General Data Flow Framework

**Goal:** single framework for all data flow problems

Single algorithm, analysis of termination, complexity

**Approach:** Domain of program properties, organized as Lattice

Lattice $\mathbb{L} = (D, \sqcap)$

Domain Meet operator

$\sqcap$ = All sets of defs

Define $X \sqsubseteq Y$ iff $X \sqcap Y = X$

Ex $\subseteq$

$\{X, Y, Z\} = \text{Top}$

$\{X, Y\}$

$\{X, Z\}$

$\{Y, Z\}$

$\{X\}$

$\{Y\}$

$\{Z\}$

$\emptyset = \perp$ Bottom

1

**Lattice Properties**

$\leq$ Partial order over domain which is

- **Transitive:** for all $x, y, z$ if $x \leq y$ and $y \leq z$ then $x \leq z$

- **Reflexive:** for all $x$, $x \leq x$

- **Anti-symmetric:** for all $x, y$ if $x \leq y$ and $y \leq x$ then $x = y$

- **Associative:** $x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z$

- **Unique least upper bound:** $x \sqcap y$

- **Unique greatest (least) element:** $\top$, $\perp$

- **Height of lattice:** length of longest path from top to bottom

2
General Data Flow Framework

**Flow Graph** $\mathcal{F}$  Ex CFG, Reverse CFG, call graph

**Lattice** $\mathcal{L} = (\mathcal{D}, \sqsubseteq)$ Top, Bottom $\top$, $\bot$

**Induced partial order** $x \sqsubseteq y$

**Set of Transfer functions** $f: \mathcal{D} \rightarrow \mathcal{D}$

- include identity, constant functions
- closed under composition of functions, $\circ$

Function assigned to each node to summarize its effects

---

Example: Liveness Problem, 2 variables

**Flow Graph** $\mathcal{F}$  Reverse CFG

**Lattice** $\mathcal{L} = (\mathcal{D}, \sqsubseteq)$ Top, Bottom $\top$, $\bot$

\[ \top = \phi \]

\[ \{x\} \rightarrow \{y\} \rightarrow \{x,y\} = \bot \]

- elements of $\mathcal{D}$ form the lattice

**Set of Transfer functions**

- All functions $\mathcal{D} \rightarrow \mathcal{D}$

---

3

---

Example: Liveness Problem, 2 variables

**Flow Graph** $\mathcal{F}$  Reverse CFG

**Lattice** $\mathcal{L} = (\mathcal{D}, \sqsubseteq)$ Top, Bottom $\top$, $\bot$

\[ \top = \phi \]

\[ \{x\} \rightarrow \{y\} \rightarrow \{x,y\} = \bot \]

- elements of $\mathcal{D}$ form the lattice

**Set of Transfer functions**

- All functions $\mathcal{D} \rightarrow \mathcal{D}$

---

4
Iterative Algorithm

Initialization: Reverse CFG if backward problem.
Initialize \( \text{Out}(B), \text{In}(B) \)

Worklist Algorithm:

Worklist := Set of all nodes

While (Worklist \neq \emptyset)

Remove node \( N \) from worklist

\( \text{OldOut} := \text{Out}(N) \)

\( \text{In}(B) = \bigcup_{P \text{ pred of } B} \text{Out}(P) \)

\( \text{Out}(B) = f_B(\text{In}(B)) \)

if \( \text{Out}(N) \neq \text{OldOut} \) then add successors of \( N \)
to Worklist

---

How know iterative algorithm works?

Termination:

Finite Lattice Functions have finite no. of possible values

\[
\begin{array}{c}
(\{x_1,x_2\}, \{x_1,x_3\}) \\
(\{x_2\}, \{x_1, x_2\}) \\
(\{\phi\}, \{x_1, x_2, x_3\})
\end{array}
\]

finite depth

Monotone Functions are non-increasing or decreasing

Iterative Method guaranteed to terminate:
Iterate till no change, all equations simultaneously satisfied

Quality of solution?
Quality of Iterative Solution

**Best Solution**: Holds for all real paths taken during program execution.

**Meet-Over-all Paths (MOP)**:
Iterative solution over all paths

\[ \text{MOP}(B) = \bigcap \{ f \mid f \geq B \} \]

where \( f \) is the initial lattice value at Entry and \( f_p \) is \( f_{B_n} \circ \ldots \circ f_{B_1} \)

for \( p = B_1 \ldots B_n \).

MOP is undecidable (even if monotonic)

---

When can MOP be achieved?

**Distributive property**: \( f(X \sqcup Y) = f(X) \sqcup f(Y) \)

Merge then apply \( f \) same as apply \( f \) then merge

**Distributive** → MOP solution via iterative algorithm

What is not distributive?

**Constant propagation**

Ex. \( X := 2 \)  \( Y := 3 \)  \( X := 3 \)  \( Y := 2 \)  \( \Rightarrow X + Y \)

**Maximal Fixed Point (MFP) Solution**:
Achieved by iterative algorithm on all problems covered.

MFP = MOP if distributive

MFP ≤ MOP ≤ Best
Summary Iterative Algorithm

Complexity: \( O(N^2) \) where \( N \) = size of FG

May take long to converge
   Can improve by good choice of node order, ...

Simple to implement

Handles irreducible graphs

Doesn't recognize program structure
   Loops, intervals

Interval-based Data Flow Analysis

Local Propagation:
   For each interval, in order inner to outer,
   collect local info for each node in the interval
   use the local info for the interval nodes to collect
   info for the node representing the interval

Global Propagation:
   For each interval, in order outer to inner,
   given the IN set for the header to the interval,
   propagate global info to all nodes in the interval

Result: CFG with global data flow info
Interval-based Solution

LOCAL: inner to outer

GLOBAL: outer to inner

\[ f_{R1} = \lim_{n \to \infty} (f_{B1} \circ f_{B2})^n \]

\[ f_{R2}(\text{In}) = \text{Out} \]

Summary Interval-based Algorithm

**Complexity:** \( O(E \cdot (E)) \)
- if reducible (\( E\) grows slowly)
- exponential if irreducible

**Backward Problems more difficult**
- Reverse CFG can have multiple entry loops

**More complicated implementation**

**Uses more space**

**Irreducible subgraphs handled separately**

**Allows for incremental update**

**Often used in practice**