Theoretical Abstractions in Data Flow Analysis

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

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

August 2017

Part 1

About These Slides

Copyright

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

  (Indian edition published by Ane Books in 2013)

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


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

Outline

- The need for a more general setting
- The set of data flow values
- The set of flow functions
- Solutions of data flow analyses
- Algorithms for performing data flow analysis
- Complexity of data flow analysis
Part 2

The Need for a More General Setting

• We seem to have covered many variations
• Yet there are analyses that do not fit the same mould of bit vector frameworks
• We use an analysis called Constant Propagation to observe the differences

A variable \( v \) is a constant with value \( c \) at program point \( p \) if in every execution instance of \( p \), the value of \( v \) is \( c \)

An Introduction to Constant Propagation

<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>Some</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

Summary Values

- \( (a, b, c, d) \) where \( a, b, c, d \) are data flow values
- \( (?, ?, ?, ?) \) indicating no constant values
- \( (1, 2, 3, ?) \) indicating an unspecified constant value
- \( (\times, \times, 3, 2) \) indicating a constant value
- \( (a = 1, b = 2, c = a + b) \)
- \( (c = a + b, d = a * b) \)
- \( d = c - 1, a = 2, b = 1, c = a + b \)

Execution Sequence

- \( n_1 \)
- \( n_2 \)
- \( n_3 \)

Difference #1: Data Flow Values

- Tuples of the form \( \langle \eta_1, \eta_2, \ldots, \eta_k \rangle \) where \( \eta_i \) is the data flow value for \( i^{th} \) variable
- Unlike bit vector frameworks, value \( \eta_i \) 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 \( c_i \) if the value of \( v_i \) is known to be \( c_i \) at compile time

Difference #2: Dependence of Data Flow Values Across Entities

- In bit vector frameworks, data flow values of different entities are independent
  - Liveness of variable \( b \) does not depend on that of any other variable
  - Availability of expression \( a * b \) does not depend on that of any other expression
- Given a statement \( a = b * c \), can the constantness of \( a \) be determined independently of the constantness of \( b \) and \( c \)?

No

Desired Solution

Aug 2017
IIT Bombay
Difference #3: Confluence Operation

- Confluence operation \( (a, c_1) \cap (a, c_2) \)

\[
\begin{array}{c|c|c|c}
(\cdot, ?) & (\cdot, \times) & (a, c_1) \\
(\cdot, ?) & (\cdot, \times) & (a, c_1) \\
(\cdot, \times) & (\cdot, \times) & (a, \times) \\
(\cdot, c_2) & (\cdot, c_2) & (\cdot, \times) \\
\end{array}
\]

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

What are its properties?

Difference #4: Flow Functions for Constant Propagation

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

\[
\begin{array}{c|c|c|c}
\text{mult} & (a_1, ?) & (a_1, \times) & (a_1, c_1) \\
(a_2, ?) & (r, ?) & (r, \times) & (r, ?) \\
(a_2, \times) & (r, \times) & (r, \times) & (r, \times) \\
(a_2, c_2) & (r, ?) & (r, \times) & (r, (c_1 \ast c_2)) \\
\end{array}
\]

- 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 \( n \)

Difference #5: Solution Computed by Iterative Method

<table>
<thead>
<tr>
<th>Iteration #1</th>
<th>Iteration #2</th>
<th>Iteration #3</th>
<th>Desired solution</th>
</tr>
</thead>
</table>
| \( a = 1 \)
| \( b = 2 \)
| \( c = a + b \) | \( 1, 2, 3, ? \)
| \( 1, 2, 3, ? \) | (\( 1, 2, 3, ? \) | \( 1, 2, 3, ? \) |

<table>
<thead>
<tr>
<th>Iteration #1</th>
<th>Iteration #2</th>
<th>Iteration #3</th>
<th>Desired solution</th>
</tr>
</thead>
</table>
| \( c = a + b \)
| \( d = a \ast b \) | \( 1, 2, 3, ? \)
| \( x, x, 3, 2 \) | \( x, x, 3, 2 \) | \( x, x, 3, 2 \) |
| \( x, x, 3, 2 \) | \( x, x, 3, 2 \) | \( x, x, 3, 2 \) |

<table>
<thead>
<tr>
<th>Iteration #1</th>
<th>Iteration #2</th>
<th>Iteration #3</th>
<th>Desired solution</th>
</tr>
</thead>
</table>
| \( d = c - 1 \)
| \( a = 2 \)
| \( b = 1 \)
| \( c = a + b \) | \( 1, 2, 3, ? \)
| \( 2, 1, 3, x \) | \( 2, 1, 3, x \) | \( 2, 1, 3, x \) |

Issues in Data Flow Analysis

- Representation
- Approximation: Partial Order, Lattices
- Existence, Computability
- Soundness, Precision
- Merge: Commutativity, Associativity, Idempotence
- Flow Functions: Monotonicity, Distributivity, Boundedness, Separability
- Complexity, efficiency
- Convergence
- Initialization
Part 3

Data Flow Values: An Overview

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

Part 4

A Digression on Lattices

Partially Ordered Sets

Sets in which elements can be compared and ordered

- **Total order.** Every element in comparable with every element (including itself)
- **Discrete order.** Every element is comparable only with itself but not with any other element
- **Partial order.** An element is comparable with some but not necessarily all elements
### Partially Ordered Sets and Lattices

**Partially ordered sets**

Partial order $\subseteq$ is reflexive, transitive, and antisymmetric.

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

An upper bound of $x, y$ is $u$ s.t. $x \subseteq u$ and $y \subseteq u$.

**Lattices**

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

- glb must be related to all other lower bounds. Hence it must be unique.

**Set** $\{1, 2, 3, 4, 6, 9, 12\}$ with $\subseteq$ relation as "divides" (i.e. $a \subseteq b$ iff $a$ divides $b$)

- Subset $\{4, 9, 6\}$ and $\{12, 9\}$ do not have an upper bound in the set.
Lattice

Set \{1, 2, 3, 4, 6, 9, 12, 18, 36\} with \subset relation as “divides”

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 “less-than-equal-to” (\(\leq\)) 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

- What about the empty set?
  
  - \(\text{glb}(\emptyset) = \top\)
    
    Every element of \(\mathbb{Z} \cup \{\infty, -\infty\}\) is vacuously a lower bound of an element in \(\emptyset\)
    
    OR
    
    Every element in \(\emptyset\) is stronger than every element in \(\mathbb{Z} \cup \{\infty, -\infty\}\) (because there is no element in \(\emptyset\))
    
    The greatest among these lower bounds is \(\top\)
  
  - \(\text{lub}(\emptyset) = \bot\)

Operations on Lattices

- Meet (\(\cap\)) and Join (\(\cup\))
  
  - \(x \cap y\) computes the glb of \(x\) and \(y\)
    
    \(z = x \cap y \Rightarrow z \subseteq x \land z \subseteq y\)
    
    \(x \cup y\) computes the lub of \(x\) and \(y\)
    
    \(z = x \cup y \Rightarrow z \supseteq x \land z \supseteq y\)
    
    \(\cap\) and \(\cup\) are commutative, associative, and idempotent

- Top (\(\top\)) and Bottom (\(\bot\)) elements
  
  \(\forall x \in L, \ x \cap \top = x\)
  
  \(\forall x \in L, \ x \cup \top = \top\)
  
  \(\forall x \in L, \ x \cap \bot = \bot\)
  
  \(\forall x \in L, \ x \cup \bot = x\)

- \(x \cap y = \text{gcd}(x, y)\)

- \(x \cup y = \text{lcm}(x, y)\)
Partial Order and Operations

- For a lattice $\sqsubseteq$ induces $\sqcap$ and $\sqcup$ and vice-versa
- The choices of $\sqsubseteq$, $\sqcap$, and $\sqcup$ cannot be arbitrary
  They have to be
  - consistent with each other, and
  - definable in terms of each other
- For some variants of lattices, $\sqcap$ or $\sqcup$ may not exist
  Yet the requirement of its consistency with $\sqsubseteq$ cannot be violated

Finite Lattices are Complete

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

Available Expressions

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Partially Available Expressions Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(\top)$</td>
<td>$(\top)$</td>
</tr>
<tr>
<td>${e_1, e_2, e_3}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>${e_1, e_2}$</td>
<td>${e_1, e_3}$</td>
</tr>
<tr>
<td>${e_2, e_3}$</td>
<td>${e_2}$</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>${e_1}$</td>
</tr>
<tr>
<td>$(\bot)$</td>
<td>${e_1, e_2, e_3}$</td>
</tr>
</tbody>
</table>

Lattice for May-Must Analysis

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

<table>
<thead>
<tr>
<th>No</th>
<th>Must</th>
<th>Interpreting data flow values</th>
</tr>
</thead>
<tbody>
<tr>
<td>May</td>
<td>$\bot$</td>
<td>$\text{No}$. Information does not hold along any path</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\text{Must}$. Information must hold along all paths</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\text{May}$. Information may hold along some path</td>
</tr>
</tbody>
</table>

- An artificial $\top$ can be added

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
- A bounded lattice iff $L$ is a lattice and has $\top$ and $\bot$ elements
A Bounded Lattice Need Not be Complete (1)

- Let $A$ be all finite subsets of $\mathbb{Z}$
  Then, $A$ is an infinite set
- The poset $L = (A \cup \{\mathbb{Z}\}, \subseteq)$ is a bounded lattice with $\top = \mathbb{Z}$ and $\bot = \emptyset$
  The join $\cup$ of this lattice is $\cup$
- To see why, consider a set $S$ containing those subsets of $L$ that do not contain the number 1
  There are two possibilities:
  - $S$ contains only a finite number of sets that do not contain 1 (say $S_f$)
    $\Rightarrow S_f$ is a finite set
  - $S$ contains all finite sets that do not contain 1 (say $S_\infty$)
    $\Rightarrow S_\infty$ is a infinite set

A Bounded Lattice Need Not be Complete (2)

- $S_f$ contains only a finite number of sets each of which does not contain 1
  - The union of all its member sets is a finite set excluding 1
  - Thus $S_f$ has a lub in $L$
- $S_\infty$ contains all finite sets that do not contain 1
  - Since the number of such sets is infinite, their union is an infinite set
  - $\mathbb{Z} \setminus \{1\}$ is not contained in $L$ (the only infinite set in $L$ is $\mathbb{Z}$)
  - $S_\infty$ does not have a lub in $L$

Hence $L$ is not complete

It may be tempting to assume that $\mathbb{Z}$ is the lub of $S_\infty$
because it is an upper bound of $S_\infty$ and no other upper bound of $S_\infty$ in the lattice is weaker $\mathbb{Z}$
- However, the join operation $\cup$ of $L$ does not compute $\mathbb{Z}$ as the lub of $S_\infty$ (because it must exclude 1)
- The join operation $\cup$ is inconsistent with the partial order $\supseteq$ of $L$. Hence we say that join does not exist for $S_\infty$
A Bounded Lattice Need Not be Complete (2)

- A bounded lattice $L$ has a glb and lub of $L$ in $L$
- A complete lattice $L$ should have glb and lub of all subsets of $L$
- A lattice $L$ should have glb and lub of all finite non-empty subsets of $L$

Ascending and Descending Chains

- Strictly ascending chain $x \sqsubseteq y \sqsubseteq \cdots \sqsubseteq z$
- Strictly descending chain $x \sqsupseteq y \sqsupseteq \cdots \sqsupseteq z$
- DCC: Descending Chain Condition
  All strictly descending chains are finite
- ACC: Ascending Chain Condition
  All strictly ascending chains are finite

Complete Lattice and Ascending and Descending Chains

- 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 $\bot$

Variants of Lattices

- Meet Semilattices
- Join Semilattices
- Bounded lattices
- Complete lattices
- Complete lattices with dcc and acc
  - dcc: descending chain condition
  - acc: ascending chain condition
Maintain $n$ servers and divide the traffic
- Each server maintains an $n$-tuple for each page
- Updates the counters for its own slot

Like for Page 1
- Server Blue
  - Page 1: 1 0 0 0
  - Page 2: 0 0 0 0
  - Page 3: 0 0 0 0

Like for Page 3
- Server Blue
  - Page 1: 1 0 0 0
  - Page 2: 0 0 0 0
  - Page 3: 0 0 0 1
An Example of Lattices: Maintaining Like Counts on Cloud

### Synchronize:
- Send the data to other servers
- Update the counters using point-wise max
An Example of Lattices: Maintaining Like Counts on Cloud

Synchronize:
- Send the data to other servers
- Update the counters using point-wise max

- Lattice of \( n \)-tuples using point-wise \( \geq \) as the partial order
  \[
  \langle x_1, x_2, \ldots, x_n \rangle \sqsubseteq \langle y_1, y_2, \ldots, y_n \rangle = (x_1 \geq y_1) \land (x_2 \geq y_2) \ldots \land (x_n \geq y_n)
  \]

- Tuples merged with max operation
  \[
  \langle x_1, x_2, \ldots, x_n \rangle \sqcap \langle y_1, y_2, \ldots, y_n \rangle = \langle \max(x_1, y_1), \max(x_2, y_2), \ldots, \max(x_n, y_n) \rangle
  \]
An Example of Lattices: Maintaining Like Counts on Cloud

**Synchronize:**
- Send the data to other servers
- Update the counters using point-wise max

<table>
<thead>
<tr>
<th>Server Blue</th>
<th>Server Red</th>
<th>Server Green</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 1</td>
<td>1 1 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>Page 2</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>Page 3</td>
<td>2 1 0</td>
<td>2 1 1</td>
</tr>
<tr>
<td>Page 4</td>
<td>2 1 0</td>
<td>2 1 1</td>
</tr>
</tbody>
</table>

After synchronization, all servers have the same data.

Count for a page:
- Take sum of all counts at any server for the page.
Constructing Lattices

- Powerset construction with subset or superset relation
- Products of lattices
  - Cartesian product
  - Lexicographic product
  - Interval product
  - Set of mappings
- Lattices on sequences using prefix or suffix as partial orders

---

Cartesian Product of Lattice

\[
\begin{align*}
\langle L_N, \subseteq_N, \cap_N, \cup_N \rangle \times_a b \\
\langle L_A, \subseteq_A, \cap_A, \cup_A \rangle =
\langle L_C, \subseteq_C, \cap_C, \cup_C \rangle
\end{align*}
\]

Example of Cartesian Product: Concept Lattices

- **Context of concepts.** A collection of objects and their attributes
- **Concepts.** Sets of attributes as exhibited by specific objects
  - A concept \( C \) is a pair \((O, A)\) where
    \( O \) is a set of objects exhibiting attributes in the set \( A \)
    Every object in \( O \) has every attribute in \( A \)
- **Partial order.** \((O_2, A_2) \subseteq (O_1, A_1) \iff O_2 \subseteq O_1\)
  - Very few objects have all properties
  - Since \( A \) is the set of attributes common to all objects in \( O \),
    \[
    O_2 \subseteq O_1 \Rightarrow A_2 \supseteq A_1
    \]
    As the number of chosen objects decreases, the number of common attributes increases
Example of Concept Lattice (1)

From *Introduction to Lattices and Order* by Davey and Priestley [2002]

<table>
<thead>
<tr>
<th>Size</th>
<th>Distance from Sun</th>
<th>Moon?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small (ss)</td>
<td>Medium (sm)</td>
<td>Large (sl)</td>
</tr>
<tr>
<td>Mercury Me</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Venus V</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Earth E</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Mars Ma</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Jupiter J</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Saturn S</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Uranus U</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Neptune N</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Pluto P</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Example of Concept Lattice (2)

We write \((O, A)\) as

\[
\begin{align*}
\{Me, V, E, Ma, J, S, U, N, P\} & \subseteq O \text{ A} \\
\{\} & \subseteq O \text{ A} \\
\{Me, V, E, Ma, P\} & \subseteq O \text{ A} \\
\{E, Ma, J, S, U, N, P\} & \subseteq O \text{ A} \\
\{ss\} & \subseteq O \text{ A} \\
\{my\} & \subseteq O \text{ A} \\
\{Me, V\} & \subseteq O \text{ A} \\
\{E, Ma\} & \subseteq O \text{ A} \\
\{P\} & \subseteq O \text{ A} \\
\{J, S\} & \subseteq O \text{ A} \\
\{U, N\} & \subseteq O \text{ A} \\
\{ss, dn, mn\} & \subseteq O \text{ A} \\
\{ss, dn, my\} & \subseteq O \text{ A} \\
\{ss, df, my\} & \subseteq O \text{ A} \\
\{ss, sl, df, my\} & \subseteq O \text{ A} \\
\{ss, sm, sl, df, my, mn\} & \subseteq O \text{ A} \\
\end{align*}
\]

Variants of Products

In each case \(L \subseteq L_1 \times L_2\)

- Cartesian Product
  \((x_1, x_2) \subseteq (y_1, y_2)\) \iff \(x_1 \subseteq_1 y_1 \land x_2 \subseteq_2 y_2\)

- Interval Product
  \((x_1, x_2) \subseteq (y_1, y_2)\) \iff \(x_1 \subseteq_1 y_1 \land x_2 \subseteq_2 y_2\)

- Lexicographic Product
  \((x_1, x_2) \subseteq (y_1, y_2)\) \iff \((x_1 \sqsubseteq_1 y_1) \lor (x_1 = y_1 \land x_2 \subseteq_2 y_2)\)

- Set of mappings \(L_1 \rightarrow L_2\)
  \((x_1, x_2) \subseteq (y_1, y_2)\) \iff \(x_1 = y_1 \land x_2 \subseteq_2 y_2\)
The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition

- Requirement: glb must exist for all non-empty finite subsets
- Corollary: ⊥ 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 \( z \))
    \( \Rightarrow \) 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 ⊥ must exist
- \( \top \) 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

\[
\begin{align*}
\{e_1, e_2, e_3\} & \quad Y \\
\{e_1, e_2\} & \quad \subseteq 111 \\
\{e_1, e_3\} & \quad \subseteq 110 \\
\{e_2, e_3\} & \quad \subseteq 101 \\
\{e_1\} & \quad \subseteq 011 \\
\{e_2\} & \quad \subseteq 001 \\
\{e_3\} & \quad \subseteq 000 \\
\emptyset & \quad X
\end{align*}
\]

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
  ▶ Approximating Money
    Earnings: Rs. 1050 can be safely approximated by Rs. 1000
    Expenses: Rs. 1050 can be safely approximated by Rs. 1100
  ▶ Approximating Time
    Travel time: 2 hours required can be safely approximated by 3 hours
    Study time: 3 available days can be safely assumed to be only 2 days

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

- Will not do incorrect optimization
- May prohibit correct optimization
- Will not miss any correct optimization
- May enable incorrect 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</td>
<td>A dead variable</td>
<td>A live variable is</td>
</tr>
<tr>
<td></td>
<td>elimination</td>
<td>is considered live</td>
<td>considered dead</td>
</tr>
<tr>
<td>Available</td>
<td>Common</td>
<td>An available</td>
<td>A non-available</td>
</tr>
<tr>
<td></td>
<td>subexpression</td>
<td>expression is</td>
<td>expression is</td>
</tr>
<tr>
<td></td>
<td>elimination</td>
<td>considered live</td>
<td>considered non-available</td>
</tr>
</tbody>
</table>

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 $y$
  - $x \sqcap y$ will not compute a value weaker than $y$
  - It may be unsafe, though

We want most exhaustive information which is also safe

Most Approximate Values in a Complete Lattice

- **Top.** $\forall x \in L, x \sqsubseteq \top$ Exhaustive approximation of all values
  - 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.** $\forall x \in L, \bot \sqsubseteq x$ Safe approximation of all values
  - 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\} \downarrow \{e_1\}
- \downarrow \{e_2\}
- \downarrow \{e_3\}
- \{e_1, e_2, e_3\}

### Live Variables Analysis

- $\emptyset \sqsubseteq \{v_1\}
- \{v_1\} \downarrow \{v_1, v_2\}
- \downarrow \{v_1, v_3\}
- \downarrow \{v_1, v_2, v_3\}
- \emptyset \sqsubseteq \{v_1, v_2, v_3\}
- \{v_1, v_2, v_3\}

$\sqsubseteq$ is $\subseteq$

$\sqcap$ is $\cap$

$\sqcup$ is $\cup$
Partial Order Relation

**Reflexive**
\[ x \sqsubseteq x \]
x can be safely used in place of x

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

**Antisymmetric**
\[ x \sqsubseteq y, y \sqsubseteq x \iff x = y \]
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

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

\[ L = \text{Lattice for all expressions} \]
\[ \hat{L} = \text{Lattice for a single expression} \]

<table>
<thead>
<tr>
<th>111</th>
<th>101</th>
<th>110</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>011</td>
<td>001</td>
</tr>
<tr>
<td></td>
<td>000</td>
<td></td>
</tr>
</tbody>
</table>

(Expressions e is available)
- \( 1 \) or \( \{e\} \)
- \( 0 \) or \( \emptyset \)

(Expressions e is not available)

Component Lattice for Data Flow Information Represented By Bit Vectors

- \( \top \) or Boolean AND
- \( \bot \) or Boolean OR

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

- \( L = \hat{L} \times \hat{L} \times \hat{L} \) and \( x = (\hat{x}_1, \hat{x}_2, \hat{x}_3) \in L \) where \( \hat{x}_i \in \hat{L} \)
- \( \subseteq = \hat{\subseteq} \times \hat{\subseteq} \times \hat{\subseteq} \) and \( n = \hat{n} \times \hat{n} \times \hat{n} \)
- \( \top = \hat{\top} \times \hat{\top} \times \hat{\top} \) and \( \bot = \hat{\bot} \times \hat{\bot} \times \hat{\bot} \)
Component Lattice for Integer Constant Propagation

- Overall lattice $L$ is the set of mappings from variables to $\hat{L}$
- $\sqcap$ and $\hat{\sqcap}$ get defined by $\sqsubseteq$ and $\hat{\sqsubseteq}$

<table>
<thead>
<tr>
<th>$\sqcap$</th>
<th>$(a, ud)$</th>
<th>$(a, nc)$</th>
<th>$(a, c_1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(a, ud)$</td>
<td>$(a, ud)$</td>
<td>$(a, nc)$</td>
<td>$(a, nc)$</td>
</tr>
<tr>
<td>$(a, nc)$</td>
<td>$(a, nc)$</td>
<td>$(a, nc)$</td>
<td>$(a, nc)$</td>
</tr>
</tbody>
</table>

If $c_1 = c_2$ then $(a, c_1)$ else $(a, nc)$

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

Component Lattice for Must Points-To Analysis

- A pointer can point to at most one location

Combined Total and Partial Availability Analysis

- Two bits per expression rather than one. Can be implemented using AND (as below) or using OR (reversed lattice)

- What approximation of safety does this lattice capture?
  Uncertain information (= no optimization) is guaranteed to be safe
General Lattice for May-Must Analysis

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

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

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 \(f : L \rightarrow L\) such that
  - \(F\) contains an identity function
  - \(F\) is closed under composition
  - 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
  - Loop Closure Boundedness and 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
    \[ f(x) = \text{Gen} \cup (x - \text{Kill}) \]
  - Lattices are powersets with partial orders as \( \subseteq \) or \( \supseteq \) relations
  - Information is merged using \( \cap \) or \( \cup \)
- Flow functions in Strong Liveness 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 \( f(y) \)
  \[ \forall x, y \in L, x \subseteq y \Rightarrow f(x) \subseteq f(y) \]
- Alternative definition
  \[ \forall x, y \in L, f(x \cap y) \subseteq f(x) \cap f(y) \]
- 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
  \[ \forall x, y \in L, f(x \cap y) = f(x) \cap f(y) \]
- Merging at intermediate points in shared segments of paths does not lead to imprecision
Monotonicity and Distributivity

> Distributive and hence monotonic
> Monotonic but not distributive

Distributivity of Bit Vector Frameworks

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

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

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

Why is Constant Propagation Non-Distributive?

- \( x = (1, 2, 3, ud) \) (Along \( Out_{n_1} \rightarrow In_{n_1} \))
- \( y = (2, 1, 3, 2) \) (Along \( Out_{n_2} \rightarrow In_{n_2} \))
- Function application for block \( n_2 \) before merging
  \[ f(x) \cap f(y) = f((1, 2, 3, ud)) \cap f((2, 1, 3, 2)) \]
  \[ = (1, 2, 3, 2) \cap (2, 1, 3, 2) \]
  \[ = (\hat{\perp}, \hat{\perp}, 3, 2) \]

- Function application for block \( n_2 \) after merging
  \[ f(x \cap y) = f((1, 2, 3, ud) \cap (2, 1, 3, 2)) \]
  \[ = f((\hat{\perp}, \hat{\perp}, 3, 2)) \]
  \[ = (\hat{\perp}, \hat{\perp}, \hat{\perp}, \hat{\perp}) \]

- \( f(x \cap y) \not\subseteq f(x) \cap f(y) \)

Non-Distributivity of Constant Propagation

Why is Constant Propagation Non-Distributive?

Possible combinations due to merging

- \( a = 1 \)
- \( b = 2 \)
- \( a = 2 \)
- \( b = 1 \)
- \( c = a + b = 3 \)

- Correct combination
Why is Constant Propagation Non-Distributive?

Possible combinations due to merging

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

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

- Correct combination

\[
\begin{align*}
  a &= 1 \\
  b &= 2 \\
  c &= a + b = 3
\end{align*}
\]

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

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

Part 7

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

An Example For Available Expressions Analysis

Program

<table>
<thead>
<tr>
<th>Input</th>
<th>A₀</th>
<th>A₁</th>
<th>A₂</th>
<th>A₃</th>
<th>A₄</th>
<th>A₅</th>
<th>A₆</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

Lattice \( L \) of data flow values at a node

\[ L \times L \times L \times L \]

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 \( Info(p) \subseteq MoP(p) \) is safe
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

<table>
<thead>
<tr>
<th>MoP</th>
<th>MFP</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td></td>
</tr>
<tr>
<td>(a = 1)</td>
<td>(\top, \top, \top, \top)</td>
</tr>
<tr>
<td>(b = 2)</td>
<td>(1, 2, 3, \top)</td>
</tr>
<tr>
<td>(c = a + b)</td>
<td>(\top, \top, 3, \top)</td>
</tr>
</tbody>
</table>

n2
\(c = a + b\)
\(d = a * b\)
\(\top, \top, 3, 2\)
\(\top, \top, 3, 2\)
\(\top, \top, 3, 2\)
\(\top, \top, 3, 2\)
\(\top, \top, 3, 2\)
\(\top, \top, 3, 2\)

n3
\(d = c - 1\)
\(a = 2\)
\(b = 1\)
\(c = a + b\)
\(\top, \top, 3, 2\)
\(\top, \top, 3, 2\)
\(\top, \top, 3, 2\)
\(\top, \top, 3, 2\)
\(\top, \top, 3, 2\)

Possible Assignments as Solutions of Data Flow Analyses

- All possible assignments
  - \(\forall i, In_i = Out_i = \top\)
  - Meet Over Paths Assignment
- All safe assignments
  - Maximum Fixed Point
- All fixed point solutions
  - Least Fixed Point
  - \(\forall i, In_i = Out_i = \bot\)
An Instance of Available Expressions Analysis

Lattice

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

Constant Functions

<table>
<thead>
<tr>
<th>f</th>
<th>f(x)</th>
<th>g</th>
<th>g(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>f_1</td>
<td>a+b</td>
<td>f_2</td>
<td>b+c</td>
</tr>
<tr>
<td>f_2</td>
<td>a+b</td>
<td>f_3</td>
<td>b+c</td>
</tr>
<tr>
<td>f_3</td>
<td>a+b</td>
<td>f_4</td>
<td>x - a+b</td>
</tr>
<tr>
<td>f_4</td>
<td>a+b</td>
<td>f_5</td>
<td>x - b+c</td>
</tr>
</tbody>
</table>

Dependent Functions

<table>
<thead>
<tr>
<th>f</th>
<th>f(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>f_1</td>
<td>a+b</td>
</tr>
<tr>
<td>f_2</td>
<td>a+b</td>
</tr>
<tr>
<td>f_3</td>
<td>a+b</td>
</tr>
<tr>
<td>f_4</td>
<td>a+b</td>
</tr>
<tr>
<td>f_5</td>
<td>a+b</td>
</tr>
</tbody>
</table>

Some Possible Assignments

<table>
<thead>
<tr>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
<th>A6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Program

- [a+b, b+c] → [a+b] → [b+c] → ∅
- [a+b] → [a+b] → [b+c] → a+b
- [b+c] → a+b → [b+c] → x
- [a+b] → [a+b] → [b+c] → x
- [b+c] → a+b → [b+c] → x

Flow Functions

<table>
<thead>
<tr>
<th>Node</th>
<th>Flow Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>In1</td>
<td>00 00 00 00 00 00</td>
</tr>
<tr>
<td>Out1</td>
<td>11 00 11 11 11 11</td>
</tr>
<tr>
<td>In2</td>
<td>11 11 00 10 01 01</td>
</tr>
<tr>
<td>Out2</td>
<td>11 11 00 10 01 10</td>
</tr>
</tbody>
</table>

- Is the lattice a meet semilattice?
- What is the meet operation that computes glb?
- Are all strictly descending chains finite?
- Does the function space have an identity function?
- Are all values in the lattice computable from a finite merge of flow functions?
- Is the function space closed under composition?
### An Instance of Available Expressions Analysis

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

**Program**

1. \(a+b\) \(b+c\)
2. \(f_1\)
3. \(f_2\) \(f_{id}\)

**Some Possible Assignments**

<table>
<thead>
<tr>
<th>Node</th>
<th>Flow Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>In1</td>
<td>00</td>
</tr>
<tr>
<td>Out1</td>
<td>11</td>
</tr>
<tr>
<td>In2</td>
<td>11</td>
</tr>
<tr>
<td>Out2</td>
<td>11</td>
</tr>
</tbody>
</table>

**Flow Functions**

<table>
<thead>
<tr>
<th>Node</th>
<th>Flow Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>In1</td>
<td>00</td>
</tr>
<tr>
<td>Out1</td>
<td>11</td>
</tr>
<tr>
<td>In2</td>
<td>11</td>
</tr>
<tr>
<td>Out2</td>
<td>11</td>
</tr>
</tbody>
</table>

---

### An Instance of Available Expressions Analysis

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

**Program**

1. \(a+b\) \(b+c\)
2. \(f_1\)
3. \(f_2\) \(f_{id}\)

**Some Possible Assignments**

<table>
<thead>
<tr>
<th>Node</th>
<th>Flow Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>In1</td>
<td>00</td>
</tr>
<tr>
<td>Out1</td>
<td>11</td>
</tr>
<tr>
<td>In2</td>
<td>11</td>
</tr>
<tr>
<td>Out2</td>
<td>11</td>
</tr>
</tbody>
</table>

**Flow Functions**

<table>
<thead>
<tr>
<th>Node</th>
<th>Flow Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>In1</td>
<td>00</td>
</tr>
<tr>
<td>Out1</td>
<td>11</td>
</tr>
<tr>
<td>In2</td>
<td>11</td>
</tr>
<tr>
<td>Out2</td>
<td>11</td>
</tr>
</tbody>
</table>
Lattice of Assignments for Available Expressions Analysis

Program

<table>
<thead>
<tr>
<th>Some Assignments</th>
<th>A₀</th>
<th>A₁</th>
<th>A₂</th>
<th>A₃</th>
<th>A₄</th>
<th>A₅</th>
<th>A₆</th>
</tr>
</thead>
<tbody>
<tr>
<td>In₁</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>Out₁</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>In₂</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>Out₂</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

Lattice \(L \times L \times L \times L\) for all assignments (many assignments omitted, e.g. node 1 could have data flow values 10 and 01)

Fixed point assignments

Safe assignments

Existence of an MoP Assignment (1)

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

- If a finite number of paths reach \(p\), then existence of solution trivially follows
  - Function space is closed under composition
  - \(\text{glb}\) exists for all non-empty finite subsets of the lattice

(Assuming that the data flow values form a meet semilattice)

Existence of an MoP Assignment (2)

\[
MoP(p) = f_{\rho_1}(BI) \cap f_{\rho_2}(BI) \cap f_{\rho_3}(BI) \cap \ldots
\]

- Every meet results in a weaker value
- The sequence \(X_1, X_2, X_3, \ldots\) follows a descending chain
- Since all strictly descending chains are finite, MoP exists

(Assuming that our meet semilattice satisfies DCC)

Computability of MoP

Does existence of MoP imply it is computable?

\[
\begin{array}{|c|c|}
\hline
\text{Paths reaching the entry of } p_2 & \text{Data Flow Value} \\
\hline
p_1, p_2 & x \\
\hline
p_1, p_2, p_3, p_2 & f(x) \\
\hline
p_1, p_2, p_3, p_2, p_5, p_2 & f(f(x)) = f^2(x) \\
\hline
p_1, p_2, p_3, p_2, p_3, p_2, p_3, p_2 & f(f(f(x))) = f^3(x) \\
\hline
\ldots & \ldots \\
\hline
\end{array}
\]

\[
MoP(p_2) = x \cap f(x) \cap f^2(x) \cap f^3(x) \cap f^4(x) \cap \ldots
\]
MoP Computation is Undecidable

There does not exist any algorithm that can compute MoP assignment for every possible instance of every possible monotone data flow framework

- Reducing MPCP (Modified Post’s Correspondence Problem) to constant propagation
- MPCP is known to be undecidable
- If an algorithm exists for detecting all constants
  ⇒ MPCP would be decidable
- Since MPCP is undecidable
  ⇒ There does not exist an algorithm for detecting all constants
  ⇒ Static analysis is undecidable

Aug 2017 IIT Bombay

Post’s Correspondence Problem (PCP)

Given strings $u_i, v_i \in \Sigma^+$ for some alphabet $\Sigma$, and two $k$-tuples,

$$U = (u_1, u_2, \ldots, u_k)$$
$$V = (v_1, v_2, \ldots, v_k)$$

Is there a sequence $i_1, i_2, \ldots, i_m$ of one or more integers such that

$$u_{i_1} u_{i_2} \ldots u_{i_m} = v_{i_1} v_{i_2} \ldots v_{i_m}$$

- The first string in the correspondence relation should be the first string from the $k$-tuple
  $$u_1 u_2 \ldots u_m = v_1 v_2 \ldots v_m$$
- For $U = (11, 1011, 10), V = (111, 10, 0)$, the solution is 2, 1, 3
- For $U = (01, 110), V = (00, 11)$, there is no solution

Aug 2017 IIT Bombay

Modified Post’s Correspondence Problem (MPCP)

- Sets $U$ and $V$ are finite and contain the same number of strings
- The strings in $U$ and $V$ are finite and are of varying lengths
- For constructing the new strings using the strings in $U$ and $V$
  ▶ The strings at the same the index of must be used
  ▶ There is no limit on the length of the new string

Indices could repeat without any bound

Aug 2017 IIT Bombay
In a lucky case we may find an algorithm which can determine that 
\( r = 0 \) along every path 
(some \( x \) is never equal to \( y \), 
MPCP instance does not have a solution)

\( r = 1 \) along some path 
(some \( x \) is equal to \( y \), 
MPCP instance has a solution)

Then MPCP is decidable

The tricky part!!

- Asserting that no \( x \) is equal to \( y \) requires us to examine infinitely many \((x, y)\) pairs
- If we keep finding \( x \) and \( y \) that are unequal, how long do we wait to decide that there is no \( x \) that is equal to \( y \)?
- In a lucky case we may find an \( x \) that is equal to \( y \), but there is no guarantee

MPCP is not decidable

\( \Rightarrow \) Constant Propagation is not decidable

- Descending chains consist of sets of pairs \((x, y)\) with \( T \) as \( \emptyset \)
  Since there is no bound on the length of \( x \) and \( y \), the number of these sets is infinite
  \( \Rightarrow \) DCC is violated

Then MPCP is decidable
Is MFP Always Computable?

MFP assignment may not be computable
- if the flow functions are non-monotonic, or
- if some strictly descending chain is not finite

Computability of MFP

- If $f$ is not monotonic, the computation may not converge

\[
\begin{array}{cccccc}
1 & f(x) & f^2(x) & f^3(x) & f^4(x) & \ldots \\
1 & 0 & 1 & 0 & 1 & \ldots \\
\end{array}
\]

\[MoP = x \cap f(x) \cap f^2(x) \cap f^3(x) \cap \ldots = 0\]

- Computing MFP iteratively

\[
\begin{array}{ccc}
1 & -1 & 0 \\
1 & -1 & 0 \\
0 & -1 & 1 \\
\end{array}
\]

MFP does not exist and is not computable

MFP exist and is computable

MFP exists? | No | Yes
---|---|---
MFP computable? | No | No
MoP exists? | No | Yes

Flow functions are monotonic
Strictly descending chains are not finite

MFP exists? | No | Yes
---|---|---
MFP computable? | No | No
MoP exists? | No | Yes
Existence and Computation of the Maximum Fixed Point

If $L$ is a meet semilattice satisfying DCC, $f : L \to L$ is monotonic, then $MFP(f) = f^{k+1}(\top) = f^k(\top)$ such that $f^{i+1}(\top) \neq f^i(\top)$, $j < k$.

Claims being made:
- $\exists k \text{ s.t. } f^{k+1}(\top) = f^k(\top)$
- Since $k$ is finite, $f^k(\top)$ exists and is computable
- $f^k(\top)$ is a fixed point
- $f^k(\top)$ is a the maximum fixed point

The proof depends on:
- The existence of glb for every pair of values in $L$
- Finiteness of strictly descending chains
- Monotonicity of $f$

Fixed Points Computation: Flow Functions Vs. Equations

- Recall that
  
  $$MFP(f) = f^{k+1}(\top) = f^k(\top)$$ such that $f^{i+1}(\top) \neq f^i(\top)$, $j < k$.

  - What is $f$ in the above?
  - Flow function of a block? Which block?

- Our method computes the maximum fixed point of data flow equations!
- What is the relation between the maximum fixed point of data flow equations and the MFP defined above?

Existence and Computation of the Maximum Fixed Point

$$f^{k+1}(\top) = f^k(\top)$$ such that $f^{i+1}(\top) \neq f^i(\top)$, $j < k$.

- $\top \sqsubseteq f(\top) \sqsubseteq f^2(\top) \sqsubseteq f^3(\top) \sqsubseteq \ldots$
- Since strictly descending chains are finite, there must exist $f^k(\top)$ such that $f^{k+1}(\top) = f^k(\top)$ and $f^{i+1}(\top) \neq f^i(\top)$, $j < k$
- If $p$ is a fixed point of $f$ then $p \sqsubseteq f^k(\top)$
- Proof strategy: Induction on $i$ for $f^i(\top)$
  - Basis ($i = 0$): $p \sqsubseteq f^0(\top) = \top$
  - Inductive Hypothesis: Assume that $p \sqsubseteq f^i(\top)$
  - Proof:
    - $f(p) \sqsubseteq f(f^i(\top))$ (f is monotonic)
    - $\Rightarrow p \sqsubseteq f(f^i(\top))(f(p) = p)$
    - $\Rightarrow p \sqsubseteq f^{i+1}(\top)$
- Since this holds for every $p$ that is a fixed point, $f^{k+1}(\top)$ must be the Maximum Fixed Point

Fixed Points Computation: Flow Functions Vs. Equations

- Data flow equations for a CFG with $N$ nodes can be written as

  $$\begin{align*}
  \text{In}_1 &= \text{BI} \\
  \text{Out}_3 &= f_3(\text{In}_1) \\
  \text{In}_2 &= \text{Out}_3 \sqcap \ldots \\
  \text{Out}_2 &= f_2(\text{In}_2) \\
  \ldots \\
  \text{In}_N &= \text{Out}_{N-1} \sqcap \ldots \\
  \text{Out}_N &= f_N(\text{In}_N)
  \end{align*}$$
Data flow equations for a CFG with $N$ nodes can be written as:

\[
\begin{align*}
\text{In}_1 &= f_{\text{in}}(\langle \text{In}_1, \text{Out}_1, \ldots, \text{In}_N, \text{Out}_N \rangle) \\
\text{Out}_1 &= f_{\text{out}}(\langle \text{In}_1, \text{Out}_1, \ldots, \text{In}_N, \text{Out}_N \rangle) \\
\text{In}_2 &= f_{\text{in}}(\langle \text{In}_1, \text{Out}_1, \ldots, \text{In}_N, \text{Out}_N \rangle) \\
\text{Out}_2 &= f_{\text{out}}(\langle \text{In}_1, \text{Out}_1, \ldots, \text{In}_N, \text{Out}_N \rangle) \\
&\ldots
\text{In}_N &= f_{\text{in}}(\langle \text{In}_1, \text{Out}_1, \ldots, \text{In}_N, \text{Out}_N \rangle) \\
\text{Out}_N &= f_{\text{out}}(\langle \text{In}_1, \text{Out}_1, \ldots, \text{In}_N, \text{Out}_N \rangle)
\end{align*}
\]

where each flow function is of the form $L \times L \times \ldots \times L \rightarrow L$.

We compute the fixed points of function $F$ defined above:

\[
\begin{align*}
\mathcal{X} &= \{ f_{\text{in}}(\mathcal{X}), f_{\text{out}}(\mathcal{X}), \ldots, f_{\text{in}}(\mathcal{X}), f_{\text{out}}(\mathcal{X}) \}
\end{align*}
\]

where $\mathcal{X} = \langle \text{In}_1, \text{Out}_1, \ldots, \text{In}_N, \text{Out}_N \rangle$. We compute the fixed points of function $\mathcal{F}$ defined above:

\[
\begin{align*}
\mathcal{F}(\mathcal{X}) &= \{ f_{\text{in}}(\mathcal{X}), f_{\text{out}}(\mathcal{X}), \ldots, f_{\text{in}}(\mathcal{X}), f_{\text{out}}(\mathcal{X}) \}
\end{align*}
\]
**An Instance of Available Expressions Analysis**

- Conventional data flow equations

  \[ \begin{align*}
  \text{In}_1 &= 00 \\
  \text{In}_2 &= \text{Out}_1 \cap \text{Out}_2 \\
  \text{Out}_1 &= 11 \\
  \text{Out}_2 &= \text{In}_2
  \end{align*} \]

- Data Flow Equation \( X = \mathcal{F}(X) \) is

  \[ \mathcal{F}(\langle \text{In}_1, \text{Out}_1, \text{In}_2, \text{Out}_2 \rangle) = \langle 00, 11, \text{Out}_1 \cap \text{Out}_2, \text{In}_2 \rangle \]

- The maximum fixed point assignment is

  \[ \mathcal{F}(\langle 11, 11, 11, 11 \rangle) = \langle 00, 11, 11, 11 \rangle \]

- The minimum fixed point assignment is

  \[ \mathcal{F}(\langle 00, 00, 00, 00 \rangle) = \langle 00, 11, 00, 00 \rangle \]

---

**Safety of FP Assignment: \( \text{FP} \sqsubseteq \text{MoP} \)**

- \( \text{MoP}(v) = \bigcap_{\rho \in \text{Paths}(v)} f_\rho(BI) \)

- Proof Obligation: \( \forall \rho_v \text{ FP}(v) \sqsubseteq f_\rho_v(BI) \)

- Claim 1: \( \forall u \rightarrow v, \text{FP}(v) \sqsubseteq f_{u \rightarrow v}(\text{FP}(u)) \)

- Proof Outline: Induction on path length

  Base case: Path of length 0

  \[ \text{FP}(	ext{Entry}) = \text{MoP(Entry)} = BI \]

  Inductive hypothesis: Assume it holds for paths consisting of \( k \) edges (say at \( u \))

  \[ \begin{align*}
  \text{FP}(u) &\sqsubseteq f_{\rho_u}(BI) \quad \text{(Inductive hypothesis)} \\
  \text{FP}(v) &\sqsubseteq f_{u \rightarrow v}(\text{FP}(u)) \quad \text{(Claim 1)} \\
  \Rightarrow \text{FP}(v) &\sqsubseteq f_{u \rightarrow v}(f_{\rho_u}(BI)) \\
  \Rightarrow \text{FP}(v) &\sqsubseteq f_{\rho_v}(BI)
  \end{align*} \]

  This holds for every FP an hence for MFP also

---

**Theoretical Abstractions: A Summary**

Necessary and sufficient conditions for designing a data flow framework

- A meet semilattice satisfying \( \text{dcc} \)
  - Meet: commutative, associative, and idempotent
  - Partial order: reflexive, transitive, and antisymmetric
  - Existence of \( \bot \)

- A function space
  - Existence of the identity function
  - Closure under composition
  - Monotonic functions
Part 9
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

- **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
- **$T_1$, $T_2$ Based Analysis.** Uses graph parsing
Classification of Edges in a Graph

Graph $G$  
A depth first spanning tree of $G$

Back edges  
Forward edges

For data flow analysis, we club tree, forward, and cross edges into 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 $G$  
$G'$ obtained after removing back edges of $G$

Practically, RPO of a graph is determined by a depth search over the graph

• 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. $ln_0 = BI$
2. for all $j \neq 0$
3. \hspace{1em} $ln_j = \top$
4. change = true
5. while change do
6. \hspace{1em} change = false
7. \hspace{1.5em} for $j = 1$ to $N - 1$
8. \hspace{2em} \hspace{1em} temp = $\bigwedge_{p \in \text{pred}(j)} f_p(ln_p)$
9. \hspace{2em} \hspace{1em} if temp $\neq ln_j$ then
10. \hspace{3em} ln$_j$ = temp
11. \hspace{3em} change = true
12. end while

- Computation of Out$_j$ has been left implicit
- Works fine for unidirectional frameworks
- $\top$ is the identity of $\cap$ (line 3)
- Reverse postorder (rpo) traversal for efficiency (line 7)
- rpo traversal AND no loops $\Rightarrow$ no need of initialization

• 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
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;
    }
}
```

Availability of $a+b$ in iteration #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;
    if (i < m)
    {
        j=0;
        while(j<n)
        {
            a=i+j;
            j=j+1;
        }
        i=i+1;
    }
}
```

Availability of $a+b$ in iteration #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;
    }
}
```

Availability of $a+b$ in iteration #2

```
c = a + b
i = 0
if (i < m)
j = 0
if (j < n)
Availability of $a+b$
in iteration #3
```

3 + 1 iterations for available expressions analysis

Aug 2017 IIT Bombay
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

Pairs of Out,In Values

| Initia-
<table>
<thead>
<tr>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,1</td>
<td>0,1</td>
<td>0,1</td>
<td>0,1</td>
<td>0,1</td>
</tr>
<tr>
<td>12</td>
<td>0,0</td>
<td>0,0</td>
<td>0,0</td>
<td>0,0</td>
</tr>
<tr>
<td>11</td>
<td>1,1</td>
<td>0,1</td>
<td>0,1</td>
<td>0,1</td>
</tr>
<tr>
<td>10</td>
<td>0,1</td>
<td>1,0</td>
<td>1,0</td>
<td>1,0</td>
</tr>
<tr>
<td>9</td>
<td>1,1</td>
<td>0,0</td>
<td>0,0</td>
<td>0,0</td>
</tr>
<tr>
<td>8</td>
<td>1,1</td>
<td>0,0</td>
<td>0,0</td>
<td>0,0</td>
</tr>
<tr>
<td>7</td>
<td>1,1</td>
<td>0,0</td>
<td>0,0</td>
<td>0,0</td>
</tr>
<tr>
<td>6</td>
<td>1,1</td>
<td>1,0</td>
<td>1,0</td>
<td>1,0</td>
</tr>
<tr>
<td>5</td>
<td>1,1</td>
<td>0,0</td>
<td>0,0</td>
<td>0,0</td>
</tr>
<tr>
<td>4</td>
<td>1,1</td>
<td>0,0</td>
<td>0,0</td>
<td>0,0</td>
</tr>
<tr>
<td>3</td>
<td>1,1</td>
<td>0,0</td>
<td>0,0</td>
<td>0,0</td>
</tr>
<tr>
<td>2</td>
<td>1,1</td>
<td>0,0</td>
<td>0,0</td>
<td>0,0</td>
</tr>
<tr>
<td>1</td>
<td>1,1</td>
<td>0,0</td>
<td>0,0</td>
<td>0,0</td>
</tr>
</tbody>
</table>

An Example of Information Flow in Our PRE Analysis

- $Pav_{In_6}$ 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)
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

- Forward Node Flow Function
- Backward Node Flow Function
- Bidirectional Node Flow Function

General Data Flow Equations

\[
\begin{align*}
In_n &= \begin{cases} 
BI_{\text{Start}} \cap f^b_n(Out_n) & n = \text{Start} \\
\left( \bigcap_{m \in \text{pred}(n)} f^f_{m \rightarrow n}(Out_m) \right) \cap f^b_n(Out_n) & \text{otherwise}
\end{cases} \\
Out_n &= \begin{cases} 
BI_{\text{End}} \cap f^f_n(In_n) & n = \text{End} \\
\left( \bigcap_{m \in \text{succ}(n)} f^b_{m \rightarrow n}(In_m) \right) \cap f^f_n(In_n) & \text{otherwise}
\end{cases}
\end{align*}
\]

- Edge flow functions are typically identity
  \( \forall x \in L, f(x) = x \)
- If particular flows are absent, the corresponding flow functions are
  \( \forall x \in L, f(x) = \top \)
Information Flow Paths in PRE

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

Complexity of Worklist Algorithms for Bit Vector Frameworks

- Assume $n$ nodes and $r$ entities
- Total number of data flow values = $2 \cdot n \cdot r$
- A data flow value can change at most once
- Complexity is $O(n \cdot r)$
- Must be same for both unidirectional and bidirectional frameworks
  (Number of data flow values does not change!)

Lacuna with Older Estimates of PRE Complexity

- Lacuna with PRE : Complexity
  - $r$ is typically $O(n)$
  - Assuming that at most one data flow value changes in one traversal
  - Worst case number of traversals = $O(n^2)$
- Practical graphs may have upto 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
- 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

- 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

- Max. Incompatible edge traversals
  - $= \text{Width of the graph} = 4$

- Maximum number of traversals
  - $= 1 + \text{Max. incompatible edge traversals} = 5$

### 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 for a unidirectional bit vector framework,
  - Width $\leq$ 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

\[
\begin{align*}
c &= a + b \\
i &= 0 \quad \text{n}_1
\end{align*}
\]

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

Work List Based Iterative Algorithm

Directly traverses information flow paths

```
1 \( L_{n_0} = BI \)
2 for all \( j \neq 0 \) do
3 { \( L_{n_j} = T \)
4    Add \( j \) to LIST
5 }
6 while LIST is not empty do
7 { Let \( j \) be the first node in LIST. Remove it from LIST
8    \( temp = \prod_{p \in \text{pred}(j)} f_p(L_{n_p}) \)
9    if \( temp \neq L_{n_j} \) then
10       \( L_{n_j} = temp \)
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</th>
<th>Node</th>
<th>Remaining work list</th>
<th>Out</th>
<th>DFV</th>
<th>Change?</th>
<th>Node</th>
<th>Resulting work list</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>n₁</td>
<td>n₂, n₃, n₄, n₅, n₆, n₇</td>
<td></td>
<td></td>
<td>No</td>
<td></td>
<td>n₂, n₃, n₄, n₅, n₆, n₇</td>
</tr>
<tr>
<td>2</td>
<td>n₂</td>
<td>n₃, n₄, n₅, n₆, n₇</td>
<td></td>
<td></td>
<td>No</td>
<td></td>
<td>n₃, n₄, n₅, n₆, n₇</td>
</tr>
<tr>
<td>3</td>
<td>n₃</td>
<td>n₄, n₅, n₆, n₇</td>
<td></td>
<td></td>
<td>No</td>
<td></td>
<td>n₄, n₅, n₆, n₇</td>
</tr>
<tr>
<td>4</td>
<td>n₄</td>
<td>n₅, n₆, n₇</td>
<td></td>
<td></td>
<td>No</td>
<td></td>
<td>n₅, n₆, n₇</td>
</tr>
<tr>
<td>5</td>
<td>n₅</td>
<td>n₆, n₇, n₄</td>
<td></td>
<td>Yes</td>
<td>n₄</td>
<td>n₅, n₆, n₇, n₄</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>n₆</td>
<td>n₇, n₄, n₄</td>
<td></td>
<td>Yes</td>
<td>n₄</td>
<td>n₇, n₄, n₄</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>n₇</td>
<td>n₄, n₄</td>
<td></td>
<td>Yes</td>
<td>n₄</td>
<td>n₄</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>n₄</td>
<td>n₅, n₆</td>
<td></td>
<td>Yes</td>
<td>n₅, n₆</td>
<td>n₅, n₆</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>n₅</td>
<td>n₆</td>
<td></td>
<td>No</td>
<td>n₆</td>
<td>n₆</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>n₆</td>
<td>n₂</td>
<td></td>
<td>Yes</td>
<td>n₂</td>
<td>n₂</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>n₂</td>
<td>n₃, n₇</td>
<td></td>
<td>Yes</td>
<td>n₃, n₇</td>
<td>n₃, n₇</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>n₃</td>
<td>n₇, n₄</td>
<td></td>
<td>Yes</td>
<td>n₄</td>
<td>n₇, n₄</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>n₇</td>
<td>n₄</td>
<td></td>
<td>Yes</td>
<td>n₄</td>
<td>n₄</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>n₄</td>
<td>Empty</td>
<td></td>
<td>No</td>
<td>Empty</td>
<td>Empty</td>
<td></td>
</tr>
</tbody>
</table>

Part 10

Precise Modelling of General Flows
Complexity of Constant Propagation?

\begin{align*}
1 \quad a &= b + 1 \\
2 \quad a &= b + 1 \\
3 \quad b &= c + 1 \\
4 \quad c &= d + 1 \\
5 \quad d &= 2 \\
\end{align*}

Iteration #1

\begin{align*}
1 \quad a &= b + 1 \\
2 \quad a &= b + 1 \\
3 \quad b &= c + 1 \\
4 \quad c &= d + 1 \\
5 \quad d &= 2 \\
\end{align*}

Iteration #2

\begin{align*}
1 \quad a &= b + 1 \\
2 \quad a &= b + 1 \\
3 \quad b &= c + 1 \\
4 \quad c &= d + 1 \\
5 \quad d &= 2 \\
\end{align*}

Iteration #3

\begin{align*}
1 \quad a &= b + 1 \\
2 \quad a &= b + 1 \\
3 \quad b &= c + 1 \\
4 \quad c &= d + 1 \\
5 \quad d &= 2 \\
\end{align*}

Iteration #4

\begin{align*}
1 \quad a &= b + 1 \\
2 \quad a &= b + 1 \\
3 \quad b &= c + 1 \\
4 \quad c &= d + 1 \\
5 \quad d &= 2 \\
\end{align*}

Part 11

Extra Topics

Tarski’s Fixed Point Theorem

Given monotonic \( f : L \to L \) where \( L \) is a complete lattice

Define

\begin{align*}
\text{\( p \) is a fixed point of \( f \):} & \quad \text{Fix(}f\text{) = \{ \text{\( p \mid f(}p\text{\) = \( p\)\)}\}} \\
\text{\( f \) is reductive at \( p \):} & \quad \text{Red(}f\text{) = \{ \text{\( p \mid f(}p\text{\) ⊑ \( p\)\)}\}} \\
\text{\( f \) is extensive at \( p \):} & \quad \text{Ext(}f\text{) = \{ \text{\( p \mid f(}p\text{\) ⊒ \( p\)\)}\}}
\end{align*}

Then

\begin{align*}
\text{LFP(}f\text{) =} & \quad \bigwedge \text{Red(}f\text{) \in Fix(}f\text{)} \\
\text{MFP(}f\text{) =} & \quad \bigvee \text{Ext(}f\text{) \in Fix(}f\text{)}
\end{align*}

Guarantees only existence, not computability of fixed points
Examples of Reductive and Extensive Sets

Finite $L$ Monotonic $f : L \to L$

<table>
<thead>
<tr>
<th>$\top$</th>
<th>$\bot$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1$</td>
<td>$v_1$</td>
</tr>
<tr>
<td>$v_2$</td>
<td>$v_2$</td>
</tr>
<tr>
<td>$v_3$</td>
<td>$v_3$</td>
</tr>
<tr>
<td>$v_4$</td>
<td>$v_4$</td>
</tr>
</tbody>
</table>

$Red(f) = \{\top, v_3, v_4, \bot\}$  
$Ext(f) = \{\top, v_1, v_2, \bot\}$  
$Fix(f) = Red(f) \cap Ext(f)$  
$Fix(f) = \{\top, \bot\}$  
$MFP(f) = lub(Ext(f))$  
$MFP(f) = lub(Fix(f))$  
$LFP(f) = glb(Red(f))$  
$LFP(f) = glb(Fix(f))$  

Existence of MFP: Proof of Tarski’s Fixed Point Theorem

1. Claim 1: Let $X \subseteq L$. 
   $\forall x \in X, \ p \sqsupseteq x \Rightarrow p \sqsupseteq \bigsqcup(X)$. 
2. In the following we use $Ext(f)$ as $X$

3. $\forall p \in Ext(f), \ hi \sqsupseteq p$
4. $hi \sqsupseteq p \Rightarrow f(hi) \sqsupseteq f(p) \sqsupseteq p$ (monotonicity) 
   $\Rightarrow f(hi) \sqsupseteq hi$ (claim 1)
Existence of MFP: Proof of Tarski’s Fixed Point Theorem

1. Claim 1: Let $X \subseteq L$.
   \[ \forall x \in X, p \supseteq x \Rightarrow p \supseteq \bigsqcup(X). \]
2. In the following we use $\text{Ext}(f)$ as $X$.
3. $\forall p \in \text{Ext}(f), \ hi \supseteq p$
4. $hi \supseteq p \Rightarrow f(hi) \supseteq f(p) \supseteq p$ (monotonicity)
   \[ \Rightarrow f(hi) \supseteq hi \quad \text{(claim 1)} \]
5. $f$ is extensive at $hi$ also: $hi \in \text{Ext}(f)$

Existence and Computation of the Maximum Fixed Point

• For monotonic $f : L \rightarrow L$
  
  ▶ Existence: $MFP(f) = \bigsqcup \text{Ext}(f) \in \text{Fix}(f)$
  
  Requires $L$ to be complete

  ▶ Computation: $MFP(f) = f^{k+1}(\top) = f^k(\top)$ such that
    $f^{j+1}(\top) \neq f^j(\top), j < k$.
  
  Requires all strictly descending chains to be finite

• Finite strictly descending and ascending chains
  \[ \Rightarrow \text{Completeness of lattice} \]

• Completeness of lattice $\nRightarrow$ Finite strictly descending chains

• $\Rightarrow$ Even if MFP exists, it may not be reachable unless all strictly descending chains are finite

Framework Properties Influencing Complexity

Depends on the loop closure properties of the framework
Complexity of Round Robin Iterative Algorithm

- Unidirectional rapid frameworks

<table>
<thead>
<tr>
<th>Task</th>
<th>Number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Irreducible $G$</td>
</tr>
<tr>
<td>Initialisation</td>
<td>1</td>
</tr>
<tr>
<td>Convergence (until change remains true)</td>
<td>$d(G, T)$ + 1</td>
</tr>
<tr>
<td>Verifying convergence (change becomes false)</td>
<td>1</td>
</tr>
</tbody>
</table>