Further 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 (previous lecture)
- Introduction to Constant Propagation (previous lecture)
- Theoretical Abstractions in Data Flow Analysis
  - The world of data flow values (previous lecture)
  - The world of functions and operations that compute data values (today)
  - Results of data flow analysis (today)
  - Algorithms for performing data flow analysis (today)
- Precise Modelling of General flows (today)
  Example: Constant Propagation
Part 2

Flow Functions
Flow Functions: An Outline of Our Discussion

- Defining flow functions
- Properties of flow functions
  (Some properties discussed in the context of solutions of data flow analysis)
The Set of Flow Functions

- $F$ is the set of functions $f : L \mapsto L$ such that
  - $F$ contains an identity function
    To model “empty” statements, i.e. statements which do not influence the data flow information
  - $F$ is closed under composition
    Cumulative effect of statements should generate data flow information from the same set.
  - For every $x \in L$, there must be a finite set of flow functions $\{f_1, f_2, \ldots, f_m\} \subseteq F$ such that
    \[ x = \bigcap_{1 \leq i \leq m} f_i(BI) \]

- Properties of $f$
  - Monotonicity and Distributivity
  - Separability
Flow Functions in Bit Vector Data Flow Frameworks

- Bit Vector Frameworks: Available Expressions Analysis, Reaching Definitions Analysis, Live variable Analysis, Anticipable Expressions Analysis, Partial Redundancy Elimination etc.
  - All functions can be defined in terms of constant $Gen$ and $Kill$
    \[
    f(x) = Gen \cup (x - Kill)
    \]
  - Lattices are powersets with partial orders as $\subseteq$ or $\supseteq$ relations
  - Information is merged using $\cap$ or $\cup$
Flow Functions in Bit Vector Data Flow Frameworks

- Bit Vector Frameworks: Available Expressions Analysis, Reaching Definitions Analysis Live variable Analysis, Anticipable Expressions Analysis, Partial Redundancy Elimination etc.
  - All functions can be defined in terms of constant $Gen$ and $Kill$
    
    \[ f(x) = Gen \cup (x - Kill) \]

  - Lattices are powersets with partial orders as $\subseteq$ or $\supseteq$ relations
  - Information is merged using $\cap$ or $\cup$

- Flow functions in Faint Variables Analysis, Pointer Analyses, Constant Propagation, Possibly Uninitialized Variables cannot be expressed using constant $Gen$ and $Kill$.

Local context alone is not sufficient to describe the effect of statements fully.
Monotonicity of Flow Functions

- Partial order is preserved: If $x$ can be safely used in place of $y$ then $f(x)$ can be safely used in place of $f(y)$
Monotonicity of Flow Functions

- Partial order is preserved: If $x$ can be safely used in place of $y$ then $f(x)$ can be safely used in place of $f(y)$

\[ x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y) \]
Monotonicity of Flow Functions

- Partial order is preserved: If $x$ can be safely used in place of $y$ then $f(x)$ can be safely used in place of $f(y)$

\[
\forall x, y \in L, x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)
\]
Monotonicity of Flow Functions

- Partial order is preserved: If \( x \) can be safely used in place of \( y \) then \( f(x) \) can be safely used in place of \( f(y) \)

\[
\forall x, y \in L, x \sqsubseteq y \implies f(x) \sqsubseteq f(y)
\]

- Alternative definition

\[
\forall x, y \in L, f(x \sqcap y) \sqsubseteq f(x) \sqcap f(y)
\]
Monotonicity of Flow Functions

- Partial order is preserved: If $x$ can be safely used in place of $y$ then $f(x)$ can be safely used in place of $f(y)$

\[ \forall x, y \in L, x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y) \]

- Alternative definition

\[ \forall x, y \in L, f(x \sqcap y) \sqsubseteq f(x) \sqcap f(y) \]

- Merging at intermediate points in shared segments of paths is safe (However, it may lead to imprecision).
Distributivity of Flow Functions

- Merging distributes over function application

\[ f(x) \sqcap f(y) \]

\[ x \quad y \]
Distributivity of Flow Functions

- Merging distributes over function application

\[ f(x \sqcap y) \]
Distributivity of Flow Functions

- Merging distributes over function application

\[ \forall x, y \in L, x \sqsubseteq y \Rightarrow f(x \sqcap y) = f(x) \sqcap f(y) \]
Distributivity of Flow Functions

- Merging distributes over function application

\[ \forall x, y \in L, x \sqsubseteq y \Rightarrow f(x \sqcap y) = f(x) \sqcap f(y) \]

- Merging at intermediate points in shared segments of paths does not lead to imprecision.
Monotonicity and Distributivity
Monotonicity and Distributivity
Monotonicity and Distributivity

\[ \top \rightarrow L \rightarrow \bot \]

\[ \bot \rightarrow L \rightarrow \top \]

\[ L \rightarrow L \]
Monotonicity and Distributivity
Monotonicity and Distributivity
Monotonicity and Distributivity

Monotonic and Distributive

$L$ $L$
Monotonicity and Distributivity

Monotonic but not Distributive
Distributivity of Bit Vector Frameworks

\[ f(x) = \text{Gen} \cup (x - \text{Kill}) \]
\[ f(y) = \text{Gen} \cup (y - \text{Kill}) \]

\[ f(x \cup y) = \text{Gen} \cup ((x \cup y) - \text{Kill}) \]
\[ = \text{Gen} \cup ((x - \text{Kill}) \cup (y - \text{Kill})) \]
\[ = (\text{Gen} \cup (x - \text{Kill}) \cup \text{Gen} \cup (y - \text{Kill})) \]
\[ = f(x) \cup f(y) \]

\[ f(x \cap y) = \text{Gen} \cup ((x \cap y) - \text{Kill}) \]
\[ = \text{Gen} \cup ((x - \text{Kill}) \cap (y - \text{Kill})) \]
\[ = (\text{Gen} \cup (x - \text{Kill}) \cap \text{Gen} \cup (y - \text{Kill})) \]
\[ = f(x) \cap f(y) \]
Non-Distributivity of Constant Propagation

\[
\begin{align*}
\text{n}_1 & : a = 1 \\
& \quad b = 2 \\
& \quad c = a + b \\
\text{n}_2 & : c = a + b \\
& \quad d = a \times b \\
\text{n}_3 & : d = c - 1 \\
& \quad a = 2 \\
& \quad b = 1 \\
& \quad c = a + b
\end{align*}
\]
Non-Distributivity of Constant Propagation

- $x = \langle 1, 2, 3, ? \rangle$ (Along $Out_{n_1} \rightarrow In_{n_2}$)

Diagram:

- $n_1$
  - $a = 1$
  - $b = 2$
  - $c = a + b$
  - $a = 1, b = 2$

- $n_2$
  - $c = a + b$
  - $d = a \times b$

- $n_3$
  - $d = c - 1$
  - $a = 2$
  - $b = 1$
  - $c = a + b$
Non-Distributivity of Constant Propagation

- $x = \langle 1, 2, 3, ? \rangle$ (Along $Out_{n_1} \rightarrow In_{n_2}$)
- $y = \langle 2, 1, 3, 2 \rangle$ (Along $Out_{n_3} \rightarrow In_{n_2}$)
Non-Distributivity of Constant Propagation

- $x = \langle 1, 2, 3, ? \rangle$ (Along $Out_{n_1} \rightarrow In_{n_2}$)
- $y = \langle 2, 1, 3, 2 \rangle$ (Along $Out_{n_3} \rightarrow In_{n_2}$)
- Function application for block $n_2$ before merging

\[
\begin{align*}
f(x) \sqcap f(y) & = f(\langle 1, 2, 3, ? \rangle) \sqcap f(\langle 2, 1, 3, 2 \rangle) \\ & = \langle 1, 2, 3, 2 \rangle \sqcap \langle 2, 1, 3, 2 \rangle \\ & = \langle \bot, \bot, 3, 2 \rangle
\end{align*}
\]
Non-Distributivity of Constant Propagation

- \( x = \langle 1, 2, 3, ? \rangle \) (Along \( Out_{n_1} \rightarrow In_{n_2} \))
- \( y = \langle 2, 1, 3, 2 \rangle \) (Along \( Out_{n_3} \rightarrow In_{n_2} \))
- Function application for block \( n_2 \) before merging
  \[
  f(x) \cap f(y) = f(\langle 1, 2, 3, ? \rangle) \cap f(\langle 2, 1, 3, 2 \rangle) \\
  = \langle 1, 2, 3, 2 \rangle \cap \langle 2, 1, 3, 2 \rangle \\
  = \langle \bot, \bot, 3, 2 \rangle
  \]
- Function application for block \( n_2 \) after merging
  \[
  f(x \cap y) = f(\langle 1, 2, 3, ? \rangle \cap \langle 2, 1, 3, 2 \rangle) \\
  = f(\langle \bot, \bot, 3, 2 \rangle) \\
  = \langle \bot, \bot, \bot, \bot \rangle
  \]
Non-Distributivity of Constant Propagation

- $x = \langle 1, 2, 3, ? \rangle$ (Along $Out_{n_1} \rightarrow In_{n_2}$)
- $y = \langle 2, 1, 3, 2 \rangle$ (Along $Out_{n_3} \rightarrow In_{n_2}$)
- Function application for block $n_2$ before merging
  \[
  f(x) \sqcap f(y) = f(\langle 1, 2, 3, ? \rangle) \sqcap f(\langle 2, 1, 3, 2 \rangle)
  = \langle 1, 2, 3, 2 \rangle \sqcap \langle 2, 1, 3, 2 \rangle
  = \langle \bot, \bot, 3, 2 \rangle
  \]
- Function application for block $n_2$ after merging
  \[
  f(x \sqcap y) = f(\langle 1, 2, 3, ? \rangle \sqcap \langle 2, 1, 3, 2 \rangle)
  = f(\langle \bot, \bot, 3, 2 \rangle)
  = \langle \bot, \bot, \bot, \bot \rangle
  \]
- $f(x \sqcap y) \sqsubseteq f(x) \sqcap f(y)$
Why is Constant Propagation Non-Distribitive?

\[
c = a + b
\]

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

\[
\begin{align*}
a &= 2 \\
b &= 1
\end{align*}
\]
Why is Constant Propagation Non-Distributive?

Possible combinations due to merging

\[
a = 1 \quad a = 2 \quad b = 1 \quad b = 2
\]

\[
c = a + b
\]
Why is Constant Propagation Non-Distributive?

Possible combinations due to merging

\[ c = a + b = 3 \]

- Correct combination.
Why is Constant Propagation Non-Distributive?

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

Possible combinations due to merging
\[ a = 1 \quad a = 2 \quad b = 1 \quad b = 2 \]
\[ c = a + b = 3 \]

• Correct combination.
Why is Constant Propagation Non-Distributive?

Possible combinations due to merging

- Wrong combination.
- Mutually exclusive information.
- No execution path along which this information holds.
Why is Constant Propagation Non-Distributive?

Possible combinations due to merging

\[ a = 1 \quad a = 2 \quad b = 1 \quad b = 2 \]
\[ c = a + b = 4 \]

- Wrong combination.
- Mutually exclusive information.
- No execution path along which this information holds.
Part 3

Solutions of Data Flow Analysis
Solutions of Data Flow Analysis: An Outline of Our Discussion

- MoP and MFP assignments and their relationship
- Existence of MoP assignment
  - Boundedness of flow functions
- Existence and Computability of MFP assignment
  - Flow functions Vs. function computed by data flow equations
- Safety of MFP solution
Solutions of Data Flow Analysis

• An assignment $A$ associates data flow values with program points. $A \subseteq B$ if for all program points $p$, $A(p) \subseteq B(p)$

• Performing data flow analysis

Given

- A set of flow functions, a lattice, and merge operation
- A program flow graph with a mapping from nodes to flow functions

Find out

- An assignment $A$ which is as exhaustive as possible and is safe
Meet Over Paths (MoP) Assignment

- The largest safe approximation of the information reaching a program point along all information flow paths.

\[ \text{MoP}(p) = \bigcap_{\rho \in \text{Paths}(p)} f_\rho(BI) \]

- \( f_\rho \) represents the compositions of flow functions along \( \rho \).
- \( BI \) refers to the relevant information from the calling context.
- All execution paths are considered potentially executable by ignoring the results of conditionals.
Meet Over Paths (MoP) Assignment

- The largest safe approximation of the information reaching a program point along all information flow paths.

\[
MoP(p) = \bigcap_{\rho \in \text{Paths}(p)} f_{\rho}(BI)
\]

- \( f_{\rho} \) represents the compositions of flow functions along \( \rho \).
- \( BI \) refers to the relevant information from the calling context.
- All execution paths are considered potentially executable by ignoring the results of conditionals.

- Any \( \text{Info}(p) \sqsubseteq MoP(p) \) is safe.
Maximum Fixed Point (MFP) Assignment

- Difficulties in computing MoP assignment
Maximum Fixed Point (MFP) Assignment

- Difficulties in computing MoP assignment
  - In the presence of cycles there are infinite paths
    If all paths need to be traversed $\Rightarrow$ Undecidability
Difficulties in computing MoP assignment

- In the presence of cycles there are infinite paths
  If all paths need to be traversed $\Rightarrow$ Undecidability

- Even if a program is acyclic, every conditional multiplies the number of paths by two
  If all paths need to be traversed $\Rightarrow$ Intractability
Maximum Fixed Point (MFP) Assignment

- Difficulties in computing MoP assignment
  - In the presence of cycles there are infinite paths
    - If all paths need to be traversed \(\Rightarrow\) Undecidability
  - Even if a program is acyclic, every conditional multiplies the number of paths by two
    - If all paths need to be traversed \(\Rightarrow\) Intractability
- Why not merge information at intermediate points?
  - Merging is safe but may lead to imprecision.
  - Computes fixed point solutions of data flow equations.
Maximum Fixed Point (MFP) Assignment

- Difficulties in computing MoP assignment
  - In the presence of cycles there are infinite paths
    - If all paths need to be traversed ⇒ Undecidability
  - Even if a program is acyclic, every conditional multiplies the number of paths by two
    - If all paths need to be traversed ⇒ Intractability

- Why not merge information at intermediate points?
  - Merging is safe but may lead to imprecision.
  - Computes fixed point solutions of data flow equations.
Assignments for Constant Propagation Example

\begin{align*}
\text{n}_1 & : & a &= 1 \\
& & b &= 2 \\
& & c &= a + b \\
\end{align*}

\begin{align*}
\text{n}_2 & : & c &= a + b \\
& & d &= a \times b \\
\end{align*}

\begin{align*}
\text{n}_3 & : & d &= c - 1 \\
& & a &= 2 \\
& & b &= 1 \\
& & c &= a + b \\
\end{align*}
Assignments for Constant Propagation Example

\[ n_1 \]
\[ a = 1 \]
\[ b = 2 \]
\[ c = a + b \]

\[ n_2 \]
\[ c = a + b \]
\[ d = a \times b \]

\[ n_3 \]
\[ d = c - 1 \]
\[ a = 2 \]
\[ b = 1 \]
\[ c = a + b \]

MoP
\[ \langle \top, \top, \top, \top \rangle \]
\[ \langle 1, 2, 3, \top \rangle \]
\[ \langle \bot, \bot, 3, 2 \rangle \]
\[ \langle \bot, \bot, 3, 2 \rangle \]
\[ \langle \bot, \bot, 3, 2 \rangle \]
\[ \langle 2, 1, 3, 2 \rangle \]
Assignments for Constant Propagation Example

\[
\begin{align*}
  n_1 & \quad a = 1 \\
  & \quad b = 2 \\
  & \quad c = a + b \\
\end{align*}
\]

\[
\begin{align*}
  n_2 & \quad c = a + b \\
  & \quad d = a \times b \\
\end{align*}
\]

\[
\begin{align*}
  n_3 & \quad d = c - 1 \\
  & \quad a = 2 \\
  & \quad b = 1 \\
  & \quad c = a + b \\
\end{align*}
\]

MoP
\[
\begin{align*}
  \langle \top, \top, \top, \top \rangle & \quad \langle 1, 2, 3, \top \rangle \\
  \langle \bot, \bot, 3, 2 \rangle & \quad \langle \bot, \bot, 3, \bot \rangle \\
  \langle \bot, \bot, 3, 2 \rangle & \quad \langle \bot, \bot, 3, \bot \rangle \\
  \langle \bot, \bot, 3, 2 \rangle & \quad \langle \bot, \bot, 3, \bot \rangle \\
  \langle 2, 1, 3, 2 \rangle & \quad \langle 2, 1, 3, \top \rangle \\
\end{align*}
\]

MFP
\[
\begin{align*}
  \langle \top, \top, \top, \top \rangle & \quad \langle 1, 2, 3, \top \rangle \\
  \langle \bot, \bot, 3, 2 \rangle & \quad \langle \bot, \bot, 3, \bot \rangle \\
  \langle \bot, \bot, 3, 2 \rangle & \quad \langle \bot, \bot, 3, \bot \rangle \\
  \langle \bot, \bot, 3, 2 \rangle & \quad \langle \bot, \bot, 3, \bot \rangle \\
  \langle 2, 1, 3, 2 \rangle & \quad \langle 2, 1, 3, \bot \rangle \\
\end{align*}
\]
Possible Assignments as Solutions of Data Flow Analyses

All possible assignments
Possible Assignments as Solutions of Data Flow Analyses

- All possible assignments
- All safe assignments
Possible Assignments as Solutions of Data Flow Analyses

- All possible assignments
- All safe assignments
- All fixed point solutions
Possible Assignments as Solutions of Data Flow Analyses

- All possible assignments
- All safe assignments
- All fixed point solutions

∀i, In_i = Out_i = T
Possible Assignments as Solutions of Data Flow Analyses

- All possible assignments: $\forall i, In_i = Out_i = \top$
- All safe assignments
- All fixed point solutions: $\forall i, In_i = Out_i = \bot$

May 2011
Possible Assignments as Solutions of Data Flow Analyses

- All possible assignments
- All safe assignments
- All fixed point solutions

∀ᵢ, Inᵢ = Outᵢ = ⊤

∀ᵢ, Inᵢ = Outᵢ = ⊥
Possible Assignments as Solutions of Data Flow Analyses

All possible assignments

\[ \forall i, \text{In}_i = \text{Out}_i = \top \]

Meet Over Paths Assignment

All safe assignments

Maximum Fixed Point

All fixed point solutions

\[ \forall i, \text{In}_i = \text{Out}_i = \bot \]
Possible Assignments as Solutions of Data Flow Analyses

- All possible assignments:
  \( \forall i, In_i = Out_i = \top \)

- All safe assignments:
  \( \forall i, In_i = Out_i = \bot \)

- Meet Over Paths Assignment
- Maximum Fixed Point
- Least Fixed Point

May 2011  Uday Khedker
Available Expr. Analysis Framework with Two Expressions

<table>
<thead>
<tr>
<th>Lattice</th>
<th>Constant Functions</th>
<th>Dependent Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>{a<em>b, b</em>c}</td>
<td>{a<em>b, b</em>c}</td>
<td>f (x)</td>
</tr>
<tr>
<td>{a<em>b} {b</em>c}</td>
<td>{a*b}</td>
<td>f (x)</td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
<td>f (x)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(f)</th>
<th>(f(x))</th>
<th>(f)</th>
<th>(f(x))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f_\top)</td>
<td>{a<em>b, b</em>c}</td>
<td>(f_{id})</td>
<td>(x)</td>
</tr>
<tr>
<td>(f_\bot)</td>
<td>(\emptyset)</td>
<td>(f_c)</td>
<td>(x \cup {a*b})</td>
</tr>
<tr>
<td>(f_a)</td>
<td>{a*b}</td>
<td>(f_d)</td>
<td>(x \cup {b*c})</td>
</tr>
<tr>
<td>(f_b)</td>
<td>{b*c}</td>
<td>(f_e)</td>
<td>(x - {a*b})</td>
</tr>
<tr>
<td>(f_f)</td>
<td>(x - {b*c})</td>
<td>(f_f)</td>
<td>(x - {b*c})</td>
</tr>
</tbody>
</table>
Available Expr. Analysis Framework with Two Expressions

Lattice

\{a \ast b, b \ast c\}

\{a \ast b\} \quad \{b \ast c\}

\emptyset

<table>
<thead>
<tr>
<th>Constant Functions</th>
<th>Dependent Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f )</td>
<td>( f(x) )</td>
</tr>
<tr>
<td>( f_{\top} )</td>
<td>( {a \ast b, b \ast c} )</td>
</tr>
<tr>
<td>( f_{\bot} )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( f_a )</td>
<td>( {a \ast b} )</td>
</tr>
<tr>
<td>( f_b )</td>
<td>( {b \ast c} )</td>
</tr>
<tr>
<td>( f_c )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( f_{f_{\bot}} )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>

Program

1

\[
\begin{bmatrix}
\ast b \\
b \ast c
\end{bmatrix}
\]

2
Available Expr. Analysis Framework with Two Expressions

Lattice

\{a*b, b*c\}  \
/ \  \
{a*b}       \{b*c\}  \
/ \  \
∅  

Constant Functions | Dependent Functions
---|---
\(f\) | \(f(x)\) | \(f\) | \(f(x)\)
\(f_\top\) | \{a*b, b*c\} | \(f_{id}\) | \(x\)
\(f_\bot\) | \(∅\) | \(f_c\) | \(x \cup \{a*b\}\)
\(f_a\) | \{a*b\} | \(f_d\) | \(x \cup \{b*c\}\)
\(f_b\) | \{b*c\} | \(f_e\) | \(x - \{a*b\}\)
\(f_f\) | \(x - \{b*c\}\)

Program

Flow Functions

<table>
<thead>
<tr>
<th>Node</th>
<th>Flow Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(f_\top)</td>
</tr>
<tr>
<td>2</td>
<td>(f_{id})</td>
</tr>
</tbody>
</table>

May 2011
Uday Khedker
Available Expr. Analysis Framework with Two Expressions

Lattice

\{a\ast b, b\ast c\}

\{a\ast b\}
\{b\ast c\}
\emptyset

Program

1
\[
\begin{array}{c}
a\ast b \\
b\ast c
\end{array}
\]

2

Flow Functions

<table>
<thead>
<tr>
<th>Node</th>
<th>Flow Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(f_\top)</td>
</tr>
<tr>
<td>2</td>
<td>(f_{id})</td>
</tr>
</tbody>
</table>

Constant Functions | Dependent Functions

<table>
<thead>
<tr>
<th>(f)</th>
<th>(f(x))</th>
<th>(f)</th>
<th>(f(x))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f_{\top})</td>
<td>{a\ast b, b\ast c}</td>
<td>(f_{id})</td>
<td>(x)</td>
</tr>
<tr>
<td>(f_{\perp})</td>
<td>(\emptyset)</td>
<td>(f_c)</td>
<td>(x \cup {a\ast b})</td>
</tr>
<tr>
<td>(f_a)</td>
<td>{a\ast b}</td>
<td>(f_d)</td>
<td>(x \cup {b\ast c})</td>
</tr>
<tr>
<td>(f_b)</td>
<td>{b\ast c}</td>
<td>(f_e)</td>
<td>(x - {a\ast b})</td>
</tr>
<tr>
<td>(f_f)</td>
<td>(\emptyset)</td>
<td>(f_f)</td>
<td>(x - {b\ast c})</td>
</tr>
</tbody>
</table>

Some Possible Assignments

<table>
<thead>
<tr>
<th></th>
<th>(A_1)</th>
<th>(A_2)</th>
<th>(A_3)</th>
<th>(A_4)</th>
<th>(A_5)</th>
<th>(A_6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(ln_1)</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>(out_1)</td>
<td>11</td>
<td>00</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>(ln_2)</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>(out_2)</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>10</td>
</tr>
</tbody>
</table>

May 2011 Uday Khedker
Available Expr. Analysis Framework with Two Expressions

Lattice

\{a\times b, b\times c\} → \{a\times b\} → \{b\times c\} → \emptyset

<table>
<thead>
<tr>
<th>Constant Functions</th>
<th>Dependent Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f)</td>
<td>(f(x))</td>
</tr>
<tr>
<td>(f\uparrow)</td>
<td>({a\times b, b\times c})</td>
</tr>
<tr>
<td>(f_{id})</td>
<td>(x)</td>
</tr>
</tbody>
</table>

- Maximum fixed point assignment
- Initialization for round robin iterative method: 11

Program

```
1
a\times b
b\times c
```

Flow Functions

<table>
<thead>
<tr>
<th>Node</th>
<th>Flow Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(f\uparrow)</td>
</tr>
<tr>
<td>2</td>
<td>(f_{id})</td>
</tr>
</tbody>
</table>

Some Possible Assignments

<table>
<thead>
<tr>
<th></th>
<th>(A_1)</th>
<th>(A_2)</th>
<th>(A_3)</th>
<th>(A_4)</th>
<th>(A_5)</th>
<th>(A_6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(In_1)</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>(Out_1)</td>
<td>11</td>
<td>00</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>(In_2)</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>(Out_2)</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>10</td>
</tr>
</tbody>
</table>
Available Expr. Analysis Framework with Two Expressions

**Lattice**

\[ \{a\ast b, b\ast c\} \]
\[ \{a\ast b\} \quad \{b\ast c\} \]
\[ \emptyset \]

**Constant Functions**

<table>
<thead>
<tr>
<th></th>
<th>(f)</th>
<th>(f(x))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f_T)</td>
<td>({a\ast b, b\ast c})</td>
<td>(f_{id})</td>
</tr>
<tr>
<td>(f_\perp)</td>
<td>(\emptyset)</td>
<td>(f_c)</td>
</tr>
<tr>
<td>(f_\top)</td>
<td>({a\ast b, b\ast c})</td>
<td>(f_d)</td>
</tr>
<tr>
<td>(f_{id})</td>
<td>({a\ast b})</td>
<td>(f_e)</td>
</tr>
<tr>
<td>(f_f)</td>
<td>(\emptyset)</td>
<td>(f_f)</td>
</tr>
</tbody>
</table>

**Not a fixed point assignment**

**Program**

**Node**

<table>
<thead>
<tr>
<th>Flow Functions</th>
<th>Flow Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(f_T)</td>
</tr>
<tr>
<td>2</td>
<td>(f_{id})</td>
</tr>
</tbody>
</table>

**Flow Functions**

**Some Possible Assignments**

<table>
<thead>
<tr>
<th></th>
<th>(A_1)</th>
<th>(A_2)</th>
<th>(A_3)</th>
<th>(A_4)</th>
<th>(A_5)</th>
<th>(A_6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(In_1)</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>(Out_1)</td>
<td>11</td>
<td>00</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>(In_2)</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>(Out_2)</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>10</td>
</tr>
</tbody>
</table>
Available Expr. Analysis Framework with Two Expressions

Lattice

\{a \ast b, b \ast c\}
\{a \ast b\} \quad \{b \ast c\}
\emptyset

Constant Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Flow Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f)</td>
<td>(f(x))</td>
</tr>
<tr>
<td>(f_{\top})</td>
<td>{a \ast b, b \ast c}</td>
</tr>
<tr>
<td>(f_{id})</td>
<td>(x)</td>
</tr>
</tbody>
</table>

Dependent Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Flow Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f)</td>
<td>(f(x))</td>
</tr>
<tr>
<td>(f_{\top})</td>
<td>{a \ast b, b \ast c}</td>
</tr>
<tr>
<td>(f_{id})</td>
<td>(x)</td>
</tr>
</tbody>
</table>

- Minimum fixed point assignment
- Initialization for round robin iterative method: 00

Program

Flow Functions

<table>
<thead>
<tr>
<th>Node</th>
<th>Flow Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(f_{\top})</td>
</tr>
<tr>
<td>2</td>
<td>(f_{id})</td>
</tr>
</tbody>
</table>

Some Possible Assignments

<table>
<thead>
<tr>
<th></th>
<th>(A_1)</th>
<th>(A_2)</th>
<th>(A_3)</th>
<th>(A_4)</th>
<th>(A_5)</th>
<th>(A_6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(In_1)</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>(Out_1)</td>
<td>11</td>
<td>00</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>(In_2)</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>(Out_2)</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>10</td>
</tr>
</tbody>
</table>

May 2011  Uday Khedker
Available Expr. Analysis Framework with Two Expressions

**Lattice**

\{a\ast b, b\ast c\}  
\{a\ast b\}  
\{b\ast c\}  
\emptyset

<table>
<thead>
<tr>
<th>Constant Functions</th>
<th>Dependent Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f)</td>
<td>(f(x))</td>
</tr>
<tr>
<td>(f_{\top})</td>
<td>({a\ast b, b\ast c})</td>
</tr>
<tr>
<td>(f_{id})</td>
<td>(x)</td>
</tr>
</tbody>
</table>

- Fixed point assignment which is neither maximum nor minimum
- Initialization for round robin iterative method: 10

**Program**

<table>
<thead>
<tr>
<th>Flow Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

**Some Possible Assignments**

<table>
<thead>
<tr>
<th></th>
<th>(A_1)</th>
<th>(A_2)</th>
<th>(A_3)</th>
<th>(A_4)</th>
<th>(A_5)</th>
<th>(A_6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(In_1)</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>(Out_1)</td>
<td>11</td>
<td>00</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>(In_2)</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>(Out_2)</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>10</td>
</tr>
</tbody>
</table>
Available Expr. Analysis Framework with Two Expressions

<table>
<thead>
<tr>
<th>Lattice</th>
<th>Constant Functions</th>
<th>Dependent Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>{a&amp;b, b&amp;c}</td>
<td>(f)</td>
<td>(f(x))</td>
</tr>
<tr>
<td>{a&amp;b}</td>
<td>(f)</td>
<td>(f(x))</td>
</tr>
<tr>
<td>{b&amp;c}</td>
<td>(f_{\bot})</td>
<td>({a&amp;b, b&amp;c})</td>
</tr>
<tr>
<td>(</td>
<td>)</td>
<td>(f_{id})</td>
</tr>
</tbody>
</table>

- Fixed point assignment which is neither maximum nor minimum
- Initialization for round robin iterative method: 01

Program

<table>
<thead>
<tr>
<th>Flow Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Some Possible Assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A_1)</td>
</tr>
<tr>
<td>(In_1)</td>
</tr>
<tr>
<td>(Out_1)</td>
</tr>
<tr>
<td>(In_2)</td>
</tr>
<tr>
<td>(Out_2)</td>
</tr>
</tbody>
</table>
Available Expr. Analysis Framework with Two Expressions

**Lattice**

\[ \{ a \ast b, b \ast c \} \]

\[ \{ a \ast b \} \quad \{ b \ast c \} \]

\[ \emptyset \]

**Constant Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Flow Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f )</td>
<td>( f(x) )</td>
</tr>
<tr>
<td>( f_\top )</td>
<td>( { a \ast b, b \ast c } )</td>
</tr>
<tr>
<td>( f_\bot )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( f_{id} )</td>
<td>( { a \ast b } )</td>
</tr>
</tbody>
</table>

**Dependent Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Flow Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f )</td>
<td>( f(x) )</td>
</tr>
<tr>
<td>( f_{id} )</td>
<td>( x )</td>
</tr>
<tr>
<td>( f_c )</td>
<td>( x \cup { a \ast b } )</td>
</tr>
<tr>
<td>( f_d )</td>
<td>( x \cup { b \ast c } )</td>
</tr>
<tr>
<td>( f_e )</td>
<td>( x - { a \ast b } )</td>
</tr>
<tr>
<td>( f_f )</td>
<td>( x - { b \ast c } )</td>
</tr>
</tbody>
</table>

- Not a fixed point assignment

**Program**

1. \( a \ast b \)
2. \( b \ast c \)

**Flow Functions**

<table>
<thead>
<tr>
<th>Node</th>
<th>Flow Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( f_\top )</td>
</tr>
<tr>
<td>2</td>
<td>( f_{id} )</td>
</tr>
</tbody>
</table>

**Some Possible Assignments**

<table>
<thead>
<tr>
<th>Assignment</th>
<th>( A_1 )</th>
<th>( A_2 )</th>
<th>( A_3 )</th>
<th>( A_4 )</th>
<th>( A_5 )</th>
<th>( A_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( In_1 )</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>( Out_1 )</td>
<td>11</td>
<td>00</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>( In_2 )</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>( Out_2 )</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>10</td>
</tr>
</tbody>
</table>
Part 4

Performing Data Flow Analysis
Performing Data Flow Analysis

- Algorithms for computing MFP solution
- Complexity of data flow analysis
- Factor affecting the complexity of data flow analysis
Iterative Methods of Performing Data Flow Analysis

Successive recomputation after conservative initialization (⊤)

- **Round Robin.** Repeated traversals over nodes in a fixed order
  
  **Termination:** After values stabilise
  
  + Simplest to understand and implement
  
  - May perform unnecessary computations
Iterative Methods of Performing Data Flow Analysis

Successive recomputation after conservative initialization ($\top$)

- **Round Robin.** Repeated traversals over nodes in a fixed order

  Termination: After values stabilise
  
  + Simplest to understand and implement
  - May perform unnecessary computations

Our examples use this method.
Iterative Methods of Performing Data Flow Analysis

Successive recomputation after conservative initialization ($\top$)

- **Round Robin.** Repeated traversals over nodes in a fixed order
  
  Termination: After values stabilise
  
  + Simplest to understand and implement
  
  - May perform unnecessary computations

- **Work List.** Dynamic list of nodes which need recomputation
  
  Termination: When the list becomes empty
  
  + Demand driven. Avoid unnecessary computations.
  
  - Overheads of maintaining work list.

Our examples use this method.
Elimination Methods of Performing Data Flow Analysis

Delayed computations of dependent data flow values of dependent nodes.

Find suitable single-entry regions.

- *Interval Based Analysis*. Uses graph partitioning.
- *T₁, T₂ Based Analysis*. Uses graph parsing.
Classification of Edges in a Graph

Graph $G$

![Graph Image]
Classification of Edges in a Graph

Graph $G$

A depth first spanning tree of $G$
Classification of Edges in a Graph

Graph $G$

A depth first spanning tree of $G$

- Back edges
- Forward edges
- Tree edges
- Cross edges
Classification of Edges in a Graph

Graph \( G \)

A depth first spanning tree of \( G \)

For data flow analysis, we club \textit{tree}, \textit{forward}, and \textit{cross} edges into \textit{forward} edges. Thus we have just forward or back edges in a control flow graph.
Reverse Post Order Traversal

- A reverse post order (rpo) is a topological sort of the graph obtained after removing back edges.

Graph $G$ and $G'$ obtained after removing back edges of $G$.

- Some possible RPOs for $G$ are: $(1, 2, 3, 4, 5, 6, 7, 8)$, $(1, 6, 7, 2, 3, 4, 5, 8)$, $(1, 6, 2, 7, 4, 3, 5, 8)$, and $(1, 2, 6, 7, 3, 4, 5, 8)$.
Round Robin Iterative Algorithm

1. $ln_0 = Bl$
2. for all $j \neq 0$ do
3.     $ln_j = \top$
4. change = true
5. while change do
6.     { change = false
7.         for $j = 1$ to $N - 1$ do
8.             { temp = $\prod_{p \in \text{pred}(j)} f_p(ln_p)$
9.                 if temp $\neq ln_j$ then
10.                    { ln_j = temp
11.                       change = true
12.                 }
13.             }
14.     }
Round Robin Iterative Algorithm

1. $l_{n0} = Bl$

2. for all $j \neq 0$ do

3. $l_{nj} = T$

4. change = true

5. while change do

6. { change = false

7. for $j = 1$ to $N - 1$ do

8. { temp = $\prod_{p \in \text{pred}(j)} f_{p}(l_{np})$

9. if temp $\neq l_{nj}$ then

10. { $l_{nj} = \text{temp}$

11. change = true

12. }

13. }

14. }

- Computation of $Out_{j}$ has been left implicit.
- Works fine for unidirectional frameworks.
Round Robin Iterative Algorithm

1. \( I_{n0} = B I \)
2. \[ \text{for all } j \neq 0 \text{ do} \]
3. \( I_n = \top \)
4. \( \text{change} = \text{true} \)
5. \[ \text{while } \text{change} \text{ do} \]
6. \[ \{ \text{change} = \text{false} \]
7. \[ \text{for } j = 1 \text{ to } N - 1 \text{ do} \]
8. \[ \{ \text{temp} = \prod_{p \in \text{pred}(j)} f_p(I_{n_p}) \]
9. \[ \text{if } \text{temp} \neq I_n \text{ then} \]
10. \[ \{ I_n = \text{temp} \]
11. \[ \text{change} = \text{true} \]
12. \[ \} \]
13. \[ \} \]
14. \}

- Computation of \( Out_j \) has been left implicit
- Works fine for unidirectional frameworks
- \( \top \) is the identity of \( \sqcap \) (line 3)
Round Robin Iterative Algorithm

1  $ln_0 = Bl$
2  for all $j \neq 0$ do
3      $ln_j = \top$
4  change = true
5 while change do
6      { change = false
7          for $j = 1$ to $N - 1$ do
8              { temp = $\prod_{p \in pred(j)} f_p(ln_p)$
9                  if temp $\neq ln_j$ then
10                     { ln_j = temp
11                        change = true
12                      }
13                  }
14  }

- Computation of $Out_j$ has been left implicit
  Works fine for unidirectional frameworks
- $\top$ is the identity of $\sqcap$ (line 3)
- Reverse postorder (rpo) traversal for efficiency (line 7)
Round Robin Iterative Algorithm

1 \( I_n^0 = B_l \)
2 for all \( j \neq 0 \) do
3 \( I_n^j = \top \)
4 change = true
5 while change do
6 { change = false
7 for \( j = 1 \) to \( N - 1 \) do
8 { temp = \( \prod_{p \in \text{pred}(j)} f_p(I_n^p) \)
9 if temp \( \neq I_n^j \) then
10 { \( I_n^j = \text{temp} \)
11 change = true
12 }
13 }
14 }

• Computation of \( Out_j \) has been left implicit
Works fine for unidirectional frameworks
• \( \top \) is the identity of \( \sqcap \) (line 3)
• Reverse postorder (rpo) traversal for efficiency (line 7)
• rpo traversal AND no loops \( \Rightarrow \) no need of initialization
Complexity of Round Robin Iterative Algorithm

- Unidirectional bit vector frameworks
  - Construct a spanning tree $T$ of $G$ to identify postorder traversal
  - Traverse $G$ in reverse postorder for forward problems and Traverse $G$ in postorder for backward problems
  - Depth $d(G, T)$: Maximum number of back edges in any acyclic path

<table>
<thead>
<tr>
<th>Task</th>
<th>Number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>First computation of $In$ and $Out$</td>
<td>1</td>
</tr>
<tr>
<td>Convergence (until $change$ remains true)</td>
<td>$d(G, T)$</td>
</tr>
<tr>
<td>Verifying convergence ($change$ becomes false)</td>
<td>1</td>
</tr>
</tbody>
</table>
### Complexity of Round Robin Iterative Algorithm

- **Unidirectional bit vector frameworks**
  - Construct a spanning tree $T$ of $G$ to identify postorder traversal
  - Traverse $G$ in reverse postorder for forward problems and
    Traverse $G$ in postorder for backward problems
  - Depth $d(G, T)$: Maximum number of back edges in any acyclic path

<table>
<thead>
<tr>
<th>Task</th>
<th>Number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>First computation of $In$ and $Out$</td>
<td>1</td>
</tr>
<tr>
<td>Convergence (until change remains true)</td>
<td>$d(G, T)$</td>
</tr>
<tr>
<td>Verifying convergence (change becomes false)</td>
<td>1</td>
</tr>
</tbody>
</table>

- What about bidirectional bit vector frameworks?
Complexity of Round Robin Iterative Algorithm

- Unidirectional bit vector frameworks
  - Construct a spanning tree $T$ of $G$ to identify postorder traversal
  - Traverse $G$ in reverse postorder for forward problems and Traverse $G$ in postorder for backward problems
  - Depth $d(G, T)$: Maximum number of back edges in any acyclic path

<table>
<thead>
<tr>
<th>Task</th>
<th>Number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>First computation of $In$ and $Out$</td>
<td>1</td>
</tr>
<tr>
<td>Convergence (until $change$ remains true)</td>
<td>$d(G, T)$</td>
</tr>
<tr>
<td>Verifying convergence ($change$ becomes false)</td>
<td>1</td>
</tr>
</tbody>
</table>

- What about bidirectional bit vector frameworks?
- What about other frameworks?
Example C Program with $d(G,T) = 2$

```c
1  void fun(int m, int n)
2  {
3      int i,j,a,b,c;
4      c=a+b;
5      i=0;
6      while(i<m)
7          {
8              j=0;
9              while(j<n)
10                 {
11                  a=i+j;
12                  j=j+1;
13                 }
14             i=i+1;
15          }
16      }
```
Example C Program with $d(G,T) = 2$

```c
void fun(int m, int n)
{
    int i, j, a, b, c;
    c = a + b;
    i = 0;
    while (i < m)
    {
        j = 0;
        while (j < n)
        {
            a = i + j;
            j = j + 1;
        }
        i = i + 1;
    }
}
```

Example C Program with \(d(G,T) = 2\)

```c
void fun(int m, int n) {
    int i, j, a, b, c;
    c = a + b;
    i = 0;
    while (i < m) {
        j = 0;
        while (j < n) {
            a = i + j;
            j = j + 1;
        }
        i = i + 1;
    }
}
```
Example C Program with \( d(G,T) = 2 \)

```c
void fun(int m, int n)
{
    int i, j, a, b, c;
    c = a + b;
    i = 0;
    while (i < m)
    {
        j = 0;
        while (j < n)
        {
            a = i + j;
            j = j + 1;
        }
        i = i + 1;
    }
}
```
Example C Program with $d(G,T) = 2$

```c
void fun(int m, int n)
{
    int i, j, a, b, c;
    c = a + b;
    i = 0;
    while (i < m)
    {
        j = 0;
        while (j < n)
        {
            a = i + j;
            j = j + 1;
        }
        i = i + 1;
    }
}
```

3 + 1 iterations for available expressions analysis
Complexity of Bidirectional Bit Vector Frameworks

Example: Consider the following CFG for PRE
Complexity of Bidirectional Bit Vector Frameworks

Example: Consider the following CFG for PRE

- Node numbers are in reverse post order
Example: Consider the following CFG for PRE

- Node numbers are in reverse post order
- Back edges in the graph are \( n_5 \to n_2 \) and \( n_{10} \to n_9 \).
Complexity of Bidirectional Bit Vector Frameworks

Example: Consider the following CFG for PRE

- Node numbers are in reverse post order
- Back edges in the graph are $n_5 \rightarrow n_2$ and $n_{10} \rightarrow n_9$.
- $d(G, T) = 1$
Complexity of Bidirectional Bit Vector Frameworks

Example: Consider the following CFG for PRE

- Node numbers are in reverse post order
- Back edges in the graph are $n_5 \rightarrow n_2$ and $n_{10} \rightarrow n_9$.
- $d(G, T) = 1$
- Actual iterations : 5
Complexity of Bidirectional Bit Vector Frameworks

Pairs of Out, In Values

<table>
<thead>
<tr>
<th>Initialization</th>
<th>Changes in Iterations</th>
<th>Final values &amp; transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>#2</td>
<td>#3</td>
</tr>
<tr>
<td>O, I</td>
<td>O, I</td>
<td>O, I</td>
</tr>
<tr>
<td>0, 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0, 1</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1, 1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1, 1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1, 1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1, 1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1, 1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1, 1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1, 1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1, 1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1, 1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1, 1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1, 1</td>
<td></td>
</tr>
</tbody>
</table>

May 2011

Uday Khedker
Complexity of Bidirectional Bit Vector Frameworks

Pairs of $Out,In$ Values

<table>
<thead>
<tr>
<th>Initialization</th>
<th>Changes in Iterations</th>
<th>Final values &amp; transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#1</td>
<td>#2</td>
</tr>
<tr>
<td>12</td>
<td>0,1</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1,1</td>
<td>1,0</td>
</tr>
<tr>
<td>5</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1,1</td>
<td>0,0</td>
</tr>
</tbody>
</table>

$Pairs of Out,In Values$
### Complexity of Bidirectional Bit Vector Frameworks

#### Pairs of $Out, In$ Values

<table>
<thead>
<tr>
<th>Initialization</th>
<th>Changes in Iterations</th>
<th>Final values &amp; transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 0,1</td>
<td>0,1 0,1 0,1 0,1 0,1</td>
<td>O,1</td>
</tr>
<tr>
<td>11 1,1</td>
<td>0,1</td>
<td></td>
</tr>
<tr>
<td>10 1,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 1,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 1,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 1,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 1,1</td>
<td>1,0</td>
<td></td>
</tr>
<tr>
<td>5 1,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 1,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 1,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 1,1</td>
<td>1,0</td>
<td></td>
</tr>
<tr>
<td>1 1,1</td>
<td>0,0</td>
<td></td>
</tr>
</tbody>
</table>

#### Diagram

![Diagram of bidirectional bit vector frameworks](image)
Complexity of Bidirectional Bit Vector Frameworks

Pairs of \textit{Out,In} Values

<table>
<thead>
<tr>
<th>Initialization</th>
<th>Changes in Iterations</th>
<th>Final values &amp; transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#1</td>
<td>#2</td>
</tr>
<tr>
<td>O,I</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0,1</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1,1</td>
<td>1,0</td>
</tr>
<tr>
<td>5</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1,1</td>
<td>1,0</td>
</tr>
<tr>
<td>1</td>
<td>1,1</td>
<td>0,0</td>
</tr>
</tbody>
</table>

\( b \times c \)
Complexity of Bidirectional Bit Vector Frameworks

Pairs of \(\text{Out}, \text{In} \) Values

<table>
<thead>
<tr>
<th>Initialization</th>
<th>Changes in Iterations</th>
<th>Final values &amp; transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#1</td>
<td>#2</td>
</tr>
<tr>
<td>O,1</td>
<td>O,1</td>
<td>O,1</td>
</tr>
<tr>
<td>0,1</td>
<td>0,0</td>
<td></td>
</tr>
<tr>
<td>1,1</td>
<td>0,1</td>
<td>0,0</td>
</tr>
<tr>
<td>1,1</td>
<td>0,1</td>
<td>0,0</td>
</tr>
<tr>
<td>1,1</td>
<td>1,0</td>
<td></td>
</tr>
<tr>
<td>1,1</td>
<td>0,0</td>
<td></td>
</tr>
<tr>
<td>1,1</td>
<td>0,0</td>
<td></td>
</tr>
<tr>
<td>1,1</td>
<td>0,0</td>
<td></td>
</tr>
<tr>
<td>1,1</td>
<td>0,0</td>
<td></td>
</tr>
<tr>
<td>1,1</td>
<td>0,0</td>
<td></td>
</tr>
<tr>
<td>1,1</td>
<td>0,0</td>
<td></td>
</tr>
</tbody>
</table>

May 2011

Uday Khedker
Complexity of Bidirectional Bit Vector Frameworks

Pairs of $Out, In$ Values

<table>
<thead>
<tr>
<th>Initialization</th>
<th>Changes in Iterations</th>
<th>Final values &amp; transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#1</td>
<td>#2</td>
</tr>
<tr>
<td>0,1</td>
<td>0,1</td>
<td>0,1</td>
</tr>
<tr>
<td>12</td>
<td>0,0</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0,1</td>
<td>0,0</td>
</tr>
<tr>
<td>11</td>
<td>0,0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0,1</td>
<td>0,1</td>
</tr>
<tr>
<td>9</td>
<td>0,1</td>
<td>0,1</td>
</tr>
<tr>
<td>8</td>
<td>0,1</td>
<td>0,1</td>
</tr>
<tr>
<td>7</td>
<td>0,1</td>
<td>0,1</td>
</tr>
<tr>
<td>6</td>
<td>1,0</td>
<td>0,0</td>
</tr>
<tr>
<td>5</td>
<td>0,1</td>
<td>0,1</td>
</tr>
<tr>
<td>4</td>
<td>0,0</td>
<td>0,0</td>
</tr>
<tr>
<td>3</td>
<td>0,0</td>
<td>0,0</td>
</tr>
<tr>
<td>2</td>
<td>0,0</td>
<td>0,0</td>
</tr>
<tr>
<td>1</td>
<td>0,0</td>
<td>0,0</td>
</tr>
</tbody>
</table>
Complexity of Bidirectional Bit Vector Frameworks

![Diagram of bidirectional bit vector frameworks]

<table>
<thead>
<tr>
<th>Initialization</th>
<th>Changes in Iterations</th>
<th>Final values &amp; transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#1</td>
<td>#2</td>
</tr>
<tr>
<td>0,1</td>
<td>O,1</td>
<td>O,1</td>
</tr>
<tr>
<td>0,1</td>
<td>0,0</td>
<td></td>
</tr>
<tr>
<td>1,1</td>
<td>0,1</td>
<td></td>
</tr>
<tr>
<td>1,1</td>
<td></td>
<td>0,1</td>
</tr>
<tr>
<td>1,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

May 2011  Uday Khedker
Complexity of Bidirectional Bit Vector Frameworks

![Graph of Bit Vector Frameworks]

<table>
<thead>
<tr>
<th>Initialization</th>
<th>Changes in Iterations</th>
<th>Final values &amp; transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#1  #2  #3  #4  #5</td>
<td></td>
</tr>
<tr>
<td>$0,1$</td>
<td>$0,1$  $0,1$  $0,1$  $0,1$  $0,1$</td>
<td>$0,1$</td>
</tr>
<tr>
<td>$12$</td>
<td>$0,1$  $0,0$</td>
<td>$0,0$</td>
</tr>
<tr>
<td>$11$</td>
<td>$1,1$  $0,1$</td>
<td>$0,0$</td>
</tr>
<tr>
<td>$11$</td>
<td>$1,1$  $0,1$</td>
<td>$0,1$ Delete</td>
</tr>
<tr>
<td>$10$</td>
<td>$1,1$  $0,1$</td>
<td>$0,1$ Insert</td>
</tr>
<tr>
<td>$9$</td>
<td>$1,1$  $1,0$</td>
<td>$1,0$ Insert</td>
</tr>
<tr>
<td>$8$</td>
<td>$1,1$  $1,0$</td>
<td>$1,0$ Insert</td>
</tr>
<tr>
<td>$7$</td>
<td>$1,1$  $0,0$</td>
<td>$0,0$</td>
</tr>
<tr>
<td>$6$</td>
<td>$1,1$  $1,0$</td>
<td>$0,0$</td>
</tr>
<tr>
<td>$5$</td>
<td>$1,1$  $0,0$</td>
<td>$0,0$</td>
</tr>
<tr>
<td>$4$</td>
<td>$1,1$  $0,1$</td>
<td>$0,0$</td>
</tr>
<tr>
<td>$3$</td>
<td>$1,1$  $0,0$</td>
<td>$0,0$</td>
</tr>
<tr>
<td>$2$</td>
<td>$1,1$  $1,0$</td>
<td>$0,0$</td>
</tr>
<tr>
<td>$1$</td>
<td>$1,1$  $0,0$</td>
<td>$0,0$</td>
</tr>
</tbody>
</table>
An Example of Information Flow in Our PRE Analysis

- \( PavIn_6 \) becomes 0 in the first iteration
- This causes many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)
An Example of Information Flow in Our PRE Analysis

- $PavIn_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)
An Example of Information Flow in Our PRE Analysis

- $P_{avln_6}$ becomes 0 in the first iteration
- This causes many other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)
An Example of Information Flow in Our PRE Analysis

- $P_{avln}_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)
An Example of Information Flow in Our PRE Analysis

- $PavIn_6$ becomes 0 in the first iteration
- This causes many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)
An Example of Information Flow in Our PRE Analysis

- $PavlIn_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)
An Example of Information Flow in Our PRE Analysis

- $PavIn_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)
An Example of Information Flow in Our PRE Analysis

- $PavIn_6$ becomes 0 in the first iteration
- This cause many all other values to become 0
- Here we see a particular sequence of changes
- Incorporating the effect of this sequence of changes requires 5 iterations
- Number of iterations is not related to depth (which is 1 for this graph)
An Example of Information Flow in Our PRE Analysis

- $PavIn_6$ becomes 0 in the first iteration.
- This causes many all other values to become 0.
- Here we see a particular sequence of changes.
- Incorporating the effect of this sequence of changes requires 5 iterations.
- Number of iterations is not related to depth (which is 1 for this graph).
Information Flow and Information Flow Paths

- Default value at each program point: $\top$
- Information flow path
Information Flow and Information Flow Paths

- Default value at each program point: $\top$
- Information flow path
  Sequence of adjacent program points
Information Flow and Information Flow Paths

- Default value at each program point: $\top$
- Information flow path
  - Sequence of adjacent program points along which data flow values change
Information Flow and Information Flow Paths

• Default value at each program point: \( \top \)

• *Information flow path*
  Sequence of adjacent program points along which data flow values change

• A change in the data flow at a program point could be
  
  ▶ *Generation of information*
  Change from \( \top \) to a non-\( \top \) due to local effect (i.e. \( f(\top) \neq \top \))
  
  ▶ *Propagation of information*
  Change from \( x \) to \( y \) such that \( y \sqsubseteq x \) due to global effect
Information Flow and Information Flow Paths

- Default value at each program point: $\top$

- **Information flow path**
  
  Sequence of adjacent program points along which data flow values change

- A change in the data flow at a program point could be
  
  - **Generation of information**
    Change from $\top$ to a non-$\top$ due to local effect (i.e. $f(\top) \neq \top$)
  
  - **Propagation of information**
    Change from $x$ to $y$ such that $y \sqsubseteq x$ due to global effect

- Information flow path (ifp) need not be a graph theoretic path
Edge and Node Flow Functions

$\begin{align*}
\text{In}_n & \quad \text{Out}_n \\
 f^n_n & \downarrow \quad f^n_{n \rightarrow m} \\
\text{In}_m & \quad \text{Out}_m \\
 f^b_m & \uparrow \quad f^b_{m \rightarrow n}
\end{align*}$
Edge and Node Flow Functions

Forward Node Flow Function

- $f_n^f$ from $n$ to $m$
- $f_m^f$ from $m$ to $n$
- $f_n^b$ from $n$ to $m$
- $f_m^b$ from $m$ to $n$

$In_n$, $Out_n$, $In_m$, $Out_m$, $In_m$, $Out_m$
Edge and Node Flow Functions

Forward Node Flow Function

Forward Edge Flow Function
Edge and Node Flow Functions

Forward Node Flow Function

\[ f_n \]

\[ f^f_n \rightarrow m \]

\[ f^f_{n \rightarrow m} \]

\[ f_m \]

\[ f^f_m \]

\[ In_n \]

\[ Out_n \]

\[ In_m \]

\[ Out_m \]

Forward Edge Flow Function

Backward Node Flow Function

\[ f_n^b \]

\[ f_{n \rightarrow m}^b \]

\[ f_m^b \]

\[ f^b_m \]

\[ In_n \]

\[ Out_n \]

\[ In_m \]

\[ Out_m \]

May 2011

Uday Khedker
Edge and Node Flow Functions

Forward Node Flow Function

\[ f^f_n \]

\[ f^f_{n \rightarrow m} \]

\[ f^f_m \]

\[ f^f_{m \rightarrow n} \]

\[ In_n \]

\[ Out_n \]

\[ In_m \]

\[ f^b_m \]

\[ f^b_{m \rightarrow n} \]

\[ Out_m \]

Backward Node Flow Function

\[ f^b_n \]

\[ f^b_{n \rightarrow m} \]

\[ In_n \]

\[ Out_n \]

\[ In_m \]

\[ f^b_m \]

\[ f^b_{m \rightarrow n} \]

\[ Out_m \]

Forward Edge Flow Function

Backward Edge Flow Function

May 2011

Uday Khedker
General Data Flow Equations

\[\begin{align*}
\text{In}_n &= \begin{cases} 
  B I_{\text{Start}} \cap f_n^b(\text{Out}_n) & n = \text{Start} \\
  \left( \bigcap_{m \in \text{pred}(n)} f_{m \rightarrow n}^f(\text{Out}_m) \right) \cap f_n^b(\text{Out}_n) & \text{otherwise}
\end{cases} \\
\text{Out}_n &= \begin{cases} 
  B I_{\text{End}} \cap f_n^f(\text{In}_n) & n = \text{End} \\
  \left( \bigcap_{m \in \text{succ}(n)} f_{m \rightarrow n}^b(\text{In}_m) \right) \cap f_n^f(\text{In}_n) & \text{otherwise}
\end{cases}
\end{align*}\]

- Edge flow functions are typically identity
  \[\forall x \in L, \ f(x) = x\]
- If particular flows are absent, the corresponding flow functions are
  \[\forall x \in L, \ f(x) = \top\]
Modelling Information Flows Using Edge and Node Flow Functions

Forward

\[ f_{k \rightarrow l} \circ f_{k \rightarrow j} \circ f_{i \rightarrow k} \]

Backward

\[ f_{i \rightarrow k} \circ f_{k \rightarrow j} \circ f_{k \rightarrow l} \]

Bidirectional

\[ f_{f_{k \rightarrow l} \circ f_{k \rightarrow j}} \]

Bidirectional

\[ f_{k \rightarrow l} \circ f_{k \rightarrow m} \]
Information could flow along arbitrary paths
Information Flow Paths in PRE

- Information could flow along arbitrary paths
Information Flow Paths in PRE

- Information could flow along arbitrary paths
Information Flow Paths in PRE

- Information could flow along arbitrary paths
Information Flow Paths in PRE

- Information could flow along arbitrary paths
Information Flow Paths in PRE

- Information could flow along arbitrary paths
Information Flow Paths in PRE

- Information could flow along arbitrary paths
Information could flow along arbitrary paths
Information Flow Paths in PRE

- Information could flow along arbitrary paths
- Theoretically predicted number: 144
Information Flow Paths in PRE

- Information could flow along arbitrary paths
- Theoretically predicted number: 144
- Actual iterations: 5
Information could flow along arbitrary paths
• Theoretically predicted number : 144
• Actual iterations : 5
• Not related to depth (1)
Lacuna with PRE Complexity

- Lacuna with PRE: Complexity $O(n^2)$ traversals.
  Practical graphs may have up to 50 nodes.
  - Predicted number of traversals: 2,500.
  - Practical number of traversals: $\leq 5$.

- No explanation for about 14 years despite dozens of efforts.

- Not much experimentation with performing advanced optimizations involving bidirectional dependency.
Complexity of Round Robin Iterative Method

- Buy OTC (Over-The-Counter) medicine.
- No U-Turn
- 1 Trip
Complexity of Round Robin Iterative Method

- Buy OTC (Over-The-Counter) medicine. No U-Turn 1 Trip
- Buy cloth. Give it to the tailor for stitching. No U-Turn 1 Trip
Complexity of Round Robin Iterative Method

- Buy OTC (Over-The-Counter) medicine. No U-Turn 1 Trip
- Buy cloth. Give it to the tailor for stitching. No U-Turn 1 Trip
- Buy medicine with doctor’s prescription. 1 U-Turn 2 Trips
Complexity of Round Robin Iterative Method

- Buy OTC (Over-The-Counter) medicine. No U-Turn 1 Trip
- Buy cloth. Give it to the tailor for stitching. No U-Turn 1 Trip
- Buy medicine with doctor’s prescription. 1 U-Turn 2 Trips
- Buy medicine with doctor’s prescription. 2 U-Turns 3 Trips

The diagnosis requires X-Ray.
Information Flow Paths and Width of a Graph

- A traversal $u \rightarrow v$ in an ifp is
  - *Compatible* if $u$ is visited before $v$ in the chosen graph traversal
  - *Incompatible* if $u$ is visited after $v$ in the chosen graph traversal
Information Flow Paths and Width of a Graph

• A traversal \( u \rightarrow v \) in an ifp is
  ▶ Compatible if \( u \) is visited before \( v \) in the chosen graph traversal
  ▶ Incompatible if \( u \) is visited after \( v \) in the chosen graph traversal

• Every incompatible edge traversal requires one additional iteration
Information Flow Paths and Width of a Graph

- A traversal $u \rightarrow v$ in an ifp is
  - Compatible if $u$ is visited before $v$ in the chosen graph traversal
  - Incompatible if $u$ is visited after $v$ in the chosen graph traversal

- Every incompatible edge traversal requires one additional iteration

- Width of a program flow graph with respect to a data flow framework
  Maximum number of incompatible traversals in any ifp, no part of which is bypassed
Information Flow Paths and Width of a Graph

- A traversal $u \rightarrow v$ in an ifp is
  - Compatible if $u$ is visited before $v$ in the chosen graph traversal
  - Incompatible if $u$ is visited after $v$ in the chosen graph traversal
- Every incompatible edge traversal requires one additional iteration
- Width of a program flow graph with respect to a data flow framework
  Maximum number of incompatible traversals in any ifp, no part of which is bypassed
- Width + 1 iterations are sufficient to converge on MFP solution
  (1 additional iteration may be required for verifying convergence)
Complexity of Bidirectional Bit Vector Frameworks

- Every “incompatible” edge traversal
  ⇒ One additional graph traversal
Complexity of Bidirectional Bit Vector Frameworks

- Every “incompatible” edge traversal ⇒ One additional graph traversal
- Max. Incompatible edge traversals = \textit{Width} of the graph = 0?
- Maximum number of traversals = 1 + Max. incompatible edge traversals
### Complexity of Bidirectional Bit Vector Frameworks

- Every “incompatible” edge traversal \( \Rightarrow \) One additional graph traversal

- Max. Incompatible edge traversals
  \( = \) Width of the graph = 1?

- Maximum number of traversals = 1 + Max. incompatible edge traversals

Graph Traversal

May 2011

Uday Khedker
Complexity of Bidirectional Bit Vector Frameworks

- Every "incompatible" edge traversal $\Rightarrow$ One additional graph traversal

- Max. Incompatible edge traversals $= Width$ of the graph $= 2$?

- Maximum number of traversals $= 1 + \text{Max. incompatible edge traversals}$
Complexity of Bidirectional Bit Vector Frameworks

- Every “incompatible” edge traversal \(\Rightarrow\) One additional graph traversal

- Max. Incompatible edge traversals = \(Width\) of the graph = 3?

- Maximum number of traversals = 1 + Max. incompatible edge traversals
Complexity of Bidirectional Bit Vector Frameworks

- Every "incompatible" edge traversal ⇒ One additional graph traversal
- Max. Incompatible edge traversals = \textit{Width} of the graph = 3?
- Maximum number of traversals = 1 + Max. incompatible edge traversals
Complexity of Bidirectional Bit Vector Frameworks

- Every “incompatible” edge traversal ⇒ One additional graph traversal

- Max. Incompatible edge traversals = Width of the graph = 3?

- Maximum number of traversals = 1 + Max. incompatible edge traversals
Complexity of Bidirectional Bit Vector Frameworks

- Every "incompatible" edge traversal $\Rightarrow$ One additional graph traversal

- Max. Incompatible edge traversals $= \text{Width of the graph} = 3$?

- Maximum number of traversals $= 1 + \text{Max. incompatible edge traversals}$
Complexity of Bidirectional Bit Vector Frameworks

- Every "incompatible" edge traversal \( \Rightarrow \) One additional graph traversal
- Max. Incompatible edge traversals = \( \text{Width} \) of the graph = 4
- Maximum number of traversals = \( 1 + \text{Max. incompatible edge traversals} \)
Complexity of Bidirectional Bit Vector Frameworks

- Every “incompatible” edge traversal \( \Rightarrow \) One additional graph traversal
- Max. Incompatible edge traversals = Width of the graph = 4
- Maximum number of traversals = \( 1 + 4 = 5 \)
Width Subsumes Depth

- Depth is applicable only to unidirectional data flow frameworks.
- Width is applicable to both unidirectional and bidirectional frameworks.
- For a given graph, $\text{Width} \leq \text{Depth}$
  Width provides a tighter bound.
Comparison Between Width and Depth

- Depth is purely a graph theoretic property whereas width depends on control flow graph as well as the data framework.
- Comparison between width and depth is meaningful only:
  - For unidirectional frameworks
  - When the direction of traversal for computing width is the natural direction of traversal.
- Since width excludes bypassed path segments, width can be smaller than depth.
Width and Depth

Assuming reverse postorder traversal for available expressions analysis

- Depth = 2
Width and Depth

\[ c = a + b \]

\[ i = 0 \]

\[ \text{if } (i < m) \]

\[ j = 0 \]

\[ \text{if } (j < n) \]

\[ a = i + j \]

\[ j = j + 1 \]

\[ i = i + 1 \]

Assuming reverse postorder traversal for available expressions analysis

- Depth = 2
- Information generation point \( n_5 \) kills expression “a + b”
Width and Depth

Assuming reverse postorder traversal for available expressions analysis

- Depth = 2
- Information generation point $n_5$ kills expression “$a + b$”
- Information propagation path $n_5 \rightarrow n_4 \rightarrow n_6 \rightarrow n_2$

No Gen or Kill for “$a + b$” along this path
Assuming reverse postorder traversal for available expressions analysis

- Depth = 2
- Information generation point \( n_5 \) kills expression “\( a + b \)"
- Information propagation path \( n_5 \rightarrow n_4 \rightarrow n_6 \rightarrow n_2 \)
  No Gen or Kill for “\( a + b \)” along this path
- Width = 2
Width and Depth

Assuming reverse postorder traversal for available expressions analysis

- Depth = 2
- Information generation point $n_5$ kills expression “$a + b$”
- Information propagation path $n_5 \rightarrow n_4 \rightarrow n_6 \rightarrow n_2$
  No Gen or Kill for “$a + b$” along this path
- Width = 2
- What about “$j + 1$”?
Width and Depth

Assuming reverse postorder traversal for available expressions analysis

- Depth = 2
- Information generation point $n_5$ kills expression “$a + b$”
- Information propagation path $n_5 \rightarrow n_4 \rightarrow n_6 \rightarrow n_2$
  No Gen or Kill for “$a + b$” along this path
- Width = 2
- What about “$j + 1$”? 
- Not available on entry to the loop
Width and Depth

Structures resulting from repeat-until loops with premature exits
- Depth = 3
Width and Depth

Structures resulting from repeat-until loops with premature exits

- Depth = 3
- However, any unidirectional bit vector is guaranteed to converge in $2 + 1$ iterations
Width and Depth

Structures resulting from repeat-until loops with premature exits

- Depth = 3
- However, any unidirectional bit vector is guaranteed to converge in $2 + 1$ iterations
- ifp 5 → 4 → 6 is bypassed by the edge 5 → 6
Width and Depth

Structures resulting from repeat-until loops with premature exits

- Depth $= 3$
- However, any unidirectional bit vector is guaranteed to converge in $2 + 1$ iterations
  - ifp $5 \rightarrow 4 \rightarrow 6$ is bypassed by the edge $5 \rightarrow 6$
  - ifp $6 \rightarrow 3 \rightarrow 6$ is bypassed by the edge $6 \rightarrow 7$
Width and Depth

Structures resulting from repeat-until loops with premature exits

- Depth = 3

- However, any unidirectional bit vector is guaranteed to converge in \(2 + 1\) iterations

- ifp \(5 \rightarrow 4 \rightarrow 6\) is bypassed by the edge \(5 \rightarrow 6\)

- ifp \(6 \rightarrow 3 \rightarrow 6\) is bypassed by the edge \(6 \rightarrow 7\)

- ifp \(7 \rightarrow 2 \rightarrow 8\) is bypassed by the edge \(7 \rightarrow 8\)
Width and Depth

Structures resulting from repeat-until loops with premature exits

- Depth = 3

- However, any unidirectional bit vector is guaranteed to converge in $2 + 1$ iterations
  - ifp $5 \rightarrow 4 \rightarrow 6$ is bypassed by the edge $5 \rightarrow 6$
  - ifp $6 \rightarrow 3 \rightarrow 6$ is bypassed by the edge $6 \rightarrow 7$
  - ifp $7 \rightarrow 2 \rightarrow 8$ is bypassed by the edge $7 \rightarrow 8$

- For forward unidirectional frameworks, width is 1
Width and Depth

Structures resulting from repeat-until loops with premature exits

- Depth = 3
- However, any unidirectional bit vector is guaranteed to converge in $2 + 1$ iterations
- ifp $5 \rightarrow 4 \rightarrow 6$ is bypassed by the edge $5 \rightarrow 6$
- ifp $6 \rightarrow 3 \rightarrow 6$ is bypassed by the edge $6 \rightarrow 7$
- ifp $7 \rightarrow 2 \rightarrow 8$ is bypassed by the edge $7 \rightarrow 8$
- For forward unidirectional frameworks, width is 1
- Splitting the bypassing edges and inserting nodes along those edges increases the width
Work List Based Iterative Algorithm

Directly traverses information flow paths

1. \( In_0 = BI \)
2. \textbf{for} all \( j \neq 0 \) \textbf{do}
3. \{ \( In_j = \top \)
4. Add \( j \) to LIST
5. \}
6. \textbf{while} LIST is not empty \textbf{do}
7. \{ Let \( j \) be the first node in LIST. Remove it from LIST
8. \( temp = \prod_{p \in \text{pred}(j)} f_p(In_p) \)
9. \textbf{if} \( temp \neq In_j \) \textbf{then}
10. \{ \( In_j = temp \)
11. Add all successors of \( j \) to LIST
12. \}
13. \}
Tutorial Problem

Perform work list based iterative analysis for earlier examples. Assume that the work list follows FIFO (First in First Out) policy.

Show the trace of the analysis in the following format:

<table>
<thead>
<tr>
<th>Step No.</th>
<th>Program Point Selected</th>
<th>Remaining Work list</th>
<th>Data Flow Value</th>
<th>Program Point(s) Added</th>
<th>Resulting Work list</th>
</tr>
</thead>
</table>

Part 5

Precise Modelling of General Flows
Complexity of Constant Propagation?

1

2 \[ a = b + 1 \]

3 \[ b = c + 1 \]

4 \[ c = d + 1 \]

5 \[ d = 2 \]
Complexity of Constant Propagation?

Iteration #1

May 2011
Complexity of Constant Propagation?

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

Iteration #1

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

Iteration #2
Complexity of Constant Propagation?

Iteration #1

Iteration #2

Iteration #3
Complexity of Constant Propagation?

Iteration #1

1. $a = b + 1$
2. $b = c + 1$
3. $c = d + 1$
4. $d = 2$

Iteration #3

1. $a = b + 1$
2. $b = 4$
3. $c = 3$
4. $d = 2$

Iteration #2

1. $a = b + 1$
2. $b = c + 1$
3. $c = d + 1$
4. $d = 2$

Iteration #4

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

May 2011
Loop Closures of Flow Functions

Paths Terminating at $p_2$ | Data Flow Value
--- | ---
$p_1, p_2$ | $x$
$p_1, p_2, p_3, p_2$ | $f(x)$
$p_1, p_2, p_3, p_2, p_3, p_2$ | $f^2(x)$
$p_1, p_2, p_3, p_2, p_3, p_2, p_3, p_2$ | $f^3(x)$
... | ...

May 2011 Uday Khedker
Loop Closures of Flow Functions

Paths Terminating at $p_2$ | Data Flow Value
---|---
$p_1, p_2$ | $x$
$p_1, p_2, p_3, p_2$ | $f(x)$
$p_1, p_2, p_3, p_2, p_3, p_2$ | $f(f(x)) = f^2(x)$
$p_1, p_2, p_3, p_2, p_3, p_2, p_3, p_2$ | $f(f(f(x))) = f^3(x)$
... | ...

- For static analysis we need to summarize the value at $p_2$ by a value which is safe after any iteration.

$$f^*(x) = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap f^4(x) \sqcap \ldots$$
Loop Closures of Flow Functions

Paths Terminating at $p_2$ | Data Flow Value
--- | ---
$p_1, p_2$ | $x$
$p_1, p_2, p_3, p_2$ | $f(x)$
$p_1, p_2, p_3, p_2, p_3, p_2$ | $f(f(x)) = f^2(x)$
$p_1, p_2, p_3, p_2, p_3, p_2, p_3, p_2$ | $f(f(f(x))) = f^3(x)$
... | ...

- For static analysis we need to summarize the value at $p_2$ by a value which is safe after any iteration.

$$f^*(x) = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap f^4(x) \sqcap \ldots$$

- $f^*$ is called the loop closure of $f$. 

May 2011

Uday Khedker
Loop Closures in Bit Vector Frameworks

- Flow functions in bit vector frameworks have constant $Gen$ and $Kill$

\[
\begin{align*}
    f^*(x) &= x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots \\
    f^2(x) &= f \left( Gen \cup (x - Kill) \right) \\
        &= Gen \cup ((Gen \cup (x - Kill)) - Kill) \\
        &= Gen \cup ((Gen - Kill) \cup (x - Kill)) \\
        &= Gen \cup (Gen - Kill) \cup (x - Kill) \\
        &= Gen \cup (x - Kill) = f(x) \\
    f^*(x) &= x \sqcap f(x)
\end{align*}
\]
Loop Closures in Bit Vector Frameworks

- Flow functions in bit vector frameworks have constant *Gen* and *Kill*

\[ f^*(x) = x \cap f(x) \cap f^2(x) \cap f^3(x) \cap \ldots \]
\[ f^2(x) = f(Gen \cup (x - Kill)) \]
\[ = Gen \cup ((Gen \cup (x - Kill)) - Kill) \]
\[ = Gen \cup ((Gen - Kill) \cup (x - Kill)) \]
\[ = Gen \cup (Gen - Kill) \cup (x - Kill) \]
\[ = Gen \cup (x - Kill) = f(x) \]
\[ f^*(x) = x \cap f(x) \]

- *Loop Closures of Bit Vector Frameworks are 2-bounded.*
Loop Closures in Bit Vector Frameworks

- Flow functions in bit vector frameworks have constant \( \text{Gen} \) and \( \text{Kill} \)

\[
f^*(x) = x \sqcap f(x) \sqcap f^2(x) \sqcap f^3(x) \sqcap \ldots \\
f^2(x) = f(\text{Gen} \cup (x - \text{Kill})) \\
= \text{Gen} \cup ((\text{Gen} \cup (x - \text{Kill})) - \text{Kill}) \\
= \text{Gen} \cup ((\text{Gen} - \text{Kill}) \cup (x - \text{Kill})) \\
= \text{Gen} \cup (\text{Gen} - \text{Kill}) \cup (x - \text{Kill}) \\
= \text{Gen} \cup (x - \text{Kill}) = f(x) \\
f^*(x) = x \sqcap f(x)
\]

- Loop Closures of Bit Vector Frameworks are 2-bounded.

- Intuition: Since \( \text{Gen} \) and \( \text{Kill} \) are constant, same things are generated or killed in every application of \( f \).

Multiple applications of \( f \) are not required unless the input value changes.
Larger Values of Loop Closure Bounds

- **Fast Frameworks** \( \equiv \) 2-bounded frameworks (eg. bit vector frameworks)
  Both these conditions must be satisfied
  - **Separability**
    Data flow values of different entities are independent
  - **Constant or Identity Flow Functions**
    Flow functions for an entity are either constant or identity

- **Non-fast frameworks**
  At least one of the above conditions is violated
Separability

\[ f : L \leftrightarrow L \text{ is } \langle \hat{h}_1, \hat{h}_2, \ldots, \hat{h}_m \rangle \text{ where } \hat{h}_i \text{ computes the value of } \hat{x}_i \]
Separability

\[ f : L \mapsto L \text{ is } \langle \hat{h}_1, \hat{h}_2, \ldots, \hat{h}_m \rangle \text{ where } \hat{h}_i \text{ computes the value of } \hat{x}_i \]

<table>
<thead>
<tr>
<th>Separable</th>
<th>Non-Separable</th>
</tr>
</thead>
</table>

Example: All bit vector frameworks  
Example: Constant Propagation
Separability

\[ f : L \rightarrow L \text{ is } \langle \hat{h}_1, \hat{h}_2, \ldots, \hat{h}_m \rangle \text{ where } \hat{h}_i \text{ computes the value of } \hat{x}_i \]

\[ \langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle \]

**Separable**

\[ f \]

\[ \langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle \]

**Non-Separable**

\[ \langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle \]

\[ f \]

\[ \langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle \]

Example: All bit vector frameworks

Example: Constant Propagation
Separability

$f : L \mapsto L$ is $\langle \hat{h}_1, \hat{h}_2, \ldots, \hat{h}_m \rangle$ where $\hat{h}_i$ computes the value of $\hat{x}_i$

Separable

$\langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle$

Non-Separable

$\langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle$

Example: All bit vector frameworks

Example: Constant Propagation
Separability

\( f : L \mapsto L \) is \( \langle \hat{h}_1, \hat{h}_2, \ldots, \hat{h}_m \rangle \) where \( \hat{h}_i \) computes the value of \( \hat{x}_i \).

**Separable**

\[ \langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle \]

\[ \hat{h}_2 \]

\[ \langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle \]

\[ \hat{h} : \hat{L} \mapsto \hat{L} \]

**Non-Separable**

\[ \langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle \]

\[ f \]

\[ \langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle \]

Example: All bit vector frameworks

Example: Constant Propagation
Separability

\( f : L \mapsto L \) is \( \langle \hat{h}_1, \hat{h}_2, \ldots, \hat{h}_m \rangle \) where \( \hat{h}_i \) computes the value of \( \hat{x}_i \)

**Separable**

\[
\langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle
\]

\[
\langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle
\]

\[
\hat{h} : \hat{L} \mapsto \hat{L}
\]

Example: All bit vector frameworks

**Non-Separable**

\[
\langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle
\]

\[
\langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle
\]

Example: Constant Propagation
Separability

\( f : L \mapsto L \) is \( \langle \hat{h}_1, \hat{h}_2, \ldots, \hat{h}_m \rangle \) where \( \hat{h}_i \) computes the value of \( \hat{x}_i \)

**Separable**

\[
\langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle
\]

\[
\langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle
\]

\[\hat{h} : L \mapsto \hat{L}\]

Example: All bit vector frameworks

**Non-Separable**

\[
\langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m \rangle
\]

\[
\langle \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m \rangle
\]

\[\hat{h} : L \mapsto \hat{L}\]

Example: Constant Propagation
Separability of Bit Vector Frameworks

- $\hat{L}$ is $\{0, 1\}$, $L$ is $\{0, 1\}^m$
- $\hat{\cap}$ is either boolean AND or boolean OR
- $\hat{\top}$ and $\hat{\bot}$ are 0 or 1 depending on $\hat{\cap}$.
- $\hat{h}$ is a \textit{bit function} and could be one of the following:

<table>
<thead>
<tr>
<th>$\text{Raise}$</th>
<th>$\text{Lower}$</th>
<th>$\text{Propagate}$</th>
<th>$\text{Negate}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{\top}$</td>
<td>$\hat{\bot}$</td>
<td>$\hat{\top}$</td>
<td>$\hat{\bot}$</td>
</tr>
<tr>
<td>$\hat{\bot}$</td>
<td>$\hat{\top}$</td>
<td>$\hat{\bot}$</td>
<td>$\hat{\top}$</td>
</tr>
</tbody>
</table>
Separability of Bit Vector Frameworks

- $\hat{L}$ is $\{0, 1\}$, $L$ is $\{0, 1\}^m$
- $\hat{\cap}$ is either boolean AND or boolean OR
- $\hat{\top}$ and $\hat{\bot}$ are 0 or 1 depending on $\hat{\cap}$.
- $\hat{h}$ is a *bit function* and could be one of the following:

<table>
<thead>
<tr>
<th></th>
<th>Raise</th>
<th>Lower</th>
<th>Propagate</th>
<th>Negate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{\top}$</td>
<td>$\hat{\top}$</td>
<td>$\hat{\top}$</td>
<td>$\hat{\top}$</td>
<td>$\hat{\bot}$</td>
</tr>
<tr>
<td>$\hat{\bot}$</td>
<td>$\hat{\bot}$</td>
<td>$\hat{\bot}$</td>
<td>$\hat{\bot}$</td>
<td>$\hat{\top}$</td>
</tr>
</tbody>
</table>

- Non-monotonicity
**Boundedness of Constant Propagation**

1. 
2. 
3. 
4. $a = b + 1$
5. $b = c + 1$
6. $c = a + 1$
7. 

May 2011

Uday Khedker
Boundedness of Constant Propagation

Summary flow function:
(data flow value at node 7)

\[ f(\langle v_a, v_b, v_c \rangle) = \langle 1 \sqcap (v_b + 1), (v_c + 1), (v_a + 1) \rangle \]
Boundedness of Constant Propagation

Summary flow function:
(data flow value at node 7)

\[ f(\langle v_a, v_b, v_c \rangle) = \langle 1 \sqcap (v_b + 1), (v_c + 1), (v_a + 1) \rangle \]

\[ f^0(\top) = \langle \top, \top, \top \rangle \]
\[ f^1(\top) = \langle 1, \top, \top \rangle \]
Boundedness of Constant Propagation

Summary flow function:
(data flow value at node 7)

\[
f(\langle v_a, v_b, v_c \rangle) = \langle 1 \sqcap (v_b + 1), (v_c + 1), (v_a + 1) \rangle
\]

\[
f^0(\top) = \langle \hat{\top}, \hat{\top}, \hat{\top} \rangle
\]
\[
f^1(\top) = \langle 1, \hat{\top}, \hat{\top} \rangle
\]
\[
f^2(\top) = \langle 1, \hat{\top}, 2 \rangle
\]
Boundedness of Constant Propagation

Summary flow function:
(data flow value at node 7)

\[ f(\langle v_a, v_b, v_c \rangle) = \langle 1 \sqcap (v_b + 1), (v_c + 1), (v_a + 1) \rangle \]

\[
\begin{align*}
    f^0(\top) &= \langle \hat{\top}, \hat{\top}, \hat{\top} \rangle \\
    f^1(\top) &= \langle 1, \hat{\top}, \hat{\top} \rangle \\
    f^2(\top) &= \langle 1, \hat{\top}, 2 \rangle \\
    f^3(\top) &= \langle 1, 3, 2 \rangle 
\end{align*}
\]
Boundedness of Constant Propagation

Summary flow function:
(data flow value at node 7)

\[ f(\langle v_a, v_b, v_c \rangle) = \langle 1 \sqcap (v_b + 1), (v_c + 1), (v_a + 1) \rangle \]

\[ f^0(\top) = \langle \top, \top, \top \rangle \]
\[ f^1(\top) = \langle 1, \top, \top \rangle \]
\[ f^2(\top) = \langle 1, \top, 2 \rangle \]
\[ f^3(\top) = \langle 1, 3, 2 \rangle \]
\[ f^4(\top) = \langle \bot, 3, 2 \rangle \]

May 2011
Uday Khedker
Boundedness of Constant Propagation

Summary flow function:
(data flow value at node 7)

\[ f(\langle v_a, v_b, v_c \rangle) = \langle 1 \sqcap (v_b + 1), (v_c + 1), (v_a + 1) \rangle \]

\[
\begin{align*}
    f^0(\top) &= \langle \wedge, \wedge, \wedge \rangle \\
    f^1(\top) &= \langle 1, \wedge, \wedge \rangle \\
    f^2(\top) &= \langle 1, \wedge, 2 \rangle \\
    f^3(\top) &= \langle 1, 3, 2 \rangle \\
    f^4(\top) &= \langle \bot, 3, 2 \rangle \\
    f^5(\top) &= \langle \bot, 3, \bot \rangle 
\end{align*}
\]
Boundedness of Constant Propagation

Summary flow function:
(data flow value at node 7)

\[ f(\langle v_a, v_b, v_c \rangle) = \langle 1 \sqcap (v_b + 1), \]
\[ (v_c + 1), \]
\[ (v_a + 1) \rangle \]

\[ f^0(\top) = \langle \hat{\top}, \hat{\top}, \hat{\top} \rangle \]
\[ f^1(\top) = \langle 1, \hat{\top}, \hat{\top} \rangle \]
\[ f^2(\top) = \langle 1, \hat{\top}, 2 \rangle \]
\[ f^3(\top) = \langle 1, 3, 2 \rangle \]
\[ f^4(\top) = \langle \bot, 3, 2 \rangle \]
\[ f^5(\top) = \langle \bot, 3, \bot \rangle \]
\[ f^6(\top) = \langle \bot, \bot, \bot \rangle \]
Boundedness of Constant Propagation

Summary flow function:
(data flow value at node 7)

\[ f(\langle v_a, v_b, v_c \rangle) = \langle 1 \sqcap (v_b + 1), (v_c + 1), (v_a + 1) \rangle \]

\[
\begin{align*}
f^0(⊤) &= \langle \hat{⊤}, \hat{⊤}, \hat{⊤} \rangle \\
f^1(⊤) &= \langle 1, \hat{⊤}, \hat{⊤} \rangle \\
f^2(⊤) &= \langle 1, \hat{⊤}, 2 \rangle \\
f^3(⊤) &= \langle 1, 3, 2 \rangle \\
f^4(⊤) &= \langle \hat{⊥}, 3, 2 \rangle \\
f^5(⊤) &= \langle \hat{⊥}, 3, \hat{⊥} \rangle \\
f^6(⊤) &= \langle \hat{⊥}, \hat{⊥}, \hat{⊥} \rangle \\
f^7(⊤) &= \langle \hat{⊥}, \hat{⊥}, \hat{⊥} \rangle 
\end{align*}
\]
Boundedness of Constant Propagation

\[ f^*(\top) = \prod_{i=0}^{6} f^i(\top) \]
Boundedness of Constant Propagation

The moral of the story:

- The data flow value of every variable could change twice
The moral of the story:

- The data flow value of every variable could change twice
- In the worst case, only one change may happen in every step of a function application
Boundedness of Constant Propagation

The moral of the story:

• The data flow value of every variable could change twice
• In the worst case, only one change may happen in every step of a function application
• Maximum number of steps: $2 \times |\text{Var}|$
The moral of the story:

- The data flow value of every variable could change twice
- In the worst case, only one change may happen in every step of a function application
- Maximum number of steps: $2 \times |\text{Var}|$
- Boundedness parameter $k$ is $(2 \times |\text{Var}|) + 1$
Modelling Flow Functions for General Flows

- General flow functions can be written as

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

where \( \text{Gen} \) and \( \text{Kill} \) have constant and dependent parts

\[ \text{Gen}_n(X) = \text{ConstGen}_n \cup \text{DepGen}_n(X) \]

\[ \text{Kill}_n(X) = \text{ConstKill}_n \cup \text{DepKill}_n(X) \]
Modelling Flow Functions for General Flows

• General flow functions can be written as

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

where Gen and Kill have constant and dependent parts

\[ \text{Gen}_n(X) = \text{ConstGen}_n \cup \text{DepGen}_n(X) \]
\[ \text{Kill}_n(X) = \text{ConstKill}_n \cup \text{DepKill}_n(X) \]

• The dependent parts take care of
  ▶ dependence across different entities as well as
  ▶ dependence on the value of the same entity in the argument X
Modelling Flow Functions for General Flows

- General flow functions can be written as

\[ f_n(X) = (X - Kill_n(X)) \cup Gen_n(X) \]

where \( Gen \) and \( Kill \) have constant and dependent parts

\[ Gen_n(X) = ConstGen_n \cup DepGen_n(X) \]
\[ Kill_n(X) = ConstKill_n \cup DepKill_n(X) \]

- The dependent parts take care of
  - dependence across different entities as well as
  - dependence on the value of the same entity in the argument \( X \)

- Bit vector frameworks are a special case

\[ DepGen_n(X) = DepKill_n(X) = \emptyset \]
Component Lattice for Integer Constant Propagation

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

<table>
<thead>
<tr>
<th>$\hat{\sqcap}$</th>
<th>$\langle v, ? \rangle$</th>
<th>$\langle v, \times \rangle$</th>
<th>$\langle v, c_1 \rangle$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle v, ? \rangle$</td>
<td>$\langle v, ? \rangle$</td>
<td>$\langle v, \times \rangle$</td>
<td>$\langle v, c_1 \rangle$</td>
</tr>
<tr>
<td>$\langle v, \times \rangle$</td>
<td>$\langle v, \times \rangle$</td>
<td>$\langle v, \times \rangle$</td>
<td>$\langle v, \times \rangle$</td>
</tr>
<tr>
<td>$\langle v, c_2 \rangle$</td>
<td>$\langle v, c_2 \rangle$</td>
<td>$\langle v, \times \rangle$</td>
<td>If $c_1 = c_2$ then $\langle v, c_1 \rangle$ else $\langle v, \times \rangle$</td>
</tr>
</tbody>
</table>
Flow Functions for Constant Propagation

- Flow function for $r = a_1 \ast a_2$

<table>
<thead>
<tr>
<th>$mult$</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>$\langle r, (c_1 \ast c_2) \rangle$</td>
</tr>
</tbody>
</table>
## Defining Data Flow Equations for Constant Propagation

<table>
<thead>
<tr>
<th>Value</th>
<th>ConstGen&lt;sub&gt;n&lt;/sub&gt;</th>
<th>DepGen&lt;sub&gt;n&lt;/sub&gt;(X)</th>
<th>ConstKill&lt;sub&gt;n&lt;/sub&gt;</th>
<th>DepKill&lt;sub&gt;n&lt;/sub&gt;(X)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>v = c, c ∈ Const</code></td>
<td><code>{⟨v, c⟩}</code></td>
<td>∅</td>
<td>∅</td>
<td>`{⟨v, d⟩</td>
</tr>
<tr>
<td><code>v = e, e ∈ Expr</code></td>
<td>∅</td>
<td><code>{⟨v, eval(e, X)⟩}</code></td>
<td>∅</td>
<td>`{⟨v, d⟩</td>
</tr>
<tr>
<td><code>read(v)</code></td>
<td><code>{⟨v, ×⟩}</code></td>
<td>∅</td>
<td>∅</td>
<td>`{⟨v, d⟩</td>
</tr>
<tr>
<td><code>other</code></td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>
## Defining Data Flow Equations for Constant Propagation

<table>
<thead>
<tr>
<th></th>
<th>$\text{ConstGen}_n$</th>
<th>$\text{DepGen}_n(X)$</th>
<th>$\text{ConstKill}_n$</th>
<th>$\text{DepKill}_n(X)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v = c$, $c \in \text{Const}$</td>
<td>${\langle v, c \rangle}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>${\langle v, d \rangle</td>
</tr>
<tr>
<td>$v = e$, $e \in \text{Expr}$</td>
<td>$\emptyset$</td>
<td>${\langle v, \text{eval}(e,X) \rangle}$</td>
<td>$\emptyset$</td>
<td>${\langle v, d \rangle</td>
</tr>
<tr>
<td>$\text{read}(v)$</td>
<td>${\langle v, \times \rangle}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>${\langle v, d \rangle</td>
</tr>
<tr>
<td>other</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

### eval($a_1$ op $a_2$, $X$)

<table>
<thead>
<tr>
<th></th>
<th>$\langle a_1, ? \rangle \in X$</th>
<th>$\langle a_1, \times \rangle \in X$</th>
<th>$\langle a_1, c_1 \rangle \in X$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle a_2, ? \rangle \in X$</td>
<td>?</td>
<td>$\times$</td>
<td>?</td>
</tr>
<tr>
<td>$\langle a_2, \times \rangle \in X$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>$\langle a_2, c_2 \rangle \in X$</td>
<td>$?$</td>
<td>$\times$</td>
<td>$c_1$ op $c_2$</td>
</tr>
</tbody>
</table>
Example Program for Constant Propagation

1. `read(e);`
2. `a = 7; b = 2; f = e; if (f > 0)`
   - True branch:
     - `a = 2; if (f ≥ e + 2)`
       - True branch:
         - `b = c + 1; if (b ≥ 7)`
           - True branch:
             - `f = f + 1`;
           - False branch:
             - `true`
         - False branch:
           - `false`
     - False branch:
       - `false`
       - `false`
       - `false`
       - `false`
   - False branch:
     - `true`
     - `false`
     - `true`
     - `true`
     - `false`

3. `false`

4. `c = d * a;`
5. `d = a + b;`
6. `if (f ≥ e + 1)`
   - True branch:
     - `d = a + 1; f = f + 1`
   - False branch:
     - `false`

7. `false`

8. `false`

9. `false`

10. `e = a + b;`
# Result of Constant Propagation

<table>
<thead>
<tr>
<th></th>
<th>Iteration #1</th>
<th>Changes in iteration #2</th>
<th>Changes in iteration #3</th>
<th>Changes in iteration #4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$In_{n_1}$</td>
<td>$\top, \top, \top, \top, \top, \top$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Out_{n_1}$</td>
<td>$\top, \top, \top, \top, \bot, \top$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$In_{n_2}$</td>
<td>$\top, \top, \top, \bot, \bot, \bot$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Out_{n_2}$</td>
<td>$7, 2, \top, \top, \bot, \bot$</td>
<td>$\bot, 2, \top, 3, \bot, \bot$</td>
<td>$\bot, 2, 6, 3, \bot, \bot$</td>
<td>$\bot, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$In_{n_3}$</td>
<td>$7, 2, \top, \top, \bot, \bot$</td>
<td>$\bot, 2, \top, 3, \bot, \bot$</td>
<td>$\bot, 2, 6, 3, \bot, \bot$</td>
<td>$\bot, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$Out_{n_3}$</td>
<td>$2, 2, \top, \top, \bot, \bot$</td>
<td>$\bot, 2, \top, 3, \bot, \bot$</td>
<td>$\bot, 2, 6, 3, \bot, \bot$</td>
<td>$\bot, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$In_{n_4}$</td>
<td>$2, 2, \top, \top, \bot, \bot$</td>
<td>$\bot, 2, \top, 3, \bot, \bot$</td>
<td>$\bot, 2, 6, 3, \bot, \bot$</td>
<td>$\bot, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$Out_{n_4}$</td>
<td>$2, \top, \top, \top, \bot, \bot$</td>
<td>$\bot, 2, \top, 3, \bot, \bot$</td>
<td>$\bot, 2, 6, 3, \bot, \bot$</td>
<td>$\bot, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$In_{n_5}$</td>
<td>$2, \top, \top, \top, \bot, \bot$</td>
<td>$\bot, 2, \top, 3, \bot, \bot$</td>
<td>$\bot, 2, 6, 3, \bot, \bot$</td>
<td>$\bot, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$Out_{n_5}$</td>
<td>$2, \top, \top, \top, \bot, \bot$</td>
<td>$\bot, 2, \top, 3, \bot, \bot$</td>
<td>$\bot, 2, 6, 3, \bot, \bot$</td>
<td>$\bot, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$In_{n_6}$</td>
<td>$2, \top, \top, \bot, \bot$</td>
<td>$\bot, 2, \top, 3, \bot, \bot$</td>
<td>$\bot, 2, 6, 3, \bot, \bot$</td>
<td>$\bot, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$Out_{n_6}$</td>
<td>$2, \top, \top, \bot, \bot$</td>
<td>$\bot, 2, \top, 3, \bot, \bot$</td>
<td>$\bot, 2, 6, 3, \bot, \bot$</td>
<td>$\bot, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$In_{n_7}$</td>
<td>$2, \top, \top, \bot, \bot$</td>
<td>$\bot, 2, \top, 3, \bot, \bot$</td>
<td>$\bot, 2, 6, 3, \bot, \bot$</td>
<td>$\bot, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$Out_{n_7}$</td>
<td>$2, \top, \top, \bot, \bot$</td>
<td>$\bot, 2, \top, 3, \bot, \bot$</td>
<td>$\bot, 2, 6, 3, \bot, \bot$</td>
<td>$\bot, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$In_{n_8}$</td>
<td>$2, \top, \top, \bot, \bot$</td>
<td>$\bot, 2, \top, 3, \bot, \bot$</td>
<td>$\bot, 2, 6, 3, \bot, \bot$</td>
<td>$\bot, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$Out_{n_8}$</td>
<td>$2, \top, 4, \bot, \bot$</td>
<td>$\bot, 2, \top, 4, \bot, \bot$</td>
<td>$\bot, 2, 6, 4, \bot, \bot$</td>
<td>$\bot, \bot, 6, \bot, \bot$</td>
</tr>
<tr>
<td>$In_{n_9}$</td>
<td>$2, \top, 4, \bot, \bot$</td>
<td>$\bot, 2, \top, 4, \bot, \bot$</td>
<td>$\bot, 2, 6, 4, \bot, \bot$</td>
<td>$\bot, \bot, 6, \bot, \bot$</td>
</tr>
<tr>
<td>$Out_{n_9}$</td>
<td>$2, \top, 3, \bot, \bot$</td>
<td>$\bot, 2, \top, 3, \bot, \bot$</td>
<td>$\bot, 2, 6, 3, \bot, \bot$</td>
<td>$\bot, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$In_{n_{10}}$</td>
<td>$\bot, 2, \top, \top, \bot, \bot$</td>
<td>$\bot, 2, \top, 3, \bot, \bot$</td>
<td>$\bot, 2, 6, 3, \bot, \bot$</td>
<td>$\bot, \bot, 6, 3, \bot, \bot$</td>
</tr>
<tr>
<td>$Out_{n_{10}}$</td>
<td>$\bot, 2, \top, \top, \bot, \bot$</td>
<td>$\bot, 2, \top, 3, \bot, \bot$</td>
<td>$\bot, 2, 6, 3, \bot, \bot$</td>
<td>$\bot, \bot, 6, 3, \bot, \bot$</td>
</tr>
</tbody>
</table>
Monotonicity of Constant Propagation

- Flow function $f_n(X) = (X - Kill_n(X)) \cup Gen_n(X)$ where

$$
\begin{align*}
Gen_n(X) &= \text{ConstGen}_n \cup \text{DepGen}_n(X) \\
Kill_n(X) &= \text{ConstKill}_n \cup \text{DepKill}_n(X)
\end{align*}
$$

- $\text{ConstGen}_n$ and $\text{ConstKill}_n$ are trivially monotonic

- To show $X_1 \sqsubseteq X_2 \Rightarrow \text{DepGen}_n(X_1) \sqsubseteq \text{DepGen}_n(X_2)$ we need to show that $X_1 \sqsubseteq X_2 \Rightarrow \text{eval}(e, X_1) \sqsubseteq \text{eval}(e, X_2)$. This follows from definition of $\text{eval}(e, X)$.

- To show $X_1 \sqsubseteq X_2 \Rightarrow (X_1 - \text{DepKill}_n(X_1)) \sqsubseteq (X_2 - \text{Dep Kill}_n(X_2))$ observe that $\text{DepKill}_n$ removes the pair corresponding to the variable modified in statement $n$. Data flow values of other variables remain unaffected.