td>
</tr>
<tr>
<td>Anticipable expressions</td>
<td>Expressions</td>
<td>Use</td>
<td>Starting at $p$</td>
</tr>
<tr>
<td>Reaching definitions</td>
<td>Definitions</td>
<td>Availability</td>
<td>Reaching $p$</td>
</tr>
<tr>
<td>Partial redundancy elimination</td>
<td>Expressions</td>
<td>Profitable hoistability</td>
<td>Involving $p$</td>
</tr>
</tbody>
</table>
An Introduction to Constant Propagation

\[ a = 1 \]
\[ b = 2 \]
\[ c = a + b \]

\[ n_1 \]

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

\[ n_2 \]

\[ d = c - 1 \]
\[ a = 2 \]
\[ b = 1 \]
\[ c = a + b \]

\[ n_3 \]
An Introduction to Constant Propagation

\[ a = 1 \]
\[ b = 2 \]
\[ c = a + b \] 

\[ d = a \times b \] 

\[ d = c - 1 \]
\[ a = 2 \]
\[ b = 1 \]
\[ c = a + b \] 

Execution Sequence
\[ \langle a, b, c, d \rangle \]
\[ \langle ?, ?, ?, ? \rangle \] 

\[ n_1 \]

May 2011
Uday Khedker
An Introduction to Constant Propagation

\[ a = 1 \]
\[ b = 2 \]
\[ c = a + b \]

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

\[ d = c - 1 \]
\[ a = 2 \]
\[ b = 1 \]
\[ c = a + b \]
An Introduction to Constant Propagation

Execution Sequence

\[ \langle a, b, c, d \rangle \]
\[ \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \]
\[ \langle ?, ?, ?, ? \rangle \]
\[ \langle 1, 2, 3, ? \rangle \]
\[ \langle 1, 2, 3, 2 \rangle \]

Node 1

\[ a = 1 \]
\[ b = 2 \]
\[ c = a + b \]

Node 2

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

Node 3

\[ d = c - 1 \]
\[ a = 2 \]
\[ b = 1 \]
\[ c = a + b \]
An Introduction to Constant Propagation

\[ a = 1 \]
\[ b = 2 \]
\[ c = a + b \]

\[ n_1 \]

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

\[ n_2 \]

\[ d = c - 1 \]
\[ a = 2 \]
\[ b = 1 \]
\[ c = a + b \]

\[ n_3 \]

\[ \langle a, b, c, d \rangle \]
\[ \langle ?, ?, ?, ? \rangle \]
\[ \langle 1, 2, 3, ? \rangle \]
\[ \langle 1, 2, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]

Execution Sequence

IN

OUT

May 2011
An Introduction to Constant Propagation

\[ a = 1 \]
\[ b = 2 \]
\[ c = a + b \]

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

\[ d = c - 1 \]
\[ a = 2 \]
\[ b = 1 \]
\[ c = a + b \]

\[ \langle a, b, c, d \rangle \]
\[ \langle ?, ?, ?, ? \rangle \]
\[ \langle 1, 2, 3, ? \rangle \]
\[ \langle 1, 2, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
An Introduction to Constant Propagation

Execution Sequence

\[ \langle a, b, c, d \rangle \]

\[ \langle ?, ?, ?, ? \rangle \]

\[ \langle 1, 2, 3, ? \rangle \]

\[ \langle 1, 2, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ n_1 \]

\[ n_2 \]

\[ n_3 \]

\[ n_1 \]

\[ n_2 \]

\[ n_3 \]

May 2011

Uday Khedker
An Introduction to Constant Propagation

\[
\begin{align*}
    a &= 1 \\
    b &= 2 \\
    c &= a + b \\
    d &= a \times b \\
    d &= c - 1 \\
    a &= 2 \\
    b &= 1 \\
    c &= a + b
\end{align*}
\]

\[
\langle a, b, c, d \rangle \\
\langle ?, ?, ?, ? \rangle \\
\langle 1, 2, 3, ? \rangle \\
\langle 1, 2, 3, 2 \rangle \\
\langle 2, 1, 3, 2 \rangle \\
\langle 2, 1, 3, 2 \rangle \\
\langle 2, 1, 3, 2 \rangle \\
\langle 2, 1, 3, 2 \rangle
\]

May 2011
An Introduction to Constant Propagation

\[
\begin{align*}
    a &= 1 \\
    b &= 2 \\
    c &= a + b \\
\end{align*}
\]

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

\[
\begin{align*}
    d &= c - 1 \\
    a &= 2 \\
    b &= 1 \\
    c &= a + b \\
\end{align*}
\]

\[\langle a, b, c, d \rangle\]

\[
\downarrow \downarrow \downarrow \downarrow \langle ? , ?, ?, ? \rangle
\]

\[
\begin{align*}
    \langle 1, 2, 3, ? \rangle \\
    \langle 2, 1, 3, ? \rangle \\
    \langle 2, 1, 3, ? \rangle \\
    \langle 2, 1, 3, ? \rangle
\end{align*}
\]

\[
\begin{align*}
    n_1 \\
    n_2 \\
    n_3 \\
    n_2 \\
    n_3 \\
    n_3
\end{align*}
\]

\[
\text{Execution Sequence}
\]

\[
\text{IN}
\]

\[
\text{OUT}
\]
An Introduction to Constant Propagation

Summary Values

\[ \langle ?, ?, ?, ?, ? \rangle \]

Execution Sequence

\[ \langle a, b, c, d \rangle \]

\[ \langle ?, ?, ?, ?, ? \rangle \]

\[ \langle 1, 2, 3, ? \rangle \]

\[ \langle 1, 2, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \| \]
An Introduction to Constant Propagation

Summary Values

$n_1$

\[ a = 1 \]
\[ b = 2 \]
\[ c = a + b \]

\[ \langle ?, ?, ?, ? \rangle \]

\[ \langle 1, 2, 3, ? \rangle \]

\[ \langle a, b, c, d \rangle \]

$n_2$

\[ c = a + b \]
\[ d = a * b \]

\[ \langle 1, 2, 3, ? \rangle \]

$n_3$

\[ d = c - 1 \]
\[ a = 2 \]
\[ b = 1 \]
\[ c = a + b \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \ldots \]
An Introduction to Constant Propagation

Summary Values

\[ \langle a, b, c, d \rangle \]
\[ \langle 1, 2, 3, ? \rangle \]
\[ \langle ?, ?, ?, ? \rangle \]

Execution Sequence

\[ n_1 \]
\[ n_2 \]
\[ n_3 \]

\[ \langle 2, 1, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \ldots \]
An Introduction to Constant Propagation

Summary Values

\[ \langle ?, ?, ?, ? \rangle \]
\[ \langle 1, 2, 3, ? \rangle \]
\[ \langle \times, \times, 3, 2 \rangle \]

Execution Sequence

\[ \langle a, b, c, d \rangle \]
\[ \langle ?, ?, ?, ? \rangle \]
\[ \langle 1, 2, 3, ? \rangle \]
\[ \langle 1, 2, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]

May 2011

Uday Khedker
An Introduction to Constant Propagation

Summary Values

\[ \langle ?, ?, ?, ?, ? \rangle \]
\[ \langle 1, 2, 3, ? \rangle \]
\[ \langle \times, \times, 3, 2 \rangle \]
\[ \langle \times, \times, 3, 2 \rangle \]

Execution Sequence

\[ \langle a, b, c, d \rangle \]
\[ \langle ?, ?, ?, ? \rangle \]
\[ \langle 1, 2, 3, ? \rangle \]
\[ \langle 1, 2, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]

\[ \ldots \]
An Introduction to Constant Propagation

Summary Values

\[ \langle a, b, c, d \rangle \]
\[ \langle ?, ?, ?, ? \rangle \]
\[ \langle 1, 2, 3, ?, ? \rangle \]
\[ \langle \times, \times, 3, 2 \rangle \]
\[ \langle \times, \times, 3, 2 \rangle \]
\[ \langle \times, \times, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]

Execution Sequence

\[ \langle a, b, c, d \rangle \]
\[ \langle ?, ?, ?, ? \rangle \]
\[ \langle 1, 2, 3, ? \rangle \]
\[ \langle \times, \times, 3, 2 \rangle \]
\[ \langle \times, \times, 3, 2 \rangle \]
\[ \langle \times, \times, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]

\[ \ldots \]
**An Introduction to Constant Propagation**

### Summary Values

- \( n_1 \)
  - \( a = 1 \)
  - \( b = 2 \)
  - \( c = a + b \)
  - \( \langle ?, ?, ?, ? \rangle \)
  - \( \langle 1, 2, 3, ? \rangle \)

- \( n_2 \)
  - \( c = a + b \)
  - \( d = a \times b \)
  - \( \langle \times, \times, 3, 2 \rangle \)
  - \( \langle \times, \times, 3, 2 \rangle \)

- \( n_3 \)
  - \( d = c - 1 \)
  - \( a = 2 \)
  - \( b = 1 \)
  - \( c = a + b \)
  - \( \langle \times, \times, 3, 2 \rangle \)
  - \( \langle 2, 1, 3, 2 \rangle \)

**Desired Solution**
Data Flow Values for Constant Propagation

- Tuples of the form $\langle \xi_1, \xi_2, \ldots, \xi_k \rangle$ where $\xi_i$ is the data flow value for $i^{th}$ variable.

  Unlike bit vector frameworks, value $\xi_i$ is not 0 or 1 (i.e. true or false). Instead, it is one of the following:
  
  - ? indicating that not much is known about the constantness of variable $v_i$
  - $\times$ indicating that variable $v_i$ does not have a constant value
  - An integer constant $c_1$ if the value of $v_i$ is known to be $c_1$ at compile time

- Alternatively, sets of pairs $\langle v_i, \xi_i \rangle$ for each variable $v_i$. 
Confluence Operation for Constant Propagation

- Confluence operation \( \langle a, c_1 \rangle \sqcap \langle a, c_2 \rangle \)

<table>
<thead>
<tr>
<th>( \sqcap )</th>
<th>( \langle a, ? \rangle )</th>
<th>( \langle a, \times \rangle )</th>
<th>( \langle a, c_1 \rangle )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \langle a, ? \rangle )</td>
<td>( \langle a, ? \rangle )</td>
<td>( \langle a, \times \rangle )</td>
<td>( \langle a, c_1 \rangle )</td>
</tr>
<tr>
<td>( \langle a, \times \rangle )</td>
<td>( \langle a, \times \rangle )</td>
<td>( \langle a, \times \rangle )</td>
<td>( \langle a, \times \rangle )</td>
</tr>
<tr>
<td>( \langle a, c_2 \rangle )</td>
<td>( \langle a, c_2 \rangle )</td>
<td>( \langle a, \times \rangle )</td>
<td>If ( c_1 = c_2 ) ( \langle a, c_1 \rangle ) Otherwise ( \langle a, \times \rangle )</td>
</tr>
</tbody>
</table>

- This is neither \( \cap \) nor \( \cup \).

What are its properties?
Flow Functions for Constant Propagation

- Flow function for \( r = a_1 \times a_2 \)

<table>
<thead>
<tr>
<th>( \text{mult} )</th>
<th>( \langle a_1, ? \rangle )</th>
<th>( \langle a_1, \times \rangle )</th>
<th>( \langle a_1, c_1 \rangle )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \langle a_2, ? \rangle )</td>
<td>( \langle r, ? \rangle )</td>
<td>( \langle r, \times \rangle )</td>
<td>( \langle r, ? \rangle )</td>
</tr>
<tr>
<td>( \langle a_2, \times \rangle )</td>
<td>( \langle r, \times \rangle )</td>
<td>( \langle r, \times \rangle )</td>
<td>( \langle r, \times \rangle )</td>
</tr>
<tr>
<td>( \langle a_2, c_2 \rangle )</td>
<td>( \langle r, ? \rangle )</td>
<td>( \langle r, \times \rangle )</td>
<td>( \langle r, (c_1 \times c_2) \rangle )</td>
</tr>
</tbody>
</table>

- This cannot be expressed in the form

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

where \( \text{Gen}_n \) and \( \text{Kill}_n \) are constant effects of block \( n \).
Round Robin Iterative Analysis for Constant Propagation

\[ a = 1 \]
\[ b = 2 \]
\[ c = a + b \]

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

\[ d = c - 1 \]
\[ a = 2 \]
\[ b = 1 \]
\[ c = a + b \]
Round Robin Iterative Analysis for Constant Propagation

Iteration
#1

\[
\begin{align*}
  n_1 & \quad a = 1 \\
  & \quad b = 2 \\
  & \quad c = a + b \\
  \langle ?, ?, ?, ? \rangle
\\
  n_2 & \quad c = a + b \\
  & \quad d = a \times b \\
  \langle 1, 2, 3, ? \rangle
\\
  n_3 & \quad d = c - 1 \\
  & \quad a = 2 \\
  & \quad b = 1 \\
  & \quad c = a + b \\
  \langle 1, 2, 3, 2 \rangle
\\
  \langle 2, 1, 3, 2 \rangle
\end{align*}
\]
Round Robin Iterative Analysis for Constant Propagation

Iteration #1

\[ a = 1 \]
\[ b = 2 \]
\[ c = a + b \]

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

Iteration #2

\[ a = 2 \]
\[ b = 1 \]
\[ c = a + b \]
Round Robin Iterative Analysis for Constant Propagation

Iteration #1
\[a = 1\]
\[b = 2\]
\[c = a + b\]

Iteration #2
\[c = a + b\]
\[d = a \times b\]

Iteration #3
\[d = c - 1\]
\[a = 2\]
\[b = 1\]
\[c = a + b\]
Round Robin Iterative Analysis for Constant Propagation

Iteration #1
\[ a = 1 \]
\[ b = 2 \]
\[ c = a + b \]
\[ \langle ?, ?, ?, ? \rangle \]

Iteration #2
\[ c = a + b \]
\[ d = a \times b \]
\[ \langle 1, 2, 3, ? \rangle \]
\[ \langle \times, \times, 3, 2 \rangle \]
\[ \langle \times, \times, 3, \times \rangle \]
\[ \langle \times, \times, 3, 2 \rangle \]

Iteration #3
\[ d = c - 1 \]
\[ a = 2 \]
\[ b = 1 \]
\[ c = a + b \]
\[ \langle 1, 2, 3, 2 \rangle \]
\[ \langle \times, \times, \times, \times \rangle \]
\[ \langle \times, \times, \times, \times \rangle \]
\[ \langle \times, \times, 3, 2 \rangle \]

Desired solution
\[ \langle ?, ?, ?, ?, ? \rangle \]
Round Robin Iterative Analysis for Constant Propagation

\[ a = 1 \]
\[ b = 2 \]
\[ c = a + b \]

\[ n_1 \]

Iteration #1

\[ \langle ?, ?, ?, ? \rangle \]

Iteration #2

\[ \langle ?, ?, ?, ? \rangle \]

Iteration #3

\[ \langle ?, ?, ?, ? \rangle \]

Desired solution

\[ \langle ?, ?, ?, ? \rangle \]

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

\[ n_2 \]

Iteration #1

\[ \langle 1, 2, 3, ? \rangle \]

Iteration #2

\[ \langle 1, 2, 3, ? \rangle \]

Iteration #3

\[ \langle 1, 2, 3, ? \rangle \]

\[ \langle 1, 2, 3, ? \rangle \]

\[ d = c - 1 \]
\[ a = 2 \]
\[ b = 1 \]
\[ c = a + b \]

\[ n_3 \]

Iteration #1

\[ \langle 1, 2, 3, 2 \rangle \]

Iteration #2

\[ \langle 1, 2, 3, 2 \rangle \]

Iteration #3

\[ \langle 1, 2, 3, 2 \rangle \]

\[ \langle 1, 2, 3, 2 \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]
\[ \langle 2, 1, 3, \times \rangle \]
\[ \langle 2, 1, 3, \times \rangle \]
\[ \langle 2, 1, 3, \times \rangle \]

\[ \langle 2, 1, 3, 2 \rangle \]
Issues in Data Flow Analysis

- Data Flow Values
- Desired Solutions
- Acceptable Operations
- Practical Algorithms
Issues in Data Flow Analysis

- Representation
- Approximation: Partial Order, Lattices

Data Flow Values
Desired Solutions
Acceptable Operations
Practical Algorithms
Issues in Data Flow Analysis

- Representation
- Approximation: Partial Order, Lattices

- Merge: Commutativity, Associativity, Idempotence
- Flow Functions: Monotonicity, Distributivity, Boundedness, Separability
Issues in Data Flow Analysis

- Representation
- Approximation: Partial Order, Lattices
- Merge: Commutativity, Associativity, Idempotence
- Flow Functions: Monotonicity, Distributivity, Boundedness, Separability
- Existence
- Safety (soundness)
- Precision
Issues in Data Flow Analysis

- Representation
- Approximation: Partial Order, Lattices

Data Flow Values
- Merge: Commutativity, Associativity, Idempotence
- Flow Functions: Monotonicity, Distributivity, Boundedness, Separability

Desired Solutions
- Existence
- Safety (soundness)
- Precision

Acceptable Operations
- Complexity, efficiency
- Convergence
- Initialization

Practical Algorithms
Part 4

Data Flow Values: An Overview
Data Flow Values: An Outline of Our Discussion

- The need to define the notion of abstraction
- Lattices, variants of lattices
- Relevance of lattices for data flow analysis
  - Partial order relation as approximation of data flow values
  - Meet operations as confluence of data flow values
- Cartesian product of lattices
- Example of lattices
Partially Ordered Sets and Lattices

Partially ordered sets

Partial order $\sqsubseteq$ is reflexive, transitive, and antisymmetric
Partially Ordered Sets and Lattices

Partially ordered sets

Partial order $\sqsubseteq$ is reflexive, transitive, and antisymmetric

A lower bound of $x, y$ is $u$ s.t. $u \sqsubseteq x$ and $u \sqsubseteq y$

An upper bound of $x, y$ is $u$ s.t. $x \sqsubseteq u$ and $y \sqsubseteq u$
Partially Ordered Sets and Lattices

- Partially ordered sets
  - Partial order \( \sqsubseteq \) is reflexive, transitive, and antisymmetric

- Lattices
  - Every non-empty finite subset has a greatest lower bound (glb) and a least upper bound (lub)
Partially OrderedSets

Set \{1, 2, 3, 4, 9\} with \sqsubseteq relation as “divides” (i.e. \(a \sqsubseteq b\) iff \(a\) divides \(b\))
Set \( \{1, 2, 3, 4, 9\} \) with \( \sqsubseteq \) relation as “divides” (i.e. \( a \sqsubseteq b \) iff \( a \) divides \( b \))
Partially Ordered Sets

Set \{1, 2, 3, 4, 9\} with \subseteq relation as “divides” (i.e. \(a \subseteq b\) iff \(a\) divides \(b\))

\[
\begin{array}{c}
4 & 9 \\
\downarrow & \downarrow \\
2 & 3 \\
\downarrow & \downarrow \\
1
\end{array}
\]

Subsets \{4, 9\} and \{2, 3\} do not have an upper bound in the set
Set \{1, 2, 3, 4, 9, 36\} with \sqsubseteq relation as “divides” (i.e. \(a \sqsubseteq b\) iff \(a\) divides \(b\))
Complete Lattice

- Lattice: A partially ordered set such that every non-empty finite subset has a glb and a lub.

Example:
Lattice \( \mathbb{Z} \) of integers under \( \leq \) relation. All finite subsets have a glb and a lub. Infinite subsets do not have a glb or a lub.
Complete Lattice

- Lattice: A partially ordered set such that every non-empty finite subset has a glb and a lub.

Example:
Lattice $\mathbb{Z}$ of integers under $\leq$ relation. All finite subsets have a glb and a lub. Infinite subsets do not have a glb or a lub.

- Complete Lattice: A lattice in which even $\emptyset$ and infinite subsets have a glb and a lub.
Complete Lattice

- Lattice: A partially ordered set such that every non-empty finite subset has a glb and a lub.

  Example:
  Lattice $\mathbb{Z}$ of integers under $\leq$ relation. All finite subsets have a glb and a lub. Infinite subsets do not have a glb or a lub.

- Complete Lattice: A lattice in which even $\emptyset$ and infinite subsets have a glb and a lub.

  Example:
  Lattice $\mathbb{Z}$ of integers under $\leq$ relation with $\infty$ and $-\infty$. 
Complete Lattice

• Lattice: A partially ordered set such that every non-empty finite subset has a glb and a lub.

Example:
Lattice $\mathbb{Z}$ of integers under $\leq$ relation. All finite subsets have a glb and a lub. Infinite subsets do not have a glb or a lub.

• Complete Lattice: A lattice in which even $\emptyset$ and infinite subsets have a glb and a lub.

Example:
Lattice $\mathbb{Z}$ of integers under $\leq$ relation with $\infty$ and $-\infty$.

- $\infty$ is the top element denoted $\top$: $\forall i \in \mathbb{Z}, \; i \leq \top$.
- $-\infty$ is the bottom element denoted $\bot$: $\forall i \in \mathbb{Z}, \; \bot \leq i$.
$\mathbb{Z} \cup \{\infty, -\infty\}$ is a Complete Lattice

- Infinite subsets of $\mathbb{Z} \cup \{\infty, -\infty\}$ have a glb and lub.
\( \mathbb{Z} \cup \{\infty, -\infty\} \) is a Complete Lattice

- Infinite subsets of \( \mathbb{Z} \cup \{\infty, -\infty\} \) have a glb and lub.

- What about the empty set?
\[ \mathbb{Z} \cup \{\infty, -\infty\} \text{ is a Complete Lattice} \]

- Infinite subsets of \( \mathbb{Z} \cup \{\infty, -\infty\} \) have a glb and lub.

- What about the empty set?

  \[ \text{glb}(\emptyset) \text{ is } \top \]
$\mathbb{Z} \cup \{\infty, -\infty\}$ is a Complete Lattice

- Infinite subsets of $\mathbb{Z} \cup \{\infty, -\infty\}$ have a glb and lub.

- What about the empty set?
  - $\text{glb}(\emptyset)$ is $\top$

Every element of $\mathbb{Z} \cup \{\infty, -\infty\}$ is vacuously a lower bound of an element in $\emptyset$ (because there is no element in $\emptyset$).
\( \mathbb{Z} \cup \{\infty, -\infty\} \) is a Complete Lattice

- Infinite subsets of \( \mathbb{Z} \cup \{\infty, -\infty\} \) have a glb and lub.

- What about the empty set?

  \( \text{glb}(\emptyset) \) is \( \top \)

  Every element of \( \mathbb{Z} \cup \{\infty, -\infty\} \) is vacuously a lower bound of an element in \( \emptyset \) (because there is no element in \( \emptyset \)). The greatest among these lower bounds is \( \top \).
\[ \mathbb{Z} \cup \{\infty, -\infty\} \text{ is a Complete Lattice} \]

- Infinite subsets of \( \mathbb{Z} \cup \{\infty, -\infty\} \) have a glb and lub.

- What about the empty set?

  - \( \text{glb}(\emptyset) \) is \( \top \)

    Every element of \( \mathbb{Z} \cup \{\infty, -\infty\} \) is vacuously a lower bound of an element in \( \emptyset \) (because there is no element in \( \emptyset \)). The greatest among these lower bounds is \( \top \).

  - \( \text{lub}(\emptyset) \) is \( \bot \)
Finite Lattices are Complete

- Any given set of elements has a glb and a lub

Available Expressions Analysis

\[
\begin{align*}
\top & \quad \{e_1, e_2, e_3\} \\
\{e_1, e_2\} & \quad \{e_1, e_3\} \quad \{e_2, e_3\} \\
\{e_1\} & \quad \{e_2\} \quad \{e_3\} \\
\emptyset & \quad \emptyset \\
\bot &
\end{align*}
\]

Partially Available Expressions Analysis

\[
\begin{align*}
\top & \quad \emptyset \\
\{e_1\} & \quad \{e_2\} \quad \{e_3\} \\
\{e_1, e_2\} & \quad \{e_1, e_3\} \quad \{e_2, e_3\} \\
\{e_1, e_2, e_3\} & \quad \bot
\end{align*}
\]
Lattice for May-Must Analysis

- There is no $\top$ among the natural values

- Interpreting data flow values
  - $\text{No}$. Information does not hold along any path
  - $\text{Must}$. Information must hold along all paths
  - $\text{May}$. Information may hold along some path

- An artificial $\top$ can be added
  However, a lub may not exist for arbitrary sets
Some Variants of Lattices

A poset $L$ is

- A **lattice** iff each non-empty finite subset of $L$ has a glb and lub.
- A **complete lattice** iff each subset of $L$ has a glb and lub.
- A **meet semilattice** iff each non-empty finite subset of $L$ has a glb.
- A **join semilattice** iff each non-empty finite subset of $L$ has a lub.
Ascending and Descending Chains

- Strictly ascending chain. $x \sqsubset y \sqsubset \cdots \sqsubset z$
- Strictly descending chain. $x \sqsupset y \sqsupset \cdots \sqsupset z$
- **DCC**: Descending Chain Condition
  All strictly descending chains are finite.
- **ACC**: Ascending Chain Condition
  All strictly ascending chains are finite.
• If $L$ satisfies acc and dcc, then
  ▶ $L$ has finite height, and
  ▶ $L$ is complete.

• A complete lattice need not have finite height (i.e. strict chains may not be finite).
Example:
Lattice of integers under $\leq$ relation with $\infty$ as $\top$ and $-\infty$ as $\bot$. 

May 2011
Uday Khedker
Operations on Lattices

- Meet (\(\sqcap\)) and Join (\(\sqcup\))

\[
\begin{array}{c}
36 \\
\downarrow \\
4 \\
\downarrow \\
2 \\
\downarrow \\
1 \\
\end{array}
\]

\[
\begin{array}{c}
36 \\
\downarrow \\
9 \\
\downarrow \\
3 \\
\end{array}
\]
Operations on Lattices

- Meet ($\sqcap$) and Join ($\sqcup$)
  - $x \sqcap y$ computes the glb of $x$ and $y$.
  - $z = x \sqcap y \Rightarrow z \subseteq x \land z \subseteq y$

```
36

4  9

2  3

1
```
Operations on Lattices

- **Meet (\(\sqcap\)) and Join (\(\sqcup\))**
  - \(x \sqcap y\) computes the glb of \(x\) and \(y\).
    
    \[z = x \sqcap y \Rightarrow z \sqsubseteq x \land z \sqsubseteq y\]

  - \(x \sqcup y\) computes the lub of \(x\) and \(y\).
    
    \[z = x \sqcup y \Rightarrow z \sqsupseteq x \land z \sqsupseteq y\]
Operations on Lattices

- **Meet (⊓)** and **Join (⊔)**
  - \( x ⊓ y \) computes the glb of \( x \) and \( y \).
    \[
    z = x ⊓ y \Rightarrow z \subseteq x \land z \subseteq y
    \]
  - \( x ⊔ y \) computes the lub of \( x \) and \( y \).
    \[
    z = x ⊔ y \Rightarrow z \supseteq x \land z \supseteq y
    \]
  - ⊓ and ⊔ are commutative, associative, and idempotent.
Operations on Lattices

• Meet ($\sqcap$) and Join ($\sqcup$)
  - $x \sqcap y$ computes the glb of $x$ and $y$.
    $z = x \sqcap y \Rightarrow z \subseteq x \land z \subseteq y$
  - $x \sqcup y$ computes the lub of $x$ and $y$.
    $z = x \sqcup y \Rightarrow z \supseteq x \land z \supseteq y$
  - $\sqcap$ and $\sqcup$ are commutative, associative, and idempotent.

• Top ($\top$) and Bottom ($\bot$) elements

\[
\forall x \in L, \; x \sqcap \top = x \\
\forall x \in L, \; x \sqcup \top = \top \\
\forall x \in L, \; x \sqcap \bot = \bot \\
\forall x \in L, \; x \sqcup \bot = x
\]
Operations on Lattices

- **Meet (\(\sqcap\)) and Join (\(\sqcup\))**
  - \(x \sqcap y\) computes the glb of \(x\) and \(y\).
    
    \[ z = x \sqcap y \Rightarrow z \sqsubseteq x \land z \sqsubseteq y \]
  
  - \(x \sqcup y\) computes the lub of \(x\) and \(y\).
    
    \[ z = x \sqcup y \Rightarrow z \sqsupseteq x \land z \sqsupseteq y \]
  
  - \(\sqcap\) and \(\sqcup\) are commutative, associative, and idempotent.

- **Top (\(\top\)) and Bottom (\(\bot\)) elements**

\[
\begin{align*}
\forall x \in L, \ x \sqcap \top &= x \\
\forall x \in L, \ x \sqcup \top &= \top \\
\forall x \in L, \ x \sqcap \bot &= \bot \\
\forall x \in L, \ x \sqcup \bot &= x
\end{align*}
\]

- **Greatest common divisor (or highest common factor)** in the lattice

\[ x \sqcap y = \text{gcd}'(x, y) \]
Operations on Lattices

- **Meet (\(\sqcap\)) and Join (\(\sqcup\))**
  - \(x \sqcap y\) computes the glb of \(x\) and \(y\).
    
    \[
    z = x \sqcap y \Rightarrow z \subseteq x \land z \subseteq y
    \]
  
  - \(x \sqcup y\) computes the lub of \(x\) and \(y\).
    
    \[
    z = x \sqcup y \Rightarrow z \supseteq x \land z \supseteq y
    \]
  
  - \(\sqcap\) and \(\sqcup\) are commutative, associative, and idempotent.

- **Top (\(\top\)) and Bottom (\(\bot\)) elements**
  
  \[
  \forall x \in L, \quad x \sqcap \top = x
  \]
  
  \[
  \forall x \in L, \quad x \sqcup \top = \top
  \]
  
  \[
  \forall x \in L, \quad x \sqcap \bot = \bot
  \]
  
  \[
  \forall x \in L, \quad x \sqcup \bot = x
  \]

- **Greatest common divisor (or highest common factor) in the lattice**
  
  \[
  x \sqcap y = gcd'(x, y)
  \]

- **Lowest common multiple in the lattice**
  
  \[
  x \sqcup y = lcm'(x, y)
  \]
Cartesian Product of Lattice

\[
\langle L_N, \sqsubseteq_N, \cap_N, \cup_N \rangle \times \langle L_A, \sqsubseteq_A, \cap_A, \cup_A \rangle = \langle L_{\lambda}, \sqsubseteq_{\lambda}, \cap_{\lambda}, \cup_{\lambda} \rangle
\]
Cartesian Product of Lattice

\[
\langle L_N, \sqsubseteq_N, \sqcap_N, \sqcup_N \rangle \times \langle L_A, \sqsubseteq_A, \sqcap_A, \sqcup_A \rangle = \langle 1, a \rangle \langle 2, a \rangle \langle 3, a \rangle \langle 4, a \rangle
\]
Cartesian Product of Lattice

\[ \langle L_N, \sqsubseteq_N, \sqcap_N, \sqcup_N \rangle \times \langle L_A, \sqsubseteq_A, \sqcap_A, \sqcup_A \rangle = \langle 1, a \rangle \times \langle 2, a \rangle \times \langle 3, a \rangle \times \langle 4, a \rangle \times \langle 1, b \rangle \times \langle 2, b \rangle \times \langle 3, b \rangle \times \langle 4, b \rangle \]
Cartesian Product of Lattice

\[
\langle L_N, \sqsubseteq_N, \sqcap_N, \sqcup_N \rangle \times \langle L_A, \sqsubseteq_A, \sqcap_A, \sqcup_A \rangle = \langle L, \sqsubseteq, \sqcap, \sqcup \rangle
\]
Cartesian Product of Lattice

\[ \langle L_N, \sqsubseteq_N, \land_N, \lor_N \rangle \times \langle L_A, \sqsubseteq_A, \land_A, \lor_A \rangle = \langle 1, a \rangle \langle 2, a \rangle \langle 3, a \rangle \langle 4, a \rangle \langle 1, b \rangle \langle 2, b \rangle \langle 3, b \rangle \langle 4, b \rangle \]
Cartesian Product of Lattice

\[ \langle \mathcal{L}, \sqsubseteq_{\mathcal{N}}, \sqcap_{\mathcal{N}}, \sqcup_{\mathcal{N}} \rangle \times \langle \mathcal{L}, \sqsubseteq_{\mathcal{A}}, \sqcap_{\mathcal{A}}, \sqcup_{\mathcal{A}} \rangle = \langle \mathcal{L}, \sqsubseteq_{\mathcal{A}}, \sqcap_{\mathcal{A}}, \sqcup_{\mathcal{A}} \rangle \]
Cartesian Product of Lattice

\[ \langle L_N, \sqsubseteq_N, \sqcap_N, \sqcup_N \rangle \times \langle L_A, \sqsubseteq_A, \sqcap_A, \sqcup_A \rangle = \]

\[ \langle 1, a \rangle \]
\[ \langle 2, a \rangle \]
\[ \langle 3, a \rangle \]
\[ \langle 1, b \rangle \]
\[ \langle 2, b \rangle \]
\[ \langle 3, b \rangle \]
\[ \langle 4, b \rangle \]
Cartesian Product of Lattice

\[ \langle L_N, \sqsubseteq_N, \sqcap_N, \sqcup_N \rangle \times \langle L_A, \sqsubseteq_A, \sqcap_A, \sqcup_A \rangle \]

\[ \times \quad a \quad b \quad \Rightarrow \quad \langle 1, a \rangle \quad \langle 2, a \rangle \quad \langle 1, b \rangle \quad \langle 3, a \rangle \quad \langle 2, b \rangle \quad \langle 4, a \rangle \quad \langle 3, b \rangle \quad \langle 4, b \rangle \]

\[ \langle L_C, \sqsubseteq_C, \sqcap_C, \sqcup_C \rangle \]
Cartesian Product of Lattice

\[ \langle L, \sqsubseteq, \sqcap, \sqcup \rangle \times \langle L, \sqsubseteq, \sqcap, \sqcup \rangle = \langle L, \sqsubseteq, \sqcap, \sqcup \rangle \]

\[ \langle 1, a \rangle \]

\[ \langle 2, a \rangle \]

\[ \langle 1, b \rangle \]

\[ \langle 3, a \rangle \]

\[ \langle 2, b \rangle \]

\[ \langle 4, a \rangle \]

\[ \langle 3, b \rangle \]

\[ \langle 4, b \rangle \]

\[ \langle L_C, \sqsubseteq, \sqcap, \sqcup \rangle \]

\[ \langle x_1, y_1 \rangle \sqsubseteq_C \langle x_2, y_2 \rangle \iff x_1 \sqsubseteq_N x_2 \land y_1 \sqsubseteq_A y_2 \]

\[ \langle x_1, y_1 \rangle \sqcap_C \langle x_2, y_2 \rangle = \langle x_1 \sqcap_N x_2, y_1 \sqcap_A y_2 \rangle \]

\[ \langle x_1, y_1 \rangle \sqcup_C \langle x_2, y_2 \rangle = \langle x_1 \sqcup_N x_2, y_1 \sqcup_A y_2 \rangle \]
The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition
The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- $\text{glb}$ must exist for all non-empty finite subsets
The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- glb must exist for all non-empty finite subsets
- \( \bot \) must exist

What guarantees the presence of \( \bot \)?
The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- \text{glb} must exist for all non-empty finite subsets
- \bot must exist
  
  What guarantees the presence of \bot?

- \top may not exist. Can be added artificially.
The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- glb must exist for all non-empty finite subsets
- ⊥ must exist

What guarantees the presence of ⊥?

- Assume that two maximal descending chains terminate at two incomparable elements $x_1$ and $x_2$

- ⊤ may not exist. Can be added artificially.
Meet semilattices satisfying the descending chain condition

- \text{glb} must exist for all non-empty finite subsets
- \bot must exist

What guarantees the presence of \bot?

- Assume that two maximal descending chains terminate at two incomparable elements \(x_1\) and \(x_2\)
- Since this is a meet semilattice, \text{glb} of \(\{x_1, x_2\}\) must exist (say z).

- \top may not exist. Can be added artificially.
Meet semilattices satisfying the descending chain condition

- \( \text{glb} \) must exist for all non-empty finite subsets
- \( \bot \) must exist

What guarantees the presence of \( \bot \)?

- Assume that two maximal descending chains terminate at two incomparable elements \( x_1 \) and \( x_2 \)
- Since this is a meet semilattice, \( \text{glb} \) of \( \{x_1, x_2\} \) must exist (say \( z \)).
  \[ \Rightarrow \] Neither of the chains is maximal.
  Both of them can be extended to include \( z \).

- \( \top \) may not exist. Can be added artificially.
Meet semilattices satisfying the descending chain condition

- glb must exist for all non-empty finite subsets
- \( \perp \) must exist

What guarantees the presence of \( \perp \)?

- Assume that two maximal descending chains terminate at two incomparable elements \( x_1 \) and \( x_2 \)
- Since this is a meet semilattice, glb of \( \{x_1, x_2\} \) must exist (say \( z \)).
  \( \Rightarrow \) Neither of the chains is maximal.
  Both of them can be extended to include \( z \).
- Extending this argument to all strictly descending chains, it is easy to see that \( \perp \) must exist.

- \( \top \) may not exist. Can be added artificially.
The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- glb must exist for all non-empty finite subsets
- ⊥ must exist

What guarantees the presence of ⊥?

- Assume that two maximal descending chains terminate at two incomparable elements $x_1$ and $x_2$
  - Since this is a meet semilattice, glb of $\{x_1, x_2\}$ must exist (say $z$).
    ⇒ Neither of the chains is maximal.
    Both of them can be extended to include $z$.
- Extending this argument to all strictly descending chains, it is easy to see that ⊥ must exist.

- ⊤ may not exist. Can be added artificially.
  - lub of arbitrary elements may not exist
The Set of Data Flow Values For Available Expressions Analysis

- The powerset of the universal set of expressions
- Partial order is the subset relation

Set View of the Lattice
The Set of Data Flow Values For Available Expressions Analysis

- The powerset of the universal set of expressions
- Partial order is the subset relation

\[
\{e_1, e_2, e_3\} \\
\{e_1, e_2\} \quad \{e_1, e_3\} \quad \{e_2, e_3\} \\
\{e_1\} \quad \{e_2\} \quad \{e_3\} \\
\emptyset
\]

Set View of the Lattice

May 2011 Uday Khedker
The Set of Data Flow Values For Available Expressions
Analysis

- The powerset of the universal set of expressions
- Partial order is the subset relation

Set View of the Lattice

Bit Vector View
The Concept of Approximation

- $x$ approximates $y$ iff $x$ can be used in place of $y$ without causing any problems.

- Validity of approximation is context specific
  - $x$ may be approximated by $y$ in one context and by $z$ in another

  ▶ Earnings: Rs. 1050 can be safely approximated by Rs. 1000.
  ▶ Expenses: Rs. 1050 can be safely approximated by Rs. 1100.
Two Important Objectives in Data Flow Analysis

- The discovered data flow information should be
  - **Exhaustive.** No optimization opportunity should be missed.
  - **Safe.** Optimizations which do not preserve semantics should not be enabled.
Two Important Objectives in Data Flow Analysis

• The discovered data flow information should be
  ▶  *Exhaustive*. No optimization opportunity should be missed.
  ▶  *Safe*. Optimizations which do not preserve semantics should not be enabled.

• Conservative approximations of these objectives are allowed
Two Important Objectives in Data Flow Analysis

- The discovered data flow information should be
  - *Exhaustive*. No optimization opportunity should be missed.
  - *Safe*. Optimizations which do not preserve semantics should not be enabled.
- Conservative approximations of these objectives are allowed
- The intended use of data flow information (≡ context) determines validity of approximations
Context Determines the Validity of Approximations

May prohibit correct optimization  May enable wrong optimization

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Application</th>
<th>Safe Approximation</th>
<th>Exhaustive Approximation</th>
</tr>
</thead>
</table>

May 2011  Uday Khedker
Context Determines the Validity of Approximations

May prohibit correct optimization

- Analysis
- Live variables

May enable wrong optimization

- Application
- Dead code elimination

<table>
<thead>
<tr>
<th></th>
<th>Safe Approximation</th>
<th>Exhaustive Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Live variables</td>
<td>A dead variable is considered live</td>
<td>A live variable is considered dead</td>
</tr>
<tr>
<td>Dead code elimination</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Context Determines the Validity of Approximations

May prohibit correct optimization

May enable wrong optimization

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Application</th>
<th>Safe Approximation</th>
<th>Exhaustive Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Live variables</td>
<td>Dead code elimination</td>
<td>A dead variable is considered live</td>
<td>A live variable is considered dead</td>
</tr>
<tr>
<td>Available expressions</td>
<td>Common subexpression elimination</td>
<td>An available expression is considered non-available</td>
<td>A non-available expression is considered available</td>
</tr>
</tbody>
</table>
### Context Determines the Validity of Approximations

- **May prohibit correct optimization**
- **May enable wrong optimization**

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Application</th>
<th>Safe Approximation</th>
<th>Exhaustive Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Live variables</td>
<td>Dead code elimination</td>
<td>A dead variable is considered live</td>
<td>A live variable is considered dead</td>
</tr>
<tr>
<td>Available expressions</td>
<td>Common subexpression elimination</td>
<td>An available expression is considered non-available</td>
<td>A non-available expression is considered available</td>
</tr>
</tbody>
</table>

**Spurious Inclusion**

**Spurious Exclusion**
Partial Order Captures Approximation

- $\sqsubseteq$ captures valid approximations for safety

$x \sqsubseteq y \Rightarrow x$ is weaker than $y$

- The data flow information represented by $x$ can be safely used in place of the data flow information represented by $y$
- It may be imprecise, though.
Partial Order Captures Approximation

- $\sqsubseteq$ captures valid approximations for safety

  $x \sqsubseteq y \Rightarrow x$ is weaker than $y$
  
  - The data flow information represented by $x$ can be safely used in place of the data flow information represented by $y$
  - It may be imprecise, though.

- $\sqsupseteq$ captures valid approximations for exhaustiveness

  $x \sqsupseteq y \Rightarrow x$ is stronger than $y$

  - The data flow information represented by $x$ contains every value contained in the data flow information represented by $y$
  - It may be unsafe, though.
Partial Order Captures Approximation

- \( \sqsubseteq \) captures valid approximations for safety
  \[ x \sqsubseteq y \Rightarrow x \text{ is weaker than } y \]
  - The data flow information represented by \( x \) can be safely used in place of the data flow information represented by \( y \)
  - It may be imprecise, though.

- \( \sqsupseteq \) captures valid approximations for exhaustiveness
  \[ x \sqsupseteq y \Rightarrow x \text{ is stronger than } y \]
  - The data flow information represented by \( x \) contains every value contained in the data flow information represented by \( y \)
  - It may be unsafe, though.

*We want most exhaustive information which is also safe.*
Most Approximate Values in a Complete Lattice

- **Top.** $\forall x \in L, \ x \sqsubseteq \top$. The most exhaustive value.

- **Bottom.** $\forall x \in L, \ \bot \sqsubseteq x$. The safest value.
Most Approximate Values in a Complete Lattice

- **Top.** $\forall x \in L, \ x \sqsubseteq \top$. The most exhaustive value.
  - Using $\top$ in place of any data flow value will never miss out (or rule out) any possible value.

- **Bottom.** $\forall x \in L, \bot \sqsubseteq x$. The safest value.
Most Approximate Values in a Complete Lattice

- **Top.** $\forall x \in L, \ x \sqsubseteq \top$. The most exhaustive value.
  - Using $\top$ in place of any data flow value will never miss out (or rule out) any possible value.
  - The consequences may be semantically *unsafe*, or *incorrect*.

- **Bottom.** $\forall x \in L, \ \bot \sqsubseteq x$. The safest value.
Most Approximate Values in a Complete Lattice

- **Top.** \( \forall x \in L, x \sqsubseteq \top \). The most exhaustive value.
  - Using \( \top \) in place of any data flow value will never miss out (or rule out) any possible value.
  - The consequences may be semantically *unsafe*, or *incorrect*.

- **Bottom.** \( \forall x \in L, \bot \sqsubseteq x \). The safest value.
  - Using \( \bot \) in place of any data flow value will never be *unsafe*, or *incorrect*. 
Most Approximate Values in a Complete Lattice

- **Top.** \( \forall x \in L, \ x \sqsubseteq \top \). The most exhaustive value.
  - Using \( \top \) in place of any data flow value will never miss out (or rule out) any possible value.
  - The consequences may be semantically **unsafe**, or **incorrect**.

- **Bottom.** \( \forall x \in L, \ \bot \sqsubseteq x \). The safest value.
  - Using \( \bot \) in place of any data flow value will never be **unsafe**, or **incorrect**.
  - The consequences may be **undefined** or **useless** because this replacement might miss out valid values.
Most Approximate Values in a Complete Lattice

- **Top.** $\forall x \in L, \ x \sqsubseteq \top$. The most exhaustive value.
  - Using $\top$ in place of any data flow value will never miss out (or rule out) any possible value.
  - The consequences may be semantically *unsafe*, or *incorrect*.

- **Bottom.** $\forall x \in L, \ \bot \sqsubseteq x$. The safest value.
  - Using $\bot$ in place of any data flow value will never be *unsafe*, or *incorrect*.
  - The consequences may be *undefined* or *useless* because this replacement might miss out valid values.

*Appropriate orientation chosen by design.*
**Setting Up Lattices**

### Available Expressions Analysis

- $\{e_1, e_2, e_3\}$
- $\{e_1, e_2\}$
- $\{e_1, e_3\}$
- $\{e_2, e_3\}$
- $\{e_1\}$
- $\{e_2\}$
- $\{e_3\}$
- $\emptyset$

### Live Variables Analysis

- $\{v_1\}$
- $\{v_2\}$
- $\{v_3\}$
- $\{v_1, v_2\}$
- $\{v_1, v_3\}$
- $\{v_2, v_3\}$
- $\{v_1, v_2, v_3\}$

\[\sqsubseteq \text{ is } \subseteq\]

\[\sqcap \text{ is } \cap\]

\[\sqcup \text{ is } \cup\]
Partial Order Relation

Reflexive \[ x \sqsubseteq x \]

Transitive \[ x \sqsubseteq y, y \sqsubseteq z \]
\[ \Rightarrow x \sqsubseteq z \]

Antisymmetric \[ x \sqsubseteq y, y \sqsubseteq x \]
\[ \Leftrightarrow x = y \]
### Partial Order Relation

<table>
<thead>
<tr>
<th>Property</th>
<th>Condition</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reflexive</td>
<td>$x \sqsubseteq x$</td>
<td>$x$ can be safely used in place of $x$</td>
</tr>
<tr>
<td>Transitive</td>
<td>$x \sqsubseteq y, y \sqsubseteq z$</td>
<td>If $x$ can be safely used in place of $y$ and $y$ can be safely used in place of $z$, then $x$ can be safely used in place of $z$</td>
</tr>
<tr>
<td>Antisymmetric</td>
<td>$x \sqsubseteq y, y \sqsubseteq x$</td>
<td>If $x$ can be safely used in place of $y$ and $y$ can be safely used in place of $x$, then $x$ must be same as $y$</td>
</tr>
</tbody>
</table>
Merging Information

- $x \sqcap y$ computes the greatest lower bound of $x$ and $y$ i.e. largest $z$ such that $z \sqsubseteq x$ and $z \sqsubseteq y$

The largest safe approximation of combining data flow information $x$ and $y$
Merging Information

- $x \sqcap y$ computes the *greatest lower bound* of $x$ and $y$ i.e. largest $z$ such that $z \sqsubseteq x$ and $z \sqsubseteq y$

  The largest safe approximation of combining data flow information $x$ and $y$

- **Commutative**  \[ x \sqcap y = y \sqcap x \]

- **Associative**  \[ x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z \]

- **Idempotent**  \[ x \sqcap x = x \]
Merging Information

- $x \sqcap y$ computes the *greatest lower bound* of $x$ and $y$ i.e. largest $z$ such that $z \sqsubseteq x$ and $z \sqsubseteq y$

  The largest safe approximation of combining data flow information $x$ and $y$

- **Commutative**  
  $x \sqcap y = y \sqcap x$  
  The order in which the data flow information is merged, does not matter

- **Associative**  
  $x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z$  
  Allow n-ary merging without any restriction on the order

- **Idempotent**  
  $x \sqcap x = x$  
  No loss of information if $x$ is merged with itself
Merging Information

• $x \sqcap y$ computes the *greatest lower bound* of $x$ and $y$ i.e. largest $z$ such that $z \sqsubseteq x$ and $z \sqsubseteq y$

  The largest safe approximation of combining data flow information $x$ and $y$

• Commutative  $x \sqcap y = y \sqcap x$  The order in which the data flow information is merged, does not matter

  Associative  $x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z$  Allow n-ary merging without any restriction on the order

  Idempotent  $x \sqcap x = x$  No loss of information if $x$ is merged with itself

• $\top$ is the identity of $\sqcap$

  ▶ Presence of loops $\Rightarrow$ self dependence of data flow information
  ▶ Using $\top$ as the initial value ensure exhaustiveness
More on Lattices in Data Flow Analysis

$L = \text{Lattice for all expressions}$

$L̂ = \text{Lattice for a single expression}$

(Expression $e$ is available)

1 or \{e\}

0 or $\emptyset$

(Expression $e$ is not available)
More on Lattices in Data Flow Analysis

\( L = \) Lattice for all expressions  \( \hat{L} = \) Lattice for a single expression

- Cartesian products if sets are used, vectors (or tuples) if bit are used.

- \( L = \hat{L} \times \hat{L} \times \hat{L} \) and \( x = \langle \hat{x}_1, \hat{x}_2, \hat{x}_3 \rangle \in L \) where \( \hat{x}_i \in \hat{L} \)

- \( \sqsubseteq = \hat{\sqsubseteq} \times \hat{\sqsubseteq} \times \hat{\sqsubseteq} \) and \( \sqcap = \hat{\sqcap} \times \hat{\sqcap} \times \hat{\sqcap} \)

- \( \top = \hat{\top} \times \hat{\top} \times \hat{\top} \) and \( \bot = \hat{\bot} \times \hat{\bot} \times \hat{\bot} \)

(Expression e is available)

- 1 or \{e\}

(Expression e is not available)

- 0 or \(\emptyset\)
Component Lattice for Data Flow Information Represented By Bit Vectors

\[ (\hat{\top}) \]
\[ 1 \]
\[ \hat{\perp} \]
\[ 0 \]

\[ (\hat{\top}) \]
\[ 0 \]
\[ \hat{\perp} \]
\[ 1 \]

\[ \hat{\perp} \text{ is } \cap \text{ or Boolean AND} \]
\[ \hat{\top} \text{ is } \cup \text{ or Boolean OR} \]
Component Lattice for Integer Constant Propagation

\[
\begin{array}{c}
\text{undef or ud} \\
\text{nonconst or nc} \\
\text{(⊥)}
\end{array}
\]

\[
\begin{array}{ccccccc}
-\infty & \cdots & -2 & -1 & 0 & 1 & 2 & \cdots & \infty
\end{array}
\]

- Overall lattice \( L \) is the product of \( \hat{L} \) for all variables.
- \( \sqcap \) and \( \hat{\sqcap} \) get defined by \( \sqsubseteq \) and \( \hat{\sqsubseteq} \).

\[
\begin{array}{|c|c|c|}
\hline
\hat{\sqcap} & \langle a, ud \rangle & \langle a, nc \rangle & \langle a, c_1 \rangle \\
\langle a, ud \rangle & \langle a, ud \rangle & \langle a, nc \rangle & \langle a, c_1 \rangle \\
\langle a, nc \rangle & \langle a, nc \rangle & \langle a, nc \rangle & \langle a, nc \rangle \\
\langle a, c_2 \rangle & \langle a, c_2 \rangle & \langle a, nc \rangle & \text{If } c_1 = c_2 \text{ then } \langle a, c_1 \rangle \text{ else } \langle a, nc \rangle \\
\hline
\end{array}
\]
Component Lattice for May Points-To Analysis

- Relation between pointer variables and locations in the memory.
Component Lattice for May Points-To Analysis

- Relation between pointer variables and locations in the memory.
- Assuming three locations $l_1$, $l_2$, and $l_3$, the component lattice for pointer $p$ is.

```
(\top)

\emptyset

\{p \mapsto l_1\} \quad \{p \mapsto l_2\} \quad \{p \mapsto l_3\}

\{p \mapsto l_1, p \mapsto l_2\} \quad \{p \mapsto l_1, p \mapsto l_3\} \quad \{p \mapsto l_2, p \mapsto l_3\}

\{p \mapsto l_1, p \mapsto l_2, p \mapsto l_2\}

(\bot)
```
Component Lattice for May Points-To Analysis

- Relation between pointer variables and locations in the memory.
- Assuming three locations $l_1$, $l_2$, and $l_3$, the component lattice for pointer $p$ is.

\[
\begin{align*}
&\left(\hat{\top}\right) \\
&\emptyset \\
&\{ p \mapsto l_1 \}, \{ p \mapsto l_2 \}, \{ p \mapsto l_3 \} \\
&\{ p \mapsto l_1, p \mapsto l_2 \}, \{ p \mapsto l_1, p \mapsto l_3 \}, \{ p \mapsto l_2, p \mapsto l_3 \} \\
&\{ p \mapsto l_1, p \mapsto l_2, p \mapsto l_2 \} \\
&\left(\hat{\bot}\right)
\end{align*}
\]

Alternatively,

\[
\begin{align*}
&\left(\hat{\top}\right) \\
&\emptyset \\
&\{ l_1 \}, \{ l_2 \}, \{ l_3 \} \\
&\{ l_1, l_2 \}, \{ l_1, l_3 \}, \{ l_2, l_3 \} \\
&\{ l_1, l_2, l_2 \} \\
&\left(\hat{\bot}\right)
\end{align*}
\]
A pointer can point to at most one location.

Component Lattice for Must Points-To Analysis

\[
\begin{array}{ccc}
(\top) & \text{undef} & (\top) \\
\downarrow & & \downarrow \\
p \rightarrow l_1 & p \rightarrow l_2 & l_1 \\
\downarrow & & \downarrow \\
none & p \rightarrow l_3 & l_2 \\
\downarrow & & \downarrow \\
(\bot) & none & (\bot) \\
\end{array}
\]

Alternatively,

\[
\begin{array}{ccc}
(\top) & \text{undef} & (\top) \\
\downarrow & & \downarrow \\
l_1 & l_2 & l_3 \\
\downarrow & & \downarrow \\
none & & none \\
\downarrow & & \downarrow \\
(\bot) & & (\bot) \\
\end{array}
\]
General Lattice for May-Must Analysis

Interpreting data flow values
- \textit{Unknown}. Nothing is known as yet
- \textit{No}. Information does not hold along any path
- \textit{Must}. Information must hold along all paths
- \textit{May}. Information may hold along some path

Possible Applications
- Pointer Analysis: No need of separate of \textit{May} and \textit{Must} analyses eg. \((p \mapsto l, \text{May})\), \((p \mapsto l, \text{Must})\), \((p \mapsto l, \text{No})\), or \((p \mapsto l, \text{Unknown})\).
- Type Inferencing for Dynamically Checked Languages