Data-Flow Frameworks

Lattice-Theoretic Formulation
Meet-Over-Paths Solution
Monotonicity/Distributivity

Jeffrey D. Ullman
Stanford University
Commonality among different data-flow problems allows new approaches to propagate to many algorithms at once.

There are actually many other data-flow problems we need to solve, using essentially the same ideas as we’ve seen so far.

- We’ll see some in the discussions of partial redundancy, pointer analysis, alias analysis, array-bounds checking, SQL injections, other bugs.
Important components:

1. *Direction* $D$: forward or backward.
2. *Domain* $V$ (possible values for IN, OUT).
3. *Meet operator* $\land$ (effect on $V$ of path confluence).
4. *Set of possible transfer functions* $F$ (effect on $V$ of passing through a basic block in the direction $D$).
This theory was the thesis at U. Wash. of Gary Kildall.
Gary is better known for CP/M, the first real PC operating system.
There is an interesting story.
  - Google query: [kildall cp/m].
V and $\land$ form a semilattice if for all $x$, $y$, and $z$ in $V$:

1. $x \land y = y \land x$ (commutativity).
2. $x \land (y \land z) = (x \land y) \land z$ (associativity).
3. $x \land x = x$ (idempotence).
4. Top element $\top$ such that for all $x$, $\top \land x = x$.
5. Bottom element $\bot$ such that for all $x$, $\bot \land x = \bot$.

- Optional and not generally used in the theory.
Example: Semilattice

- $V = \text{power set of some finite set.}$
- $\land = \text{union.}$
- Union is idempotent, commutative, and associative.
- What are the top and bottom elements?
Example: Powerset Semilattice

Convention:
Draw the meet of elements below both.

\[ x \land y \]

\{1\} \quad \{2\} \quad \{3\}

\{1,2\} \quad \{1,3\} \quad \{2,3\}

\{1,2,3\}
Partial Order for a Semilattice

- Say $x \leq y$ iff $x \land y = x$.
- Also, $x < y$ iff $x \leq y$ and $x \neq y$.
- $\leq$ really is a partial order:
  1. $x \leq y$ and $y \leq z$ imply $x \leq z$ (proof in text).
  2. $x \leq y$ and $y \leq x$ iff $x = y$.

Proof:
- If: Idempotence gives us $x \leq x$.
- Only-if: $x \land y = x$ and $y \land x = y$ (given).
  Thus, $x = x \land y$ (given) = $y \land x$ (commutativity) = $y$ (given).
Axioms for Transfer Functions

1. F includes the identity function.
   - Why needed?
     - Only logical transfer function for an empty block.
2. F is closed under composition.
   - Why needed?
     - Transfer function for a path is the composition of transfer functions for the blocks along that path.
     - And the composition is associative (order doesn’t matter).
The problems seen so far fit the model.

- **RD’s**: Forward, meet = union, transfer functions based on Gen and Kill.
- **AE’s**: Forward, meet = intersection, transfer functions based on Gen and Kill.
- **LV’s**: Backward, meet = union, transfer functions based on Use and Def.
Example: Reaching Definitions

- Direction $D = \text{forward}$.  
- Domain $V = \text{set of all sets of definitions in the flow graph}$.  
- $\wedge = \text{union}$.  
- Functions $F = \text{all "gen-kill" functions of the form } f(x) = (x - K) \cup G, \text{ where } K \text{ and } G \text{ are sets of definitions (members of } V)$.  

Example: RD Satisfies Axioms

- Union on a power set forms a semilattice (idempotent, commutative, associative).
- Identity function: let $K = G = \emptyset$.
- Composition closure: A little algebra.
  \[
  f_1(x) = (x - K_1) \cup G_1 \\
  f_2(x) = (x - K_2) \cup G_2 \\
  f_1(f_2(x)) = (((x - K_2) \cup G_2) - K_1) \cup G_1 = \\
  (x - (K_1 \cup K_2)) \cup ((G_2 - K_1) \cup G_1)
  \]
- Associativity of composition: more algebra.
**Example: Partial Order**

- For RD’s, $S \leq T$ means $S \cup T = S$.
- Equivalently $S \supseteq T$.
  - Seems “backward,” but that’s what the definitions give you.
- **Intuition**: $\leq$ measures “ignorance.”
  - The more definitions we know about, the less ignorance we have.
  - $\top$ = “total ignorance.”
Using a DFA Framework

- We apply an iterative algorithm to the following information:
  1. DFA framework \((D, V, \wedge, F)\).
  2. A flow graph, with an associated function \(f_B\) in \(F\) for each block \(B\).
     - **Assumption**: There are Entry and Exit blocks that do nothing. Entry has no predecessors; Exit has no successors.
     - Sometimes can be ignored.
  3. A **boundary value** \(v_{\text{ENTRY}}\) or \(v_{\text{EXIT}}\) if \(D = \text{forward or backward, respectively.}\)
The interesting stuff happens here
Iterative Algorithm (Forward)

\[
\text{OUT}[\text{Entry}] = v_{\text{ENTRY}};
\]
\[
\text{for (other blocks } B) \quad \text{OUT}[B] = T;
\]
\[
\text{while (changes to any OUT)}
\]
\[
\text{for (each block } B) \{ \\
\quad \text{IN}(B) = \bigwedge \text{ predecessors } P \text{ of } B \text{ OUT}(P); \\
\quad \text{OUT}(B) = f_B(\text{IN}(B)); \\
\}
\]
Iterative Algorithm (Backward)

- Same thing – just:
  1. Swap IN and OUT everywhere.
  2. Replace entry by exit.
What Does the Iterative Algorithm Do?

- **MFP** (*maximal fixedpoint*) = result of iterative algorithm.
- **MOP** = “Meet Over all Paths” (from the entry to a given point) of the transfer function along that path applied to \( v_{\text{ENTRY}} \).
- **IDEAL** = ideal solution = meet over all executable paths from entry to the point.

**Question**: why might a path not be “executable”?
Convention: We’ll write compositions from left to right, as $f_1 f_2 f_3 ...$ so they look like the paths themselves.
Maximum Fixedpoint

- **Fixedpoint** = solution to the equations used in iteration:
  \[
  \text{IN}(B) = \bigwedge \text{predecessors } P \text{ of } B \text{ OUT}(P); \\
  \text{OUT}(B) = f_B(\text{IN}(B));
  \]

- **Maximum** = IN’s and OUT’s of any other solution are \(\leq\) the result of the iterative algorithm (MFP).

- **Subtle point**: “maximum” = “maximum ignorance” = “we don’t believe any fact that is not justified by the flow graph and the equations.”

  - This is a good thing.
Example: Reaching Definitions

- MFP = solution with maximum “ignorance.”
- That is, the smallest sets of definitions that satisfy the equations.
- Makes sense:
  - We need to discover all definitions that really reach.
    - That’s what “fixedpoint” gives us.
  - But we want to avoid “imaginary” definitions that are not justified.
    - That’s what “maximum” gives us.
Here is the MFP

But if we add d at these two points, we still have a solution, just not the maximal (ignorance) solution.
All solutions are really meets of the result of starting with $v_{\text{ENTRY}}$ and following some set of paths (possibly imaginary) to the point in question.

If we don’t include at least the IDEAL paths, we have an error.

- But try not to include too many more.
MOP Versus IDEAL

- At each block B, MOP[B] ≤ IDEAL[B].
- **Why**? The meet over many paths is ≤ the meet over a subset of those paths.
- **Example**: \( x \land y \land z \leq x \land y \) because
  \[
  (x \land y \land z) \land (x \land y) = x \land y \land z.
  \]
  - Use commutativity, associativity, idempotence.
MOP Versus IDEAL – (2)

- **Intuition**: Anything not $\leq$ IDEAL is not safe, because there is some executable path whose effect is not accounted for.
- **Conversely**: any solution that is $\leq$ IDEAL accounts for all executable paths (and maybe more paths), and is therefore safe, even if not 100% accurate.
MFP Versus MOP

- Is MFP ≤ MOP?
  - If so, then since MOP ≤ IDEAL, we have MFP ≤ IDEAL, and therefore MFP is safe.
- Yes, but ...
- Requires two assumptions about the framework:
  1. “Monotonicity.”
  2. *Finite height* (no infinite chains \( \ldots < x_2 < x_1 < x \)).
     - Needed to assure convergence of the algorithm.
     - Example: OK for power sets of a finite set, and therefore for the frameworks we have seen.
**Intuition**: If we computed the MOP directly, we would compose functions along all paths, then take a big meet:

\[ \bigwedge_{\text{all paths } P \text{ to block } B} f_P(v_{\text{ENTRY}}) \]

- But the MFP (iterative algorithm) alternates compositions and meets arbitrarily.
  - That’s why we need the “monotonicity” assumption.
A framework is *monotone* if the functions respect $\leq$. That is:
- If $x \leq y$, then $f(x) \leq f(y)$.
- Equivalently: $f(x \land y) \leq f(x) \land f(y)$.

Proof that these are equivalent is on p. 625 of the text. But we only need the second; the first justifies the term “monotonicity.”
Good News!

- The frameworks we’ve studied so far are all monotone.
  - Easy proof for functions in Gen-Kill form.
- And they have finite height.
  - Only a finite number of defs, variables, etc. in any program.
Since $f(x \land y) \leq f(x) \land f(y)$, the MFP is safe.
Strictly stronger than monotonicity is the *distributivity* condition:

\[ f(x \land y) = f(x) \land f(y) \]
Even More Good News!

- All the Gen-Kill frameworks are distributive.
- If a framework is distributive, then combining paths early doesn’t hurt.
  - MOP = MFP.
  - That is, the iterative algorithm computes a solution that takes into account all and only the physical paths.
Reducible Flow Graphs

Depth-First Spanning Trees
Depth-First Ordering
Dominator in Flow Graphs
1. Proper ordering of nodes of a flow graph speeds up the iterative algorithms: “depth-first ordering.”
2. “Normal” flow graphs have a surprising property --- “reducibility” --- that simplifies several matters.
3. Outcome: few iterations “normally” needed.
Depth-First Search

- Start at entry.
- If you can follow an edge to an unvisited node, do so.
- If not, backtrack to your parent (node from which you were visited).
- Constructs a Depth-First Spanning Tree:
  - Root = entry.
  - Tree edges are the edges along which we first visit the node at the head.
Example: DFST
Depth-First Node Order

- The reverse of the order in which a DFS retreats from the nodes.
- Alternatively, reverse of postorder traversal of the tree.
Example: DF Order
Four Kinds of Edges

1. Tree edges.
2. *Forward edges* (node to proper descendant).
3. *Retreating edges* (node to ancestor).
   - Includes a self-loop.
4. *Cross edges* (between two nodes, neither of which is an ancestor of the other.)
Of these edges, only retreating edges go from high to low in DF order.

- **Example** of proof: You must retreat from the head of a tree edge before you can retreat from its tail.
  - So head gets a higher number than the tail.

- **Also surprising**: all cross edges go right to left in the DFST.
  - Assuming we add children of any node from the left.
Example: Non-Tree Edges

Retreating

Forward

Cross
1. “Normal” flow graphs are “reducible.”
2. “Dominators” needed to explain reducibility.
3. In reducible flow graphs, loops have single entry points, and retreating edges are independent of the DFST chosen (and called “back” edges).
4. Leads to relationship between DF order and efficient iterative algorithm.
Dominators

- Node $d$ *dominates* node $n$ if every path from the entry to $n$ goes through $d$.
- Text has a forward-intersection iterative algorithm for finding dominators.
- Quick observations:
  1. Every node dominates itself.
  2. The entry dominates every node.
Example: Dominators
Common Dominator Cases

- The test of a while loop dominates all blocks in the loop body.
- The test of an if-then-else dominates all blocks in either branch.
Back Edges

- An edge is a *back edge* if its head dominates its tail.
- **Theorem**: Every back edge is a retreating edge in every DFST of every flow graph.
  - Converse almost always true, but not always.

Note: We assume all nodes are reachable from the entry, or else they can be deleted from the flow graph.
Example: Back Edges

![Diagram of a graph with back edges labeled {1,4}, {1,5}, {1,2}, and {1,2,3}.]
Reducible Flow Graphs

- A flow graph is *reducible* if every retreating edge in any DFST for that flow graph is a back edge.
- **Testing reducibility**: Remove all back edges from the flow graph and check that the result is acyclic.
- **Hint why it works**: All cycles must include some retreating edge in every DFST.
  - In particular, the edge that enters the first node in the cycle that is visited by the DF search.
DFST on a Cycle

Search must reach these nodes before leaving the cycle.

So this is a retreating edge.

Depth-first search reaches here first.
Example: Remove Back Edges

Diagram of a graph with nodes 1, 2, 3, 4, and 5 connected by arrows indicating the direction of the edges.
Example: Remove Back Edges

Remaining graph is acyclic.
Why Reducibility?

- **Folk theorem**: All flow graphs in practice are reducible.
- **Fact**: If you use only while-loops, for-loops, repeat-loops, if-then(-else), break, and continue, then your flow graph is reducible.
In any DFST, one of these edges will be a retreating edge.

But no heads dominate their tails, so deleting back edges leaves the cycle.