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 2009

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:


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

What We Have Seen So Far . . .

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

Data Flow Values for Constant Propagation

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

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

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

An Introduction to Constant Propagation

Summary Values

<table>
<thead>
<tr>
<th>$n_1$</th>
<th>$a = 1$</th>
<th>$b = 2$</th>
<th>$c = a + b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_2$</td>
<td>$c = a + b$</td>
<td>$d = a * b$</td>
<td></td>
</tr>
<tr>
<td>$n_3$</td>
<td>$d = c - 1$</td>
<td>$a = 2$</td>
<td>$b = 1$</td>
</tr>
</tbody>
</table>

Execution Sequence

$\langle a, b, c, d \rangle$
Confluence Operation for Constant Propagation

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

<table>
<thead>
<tr>
<th>( \sqcap )</th>
<th>( \langle a, ud \rangle )</th>
<th>( \langle a, nc \rangle )</th>
<th>( \langle a, c_1 \rangle )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \langle a, ud \rangle )</td>
<td>( \langle a, ud \rangle )</td>
<td>( \langle a, nc \rangle )</td>
<td>( \langle a, c_1 \rangle )</td>
</tr>
<tr>
<td>( \langle a, nc \rangle )</td>
<td>( \langle a, nc \rangle )</td>
<td>( \langle a, nc \rangle )</td>
<td>( \langle a, nc \rangle )</td>
</tr>
<tr>
<td>( \langle a, c_2 \rangle )</td>
<td>( \langle a, c_2 \rangle )</td>
<td>( \langle a, nc \rangle )</td>
<td>( \langle a, c_1 \rangle )</td>
</tr>
</tbody>
</table>

- This is neither \( \sqcap \) nor \( \sqcup \).

What are its properties?

Flow Functions for Constant Propagation

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

<table>
<thead>
<tr>
<th>mult</th>
<th>( \langle a_1, ud \rangle )</th>
<th>( \langle a_1, nc \rangle )</th>
<th>( \langle a_1, c_1 \rangle )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \langle a_2, ud \rangle )</td>
<td>( \langle r, ud \rangle )</td>
<td>( \langle r, nc \rangle )</td>
<td>( \langle r, ud \rangle )</td>
</tr>
<tr>
<td>( \langle a_2, nc \rangle )</td>
<td>( \langle r, nc \rangle )</td>
<td>( \langle r, nc \rangle )</td>
<td>( \langle r, nc \rangle )</td>
</tr>
<tr>
<td>( \langle a_2, c_2 \rangle )</td>
<td>( \langle r, ud \rangle )</td>
<td>( \langle r, nc \rangle )</td>
<td>( \langle r, (c_1 \ast c_2) \rangle )</td>
</tr>
</tbody>
</table>

- This cannot be expressed in the form

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

where \( Gen_n \) and \( Kill_n \) are constant effects of block \( n \).

Issues in Data Flow Analysis

- Representation
- Lattice
- Partial Order, Top, Bottom
- Merger
- Commutativity, Associativity, Idempotence
- Flow Functions
- Monotonicity, Distributivity, \( k \)-Boundedness, Separability

Existence
- Safety
- Precision
- Complexity
- Convergence
- Initialisation

Part 3

A Digression on Lattices
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 \( y \subseteq u \)

Partially Ordered Sets

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

Subsets \( \{4, 9\} \) and \( \{2, 3\} \) do not have an upper bound in the set

Lattices

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

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

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

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

  Example:
  Lattice \( \mathbb{Z} \) of integers under \( \leq \) relation with \( \infty \) and \( -\infty \).
    - \( \infty \) is the top element denoted \( \top \): \( \forall i \in \mathbb{Z}, i \leq \top \).
    - \( -\infty \) is the bottom element denoted \( \bot \): \( \forall i \in \mathbb{Z}, \bot \leq i \).

\[ \mathbb{Z} \cup \{ \infty, -\infty \} \text{ is a Complete Lattice} \]

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

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

Finite Lattices are Complete

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

<table>
<thead>
<tr>
<th>Available Expressions Analysis</th>
<th>Partially Available Expressions Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>( { i } )</td>
<td>( { } )</td>
</tr>
<tr>
<td>( { e_1, e_2 } )</td>
<td>( { e_1 } )</td>
</tr>
<tr>
<td>( { e_1, e_3 } )</td>
<td>( { e_2 } )</td>
</tr>
<tr>
<td>( { e_2, e_3 } )</td>
<td>( { e_3 } )</td>
</tr>
</tbody>
</table>

Lattice for May-Must Analysis

- There is no \( \top \) among the natural values

- An artificial \( \top \) can be added
  However, a lub may not exist for arbitrary sets

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

- Let \( A \) be all finite subsets of \( \mathbb{Z} \).
- The poset \((A \cup \{\mathbb{Z}\}, \subseteq)\) is a bounded lattice with \( \top = \mathbb{Z} \) and \( \bot = \emptyset \).
- Does the set of all sets that do not contain a given number (say 1) has a glb in \( A \cup \{\mathbb{Z}\} \)?
- The union of all finite sets that do not contain 1 is an infinite set that does not contain 1. This set is not contained in \( A \cup \{\mathbb{Z}\} \).

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** with \( \bot \) element
- **Meet Semilattices** satisfying dcc
- **Join Semilattices** with \( \top \) element
- **Join Semilattices** satisfying acc
- **Complete lattices** with dcc and acc

\[ \text{dcc: descending chain condition} \]
\[ \text{acc: ascending chain condition} \]

**Operations on Lattices**

- **Meet (\( \sqcap \)) and Join (\( \sqcup \))**
  \[ z = x \sqcap y \Rightarrow z \sqsubseteq x \land z \sqsubseteq y \]
  \[ z = x \sqcup y \Rightarrow z \sqsupseteq x \land z \sqsupseteq y \]

\[ x \sqcap y = \gcd(x, y) \]
\[ x \sqcup y = \text{lcm}(x, y) \]

**Cartesian Product of Lattices**

\[ (L_A, \sqsubseteq_A, \sqcap_A, \sqcup_A) \times (L_N, \sqsubseteq_N, \sqcap_N, \sqcup_N) = (L_C, \sqsubseteq_C, \sqcap_C, \sqcup_C) \]

\[ \langle x_1, y_1 \rangle \sqsubseteq_C \langle x_2, y_2 \rangle \Leftrightarrow x_1 \sqsubseteq_N x_2 \land y_1 \sqsubseteq_A y_2 \]
\[ \langle x_1, y_1 \rangle \sqcap_C \langle x_2, y_2 \rangle = \langle x_1 \sqcap_N x_2, y_1 \sqcap_A y_2 \rangle \]
\[ \langle x_1, y_1 \rangle \sqcup_C \langle x_2, y_2 \rangle = \langle x_1 \sqcup_N x_2, y_1 \sqcup_A y_2 \rangle \]
### The Set of Data Flow Values

Meet semilattices satisfying the descending chain condition
- $\text{glb}$ must exist for all non-empty finite subsets
- $\bot$ must exist

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

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

```
Set View of the Lattice

\{e_1, e_2, e_3\} \rightarrow \{e_1, e_2\} \rightarrow \{e_1\} \rightarrow \emptyset
{e_1, e_2\} \rightarrow \{e_1, e_3\} \rightarrow \{e_1\} \rightarrow \emptyset
{e_2, e_3\} \rightarrow \{e_2\} \rightarrow \emptyset

```

Bit Vector View

```
000
001
010
100
101
110
111

```

### The Concept of Approximation

- $x$ approximates $y$ iff $x$ can be used in place of $y$ without causing any problems.
- Validity of approximation is context specific
  - $x$ may be approximated by $y$ in one context and by $z$ in another
    - Earnings: Rs. 1050 can be safely approximated by Rs. 1000.
    - Expenses: Rs. 1050 can be safely approximated by Rs. 1100.
Two Important Objectives in Data Flow Analysis

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

Context Determines the Validity of Approximations

May prohibit correct optimization
May enable wrong optimization

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

Partial Order Captures Approximation

- ≱ captures valid approximations for safety
  \[ x \sqsubseteq y \Rightarrow x \text{ is weaker than } y \]
  - The data flow information represented by x can be safely used in place of the data flow information represented by y
  - It may be imprecise, though.
- ⊑ captures valid approximations for exhaustiveness
  \[ x \sqsubseteq y \Rightarrow x \text{ is stronger than } y \]
  - The data flow information represented by x contains every value contained in the data flow information represented by y
  - It may be unsafe, though.

We want most exhaustive information which is also safe.
Setting Up Lattices

Available Expressions Analysis

Live Variables Analysis

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

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

\[ \{ v_1, v_2, v_3 \} \]

\[ \{ v_1, v_2, v_3 \} \]

\[ \bigcap \text{ is } \bigcap \]

\[ \bigcap \text{ is } \bigcup \]

Partial Order Relation

Reflexive \[ x \sqsubseteq x \]

If \( x \) 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 \[ x \sqsubseteq y, y \sqsubseteq x \Leftrightarrow x = y \]

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

- \( x \cap y \) 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 \[ x \cap y = y \cap x \]
  
  The order in which the data flow information is merged, does not matter

- Associative \[ x \cap (y \cap z) = (x \cap y) \cap z \]
  
  Allow n-ary merging without any restriction on the order

- Idempotent \[ x \cap x = x \]
  
  No loss of information if \( x \) is merged with itself

- \( x \cap \top = x \) (ensures exhaustiveness)

- \( x \cap \bot = \bot \) (ensures safety)

More on Lattices in Data Flow Analysis

\[ L = \text{Lattice for all expressions} \]

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

\[ \begin{array}{ccc}
111 & 11 & 111 \\
110 & 101 & 011 \\
100 & 010 & 001 \\
000 & & \\
\end{array} \]

\[ \begin{array}{ccc}
\text{(Expression } e \text{ is available)}
\end{array} \]

\[ \begin{array}{ccc}
1 \text{ or } \{ e \} \\
\Downarrow \\
0 \text{ or } \emptyset \\
\end{array} \]

\[ \begin{array}{ccc}
\text{(Expressions } e \text{ is not available)}
\end{array} \]

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

- \[ L = \hat{L} \times \hat{L} \times \hat{L} \text{ and } x = (\hat{x}_1, \hat{x}_2, \hat{x}_3) \in L \text{ where } \hat{x}_i \in \hat{L} \]
- \[ \sqsubseteq = \hat{\sqsubseteq} \times \hat{\sqsubseteq} \times \hat{\sqsubseteq} \text{ and } \sqcap = \hat{\sqcap} \times \hat{\sqcap} \times \hat{\sqcap} \]
- \[ \top = \hat{\top} \times \hat{\top} \times \hat{\top} \text{ and } \bot = \hat{\bot} \times \hat{\bot} \times \hat{\bot} \]
Component Lattice for Data Flow Information Represented By Bit Vectors

\[
\begin{array}{c}
\hat{\top} & \hat{\top} \\
1 & 0 \\
\equiv & \equiv \\
0 & 1 \\
\bot & \bot
\end{array}
\]

\(\cap\) is \(\cap\) or Boolean AND

\(\lor\) is \(\lor\) or Boolean OR

- Component Lattice for Integer Constant Propagation

\[
\begin{array}{c}
\hat{\top} & \hat{\top} \\
\text{undef or ud} & \\
-\infty & \cdots & -1 & 0 & 1 & 2 & \cdots & \infty \\
\hat{\bot} & \\
\text{nonconst or nc}
\end{array}
\]

Overall lattice \(L\) is the product of \(\hat{L}\) for all variables.

- Component Lattice for May Points-To Analysis

Relation between pointer variables and locations in the memory.

- 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)

```
unknown
(Bits 11)

must-be-available
(Bits 10)

is-not-available
(Bits 01)

may-be-available
(Bits 00)
```

Can also be implemented as a product of 1-0 and 0-1 lattice with AND for the first bit and OR for the second bit.

- What approximation of safety does this lattice capture?
  Uncertain information (= no optimization) is safer than definite information.

General Lattice for May-Must Analysis

![Lattice Diagram]

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 \( f : L \mapsto L \) 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
    \[ x = \bigcap_{1 \leq i \leq m} f_i(B) \]

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

- 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

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

• Merging at intermediate points in shared segments of paths does not lead to imprecision.

Monotonicity and Distributivity

Monotonic and Distributive

Monotonic but not Distributive

Distributivity of Bit Vector Frameworks

\[
\begin{align*}
  f(x) &= \text{Gen} \cup (x - \text{Kill}) \\
  f(y) &= \text{Gen} \cup (y - \text{Kill}) \\
  f(x \cup y) &= \text{Gen} \cup ((x \cup y) - \text{Kill}) \\
  &= \text{Gen} \cup ((x - \text{Kill}) \cup (y - \text{Kill})) \\
  &= (\text{Gen} \cup (x - \text{Kill}) \cup \text{Gen} \cup (y - \text{Kill})) \\
  &= f(x) \cup f(y) \\
  f(x \cap y) &= \text{Gen} \cup ((x \cap y) - \text{Kill}) \\
  &= \text{Gen} \cup ((x - \text{Kill}) \cap (y - \text{Kill})) \\
  &= (\text{Gen} \cup (x - \text{Kill}) \cap \text{Gen} \cup (y - \text{Kill})) \\
  &= f(x) \cap f(y)
\end{align*}
\]

Non-Distributivity of Constant Propagation

\[
\begin{align*}
  x &= \langle 1, 2, 3, ud \rangle \ (\text{Along Out}_{n_1} \rightarrow \text{In}_{n_2}) \\
  y &= \langle 2, 1, 3, 2 \rangle \ (\text{Along Out}_{n_2} \rightarrow \text{In}_{n_3}) \\
  \text{Function application before merging} \\
  f(x) \cap f(y) &= f(\langle 1, 2, 3, ud \rangle) \cap f(\langle 2, 1, 3, 2 \rangle) \\
  &= \langle 1, 2, 3, 2 \rangle \cap \langle 2, 1, 3, 2 \rangle \\
  &= \langle \bot, \bot, 3, 2 \rangle \\
  \text{Function application after merging} \\
  f(x \cap y) &= f(\langle 1, 2, 3, ud \rangle \cap \langle 2, 1, 3, 2 \rangle) \\
  &= f(\langle 1, \bot, 3, 2 \rangle) \\
  &= \langle \bot, \bot, \bot, \bot \rangle \\
  f(x \cap y) \sqsubseteq f(x) \cap f(y)
\end{align*}
\]
Why is Constant Propagation Non-Distributive?

Possible combinations due to merging

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

- Correct combination.

Possible combinations due to merging

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

- Wrong combination.
- Mutually exclusive information.
- No execution path along which this information holds.
Part 6

Solutions of Data Flow Analysis

- MoP and MFP assignments and their relationship
- Existence of MoP solution
- Existence and Computability of MFP solution
- Safety of MFP solution

Meet Over Paths (MoP) Assignment

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

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

- \(f_{\rho}\) 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.

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
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

All possible assignments
∀i, In_i = Out_i = \top
Meet Over Paths Assignment

All safe assignments
Maximum Fixed Point

All fixed point solutions
Least Fixed Point
∀i, In_i = Out_i = \bot

Possible Assignments as Solutions of Data Flow Analyses

Existence of an MoP Assignment

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

- If all paths reaching p are acyclic, then existence of solution trivially follows from the definition of the function space.
- If cyclic paths also reach p, then there are an infinite number of unbounded paths.
  ⇒ Need to define loop closures.
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.

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

- \( f^* \) is called the loop closure of \( f \).

Loop Closures in Bit Vector Frameworks

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

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

- Intuition: Since Gen and 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.

Bounded Loop Closures May not be Computable

- If \( f \) is not monotonic, the computation may not converge

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

\[
 f^*(x) = x \cap f(x) = 0 \quad \text{Solution exists}
\]

- Iteratively computing the solution

The values in the loop keep changing

Boundedness of \( f \) requires the existence of some \( k \) such that

\[
 f^*(x) = x \cap f(x) \cap f^2(x) \cap \ldots \cap f^{k-1}(x)
\]

Given, monotonic \( f \), loop closures are bounded because of any of the following:

- \( x \subseteq f(x) \). All applications of \( f \) can be ignored

- \( x \nsubseteq f(x) \). In this case, \( x, f(x), f^2(x), \ldots \) follow a descending chain. If descending chains are bounded, loop closures are bounded.

- \( x \) and \( f(x) \) are incomparable. In this case \( \prod_{i=0}^{j} f^i(x) \) follows a strictly descending chain. If descending chains are bounded, loop closures are bounded.
Existence and Computation of the Maximum Fixed Point

For monotonic $f : L \mapsto L$, if all descending chains are finite, then

$$MFP(f) = f^{k+1}(\top) = f^k(\top)$$

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

- If $p$ is a fixed point of $f$ then $f^k(\top) \supseteq p$.
- Proof strategy: Induction on $i$ for $f^i(\top)$
  - Basis ($i = 0$): $f^0(\top) = \top \supseteq p$.
  - Inductive Hypothesis: Assume that $f^i(\top) \supseteq p$.
  - Proof:
    $$f(f^i(\top)) \supseteq f(p) \quad (f \text{ is monotonic})$$
    $$\Rightarrow f(f^i(\top)) \supseteq p \quad (f(p) = p)$$
    $$\Rightarrow f^{i+1}(\top) \supseteq p$$
  - $\Rightarrow f^{k+1}(\top)$ is the MFP.

Fixed Points Computation: Flow Functions Vs. Equations

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

$$\langle In_1, Out_1, \ldots, In_N, Out_N \rangle = f_{in}(\langle In_1, Out_1, \ldots, In_N, Out_N \rangle),$$

$$\langle Out_1, \ldots, Out_N \rangle = f_{out}(\langle In_1, Out_1, \ldots, In_N, Out_N \rangle),$$

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

Recall that

$$MFP(f) = f^{k+1}(\top) = f^k(\top)$$

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

- 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?
Fixed Points Computation: Flow Functions Vs. Equations

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

\[
\mathcal{X} = \langle f_{\text{in}}(\mathcal{X}), f_{\text{out}}(\mathcal{X}), \ldots, f_{\text{in}}(\mathcal{X}), f_{\text{out}}(\mathcal{X}) \rangle
\]

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 \( F \) defined above

Safety of MFP Assignment: MFP \( \sqsubseteq \) MoP

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

- Proof Obligation: \( \forall_{\rho v} \text{MFP}(v) \sqsubseteq f_{\rho v}(B) \)

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

- Proof Outline: Induction on path length

  Base case: Path of length 0.

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

  \[
  \text{MFP}(u) \sqsubseteq f_{\rho u}(B) \quad \text{(Inductive hypothesis)}
  \]

  \[
  \text{MFP}(v) \sqsubseteq f_{u \rightarrow v} (f_{\rho u}(B)) \quad \text{(Claim 1)}
  \]

  \[
  \text{MFP}(v) \sqsubseteq f_{\rho v}(B)
  \]

Part 7

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

Aug 2009
IIT Bombay

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
  - Our examples use this method.
- **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.

Aug 2009
IIT Bombay

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.

Aug 2009
IIT Bombay

Round Robin Iterative Algorithm

```
2  In₀ = B₁
3  for all \( j \neq 0 \) do
4      Inₖ = ⊤
5  change = true
6  while change do
7      change = false
8      for \( j = 1 \) to \( N - 1 \) do
9          \{ temp = \( \prod_{p \in \text{pred}(j)} f_p(In_p) \)
10         if temp \neq In_j then
11            \{ In_j = temp
12               change = true
13            \}
14        \}
15    \}
```
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>Initialisation</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 $d(G, T) = 2$

```
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;
    }
    i = i + 1;
  }
}
```

3 + 1 iterations for available expressions analysis

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 $n_5 \to n_2$ and $n_{10} \to n_9$.
- $d(G, T) = 1$
- Actual iterations: $5$

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**
- **Forward Edge Flow Function**
- **Backward Edge Flow Function**

**General Data Flow Equations**

\[
\begin{align*}
\text{In}_n &= \begin{cases} 
B|_{\text{Start}} \cap f^b_n(\text{Out}_n) 
\quad & n = \text{Start} \\
\prod_{m \in \text{pred}(n)} f^f_{m \rightarrow n}(\text{Out}_m) \cap f^b_n(\text{Out}_n) 
\quad & \text{otherwise}
\end{cases} \\
\text{Out}_n &= \begin{cases} 
B|_{\text{End}} \cap f^f_n(\text{In}_n) 
\quad & n = \text{End} \\
\prod_{m \in \text{succ}(n)} f^b_{m \rightarrow n}(\text{In}_m) \cap f^f_n(\text{In}_n) 
\quad & \text{otherwise}
\end{cases}
\end{align*}
\]

- 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**
- **Backward**
- **Bidirectional**

**Information Flow Paths in PRE**

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

- 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: ≤ 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, \( \text{Width} \leq \text{Depth} \)

Width provides a tighter bound

**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 \( n_5 \rightarrow n_4 \rightarrow n_5 \rightarrow n_2 \)

  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

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

- ifp 5 → 4 → 6 is bypassed by the edge 5 → 6

- ifp 6 → 3 → 6 is bypassed by the edge 6 → 7

- ifp 7 → 2 → 8 is bypassed by the edge 7 → 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 for all \( j \neq 0 \) do
3 \{ \( In_j = \top \)
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 \( temp = \bigcap_{p \in \text{pred}(j)} f_p(In_p) \)
9 if \( temp \neq In_j \) then
10 \{ \( In_j = temp \)
11 Add all successors of \( j \) to LIST
12 \}
13 \}
```

Precise Modelling of General Flows

Complexity of Constant Propagation?

Larger Values of Loop Closure Bounds

- Fast Frameworks \( \equiv \) 2-bounded frameworks (e.g., 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

\( f : L \mapsto \hat{L} \) 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

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

\( \hat{h}_2 \)

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

### Non-Separable

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

\( \hat{h}_2 \)

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

Example: All bit vector frameworks

Example: Constant Propagation

Larger Values of Loop Closure Bounds

Composite flow function for the loop is

\( f((v_b, v_c, v_d)) = (v_b + 1, v_c + 1, v_d + 1, 2) \)

\( f \) is not 2-bounded because:

\[ f((\top, \top, \top, \top)) = (\top, \top, \top, 2) \]

\[ f^2((\top, \top, \top, \top)) = (\top, \top, 3, 2) \]

\[ f^3((\top, \top, \top, \top)) = (\top, 4, 3, 2) \]

\[ f^4((\top, \top, \top, \top)) = (5, 4, 3, 2) \]

\[ f^5((\top, \top, \top, \top)) = (5, 4, 3, 2) \]

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

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

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

- The dependent parts take care of dependence across different entities as well as dependence on the value of the argument

- Bit vector frameworks are a special case

\[ \text{DepGen}_n(X) = \text{DepKill}_n(X) = \emptyset \]
Part 9

Extra Topics

Remove 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

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
  \[
  u_{i_1}u_{i_2}\ldots u_{i_m} = v_{i_1}v_{i_2}\ldots v_{i_m}
  \]
• For \( U = (101, 11, 100) \) and \( V = (01, 1, 11001) \) the solution is 2, 3, 2.
  \[
  u_2u_3u_2 = 110011 \\
  v_2v_3v_2 = 110011
  \]
  • For \( U = (1, 10111, 10) \), \( V = (111, 10, 0) \), the solution is 2, 1, 1, 3.

Modified Post’s Correspondence Problem (MPCP)

• The first string in the correspondence relation should be the first string from the \( k \)-tuple.
  \[
  u_1u_{i_1}u_{i_2}\ldots u_{i_m} = v_1v_{i_1}v_{i_2}\ldots v_{i_m}
  \]
• For \( U = (11, 1, 1011) \), \( V = (1, 111, 10, 0) \), the solution is 3, 2, 2, 4.
  \[
  u_1u_3u_2u_4 = 1101111110 \\
  v_1v_3v_2v_4 = 1101111110
  \]
Hecht’s MPCP to Constant Propagation Reduction

Given: An instance of MPCP with $\Sigma = \{0, 1\}$.

$x = "1"; y = "1"

$i = \text{atoi}(x) ; j = \text{atoi}(y)$

$r = 1 / ((i - j)^2 + 1)$

$\Rightarrow \text{MPCP is decidable}$

$\Rightarrow \text{MPCP is not decidable} \Rightarrow \text{Constant Propagation is not decidable}$

Tarski’s Fixed Point Theorem

Given monotonic $f : L \mapsto L$ where $L$ is a complete lattice.

Define

- $p$ is a fixed point of $f$: $\text{Fix}(f) = \{ p \mid f(p) = p \}$
- $f$ is reductive at $p$: $\text{Red}(f) = \{ p \mid f(p) \sqsubseteq p \}$
- $f$ is extensive at $p$: $\text{Ext}(f) = \{ p \mid f(p) \sqsupseteq p \}$

Then

$LFP(f) = \bigsqcap \text{Red}(f) \in \text{Fix}(f)$

$MFP(f) = \bigsqcup \text{Ext}(f) \in \text{Fix}(f)$

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

Finite \( L \)  

Monotonic \( f : L \mapsto L \)

Reductive \( \text{Red}(f) = \{\top, v_3, v_4, \bot\} \)

Extensive \( \text{Ext}(f) = \{\top, v_1, v_2, \bot\} \)

Fixed \( \text{Fix}(f) = \text{Red}(f) \cap \text{Ext}(f) \)

\( \text{MFP}(f) = \text{lub}(\text{Ext}(f)) \)

\( \text{LFP}(f) = \text{glb}(\text{Red}(f)) \)

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

1. Claim 1: Let \( X \subseteq L \).
   \( \forall x \in X, \ p \supseteq x \Rightarrow p \supseteq \bigcup(X) \).

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)
   \( \Rightarrow f(hi) \supseteq hi \) (claim 1)
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 $\text{Ext}(f)$ as $X$.
3. $\forall p \in \text{Ext}(f), hi \sqsupseteq p$
4. $hi \sqsupseteq p$
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: $\text{MFP}(f) = \bigsqcup\text{Ext}(f) \in \text{Fix}(f)$
    Requires $L$ to be complete.
  - Computation: $\text{MFP}(f) = f^{k+1}(\top) = f^k(\top)$ 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
  - Completeness of lattice
- Completeness of lattice $\Rightarrow$ 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

- $k$-Bounded Frameworks
  - Fast Frameworks ($k = 2$)
    $f^2(x) \sqsupseteq f(x)$
  - Rapid Frameworks
    $f^2(x) \sqsupseteq f(x)$
  - Bit Vector Frameworks
    $f^2(x) = f(x)$

- Necessary and sufficient
- Necessary but not sufficient
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>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>

\[*\] Aug 2009 IIT Bombay