Lecture 35

Code Optimizations

Awanish Pandey

Department of Computer Science and Engineering
Indian Institute of Technology
Roorkee

April 29, 2025

Lattice Theory

- Way to represent a Partial Order set

Lattice Theory

- Way to represent a Partial Order set
- Greatest Lower Bound (GLB)

Lattice Theory

- Way to represent a Partial Order set
- Greatest Lower Bound (GLB)
  - $g \leq x$

Lattice Theory

- Way to represent a Partial Order set
- Greatest Lower Bound (GLB)
  - $g \leq x$
  - $g \leq y$

Lattice Theory

- Way to represent a Partial Order set
- Greatest Lower Bound (GLB)
  - $g \leq x$
  - $g \leq y$
  - $z \leq x \text{ and } z \leq y \text{ then } z \leq g$

Lattice Theory

- Way to represent a Partial Order set
- Greatest Lower Bound (GLB)
  - $g \leq x$
  - $g \leq y$
  - $z \leq x \; and \; z \leq y \; then \; z \leq g$
- Height of lattice

Lattice Theory

- Way to represent a Partial Order set
- Greatest Lower Bound (GLB)
  - $g \leq x$
  - $g \leq y$
  - $z \leq x$ *and* $z \leq y$ *then* $z \leq g$
- Height of lattice
- Type of solutions

Lattice Theory

- Way to represent a Partial Order set
- Greatest Lower Bound (GLB)
  - $g \leq x$
  - $g \leq y$
  - $z \leq x$ *and* $z \leq y$ *then* $z \leq g$
- Height of lattice
- Type of solutions
  - Ideal: Collect information from all the feasible paths (undecidable)

Lattice Theory

- Way to represent a Partial Order set
- Greatest Lower Bound (GLB)
  - $g \leq x$
  - $g \leq y$
  - $z \leq x$ *and* $z \leq y$ *then* $z \leq g$
- Height of lattice
- Type of solutions
  - Ideal: Collect information from all the feasible paths (undecidable)
  - MOP: Collect information from all the paths (unbounded)

Lattice Theory

- Way to represent a Partial Order set
- Greatest Lower Bound (GLB)
  - $g \leq x$
  - $g \leq y$
  - $z \leq x \text{ and } z \leq y \text{ then } z \leq g$
- Height of lattice
- Type of solutions
  - Ideal: Collect information from all the feasible paths (undecidable)
  - MOP: Collect information from all the paths (unbounded)
  - MFP: Visit basic block (not necessary in execution order) and collect the information

Lattice Theory

- Way to represent a Partial Order set
- Greatest Lower Bound (GLB)
  - $g \leq x$
  - $g \leq y$
  - $z \leq x \text{ and } z \leq y \text{ then } z \leq g$
- Height of lattice
- Type of solutions
  - Ideal: Collect information from all the feasible paths (undecidable)
  - MOP: Collect information from all the paths (unbounded)
  - MFP: Visit basic block (not necessary in execution order) and collect the information
- Transfer function

Lattice Theory

- Way to represent a Partial Order set
- Greatest Lower Bound (GLB)
  - $g \leq x$
  - $g \leq y$
  - $z \leq x \text{ and } z \leq y \text{ then } z \leq g$
- Height of lattice
- Type of solutions
  - Ideal: Collect information from all the feasible paths (undecidable)
  - MOP: Collect information from all the paths (unbounded)
  - MFP: Visit basic block (not necessary in execution order) and collect the information
- Transfer function
  - Monotone

Lattice Theory

- Way to represent a Partial Order set
- Greatest Lower Bound (GLB)
  - $g \leq x$
  - $g \leq y$
  - $z \leq x \textit{ and } z \leq y \textit{ then } z \leq g$
- Height of lattice
- Type of solutions
  - Ideal: Collect information from all the feasible paths (undecidable)
  - MOP: Collect information from all the paths (unbounded)
  - MFP: Visit basic block (not necessary in execution order) and collect the information
- Transfer function
  - Monotone
  - Distributive

Lattice Theory

- Way to represent a Partial Order set
- Greatest Lower Bound (GLB)
  - $g \leq x$
  - $g \leq y$
  - $z \leq x \textit{ and } z \leq y \textit{ then } z \leq g$
- Height of lattice
- Type of solutions
  - Ideal: Collect information from all the feasible paths (undecidable)
  - MOP: Collect information from all the paths (unbounded)
  - MFP: Visit basic block (not necessary in execution order) and collect the information
- Transfer function
  - Monotone
  - Distributive

If transfer function is monotone and distributive, then MFP solution will be equal to MOP.

Loops

- Dominators

Loops

- Dominators
  - Node d dominates node n, if every path from the *entry* node of the flow graph to n goes through d.

Loops

- Dominators
  - Node d dominates node n, if every path from the *entry* node of the flow graph to n goes through d.
  - Every node dominates itself.

Loops

- Dominators
  - Node d dominates node n, if every path from the *entry* node of the flow graph to n goes through d.
  - Every node dominates itself.
  - $IN[B] = \cap_{P \text{ is predecessor of } B} OUT[P]$

Meet operator is the intersection to calculate dominators.

Loops

- Dominators
  - Node d dominates node n, if every path from the *entry* node of the flow graph to n goes through d.
  - Every node dominates itself.
  - $IN[B] = \cap_{P \text{ is predecessor of } B} OUT[P]$
  - $OUT[B] = \{B\} \cup IN[B]$

Loops

- Dominators
  - Node d dominates node n, if every path from the *entry* node of the flow graph to n goes through d.
  - Every node dominates itself.
  - $IN[B] = \cap_{P\ is\ predecessor\ of\ B} OUT[P]$
  - $OUT[B] = \{B\} \cup IN[B]$
- Depth First Ordering (Reverse of post order sequence)

Loops

- Dominators
  - Node d dominates node n, if every path from the *entry* node of the flow graph to n goes through d.
  - Every node dominates itself.
  - $IN[B] = \cap_{P \text{ is predecessor of } B} OUT[P]$
  - $OUT[B] = \{B\} \cup IN[B]$
- Depth First Ordering (Reverse of post order sequence)
- Edges in Depth First Spanning tree

Loops

- Dominators
  - Node d dominates node n, if every path from the *entry* node of the flow graph to n goes through d.
  - Every node dominates itself.
  - $IN[B] = \cap_{P \text{ is predecessor of } B} OUT[P]$
  - $OUT[B] = \{B\} \cup IN[B]$
- Depth First Ordering (Reverse of post order sequence)
- Edges in Depth First Spanning tree
  - Advancing Edge

Loops

- Dominators
  - Node d dominates node n, if every path from the *entry* node of the flow graph to n goes through d.
  - Every node dominates itself.
  - $IN[B] = \cap_{P \text{ is predecessor of } B} OUT[P]$
  - $OUT[B] = \{B\} \cup IN[B]$
- Depth First Ordering (Reverse of post order sequence)
- Edges in Depth First Spanning tree
  - Advancing Edge
  - Retreating edge

Loops

- Dominators
  - Node d dominates node n, if every path from the *entry* node of the flow graph to n goes through d.
  - Every node dominates itself.
  - $IN[B] = \cap_{P \text{ is predecessor of } B} OUT[P]$
  - $OUT[B] = \{B\} \cup IN[B]$
- Depth First Ordering (Reverse of post order sequence)
- Edges in Depth First Spanning tree
  - Advancing Edge
  - Retreating edge
  - Back edge

Loops

- Dominators
  - Node d dominates node n, if every path from the *entry* node of the flow graph to n goes through d.
  - Every node dominates itself.
  - $IN[B] = \cap_{P \text{ is predecessor of } B} OUT[P]$
  - $OUT[B] = \{B\} \cup IN[B]$
- Depth First Ordering (Reverse of post order sequence)
- Edges in Depth First Spanning tree
  - Advancing Edge
  - Retreating edge
  - Back edge
  - Cross Edges

Loops

- Dominators
  - Node d dominates node n, if every path from the *entry* node of the flow graph to n goes through d.
  - Every node dominates itself.
  - $IN[B] = \cap_{P \text{ is predecessor of } B} OUT[P]$
  - $OUT[B] = \{B\} \cup IN[B]$
- Depth First Ordering (Reverse of post order sequence)
- Edges in Depth First Spanning tree
  - Advancing Edge
  - Retreating edge
  - Back edge
  - Cross Edges
- Reducible if all retreating edges are back edges

Loops

- Dominators
  - Node d dominates node n, if every path from the *entry* node of the flow graph to n goes through d.
  - Every node dominates itself.
  - $IN[B] = \cap_{P \text{ is predecessor of } B} OUT[P]$
  - $OUT[B] = \{B\} \cup IN[B]$
- Depth First Ordering (Reverse of post order sequence)
- Edges in Depth First Spanning tree
  - Advancing Edge
  - Retreating edge
  - Back edge
  - Cross Edges
- Reducible if all retreating edges are back edges
- Natural Loops

Loops

- Dominators
  - Node d dominates node n, if every path from the *entry* node of the flow graph to n goes through d.
  - Every node dominates itself.
  - $IN[B] = \cap_{P \text{ is predecessor of } B} OUT[P]$
  - $OUT[B] = \{B\} \cup IN[B]$
- Depth First Ordering (Reverse of post order sequence)
- Edges in Depth First Spanning tree
  - Advancing Edge
  - Retreating edge
  - Back edge
  - Cross Edges
- Reducible if all retreating edges are back edges
- Natural Loops
  - Single entry node (header)

Loops

- Dominators
  - Node d dominates node n, if every path from the *entry* node of the flow graph to n goes through d.
  - Every node dominates itself.
  - $IN[B] = \cap_{P \text{ is predecessor of } B} OUT[P]$
  - $OUT[B] = \{B\} \cup IN[B]$
- Depth First Ordering (Reverse of post order sequence)
- Edges in Depth First Spanning tree
  - Advancing Edge
  - Retreating edge
  - Back edge
  - Cross Edges
- Reducible if all retreating edges are back edges
- Natural Loops
  - Single entry node (header)
  - There must be a back edge that enters loop header

Loops

- Dominators
  - Node d dominates node n, if every path from the *entry* node of the flow graph to n goes through d.
  - Every node dominates itself.
  - $IN[B] = \cap_{P \text{ is predecessor of } B} OUT[P]$
  - $OUT[B] = \{B\} \cup IN[B]$
- Depth First Ordering (Reverse of post order sequence)
- Edges in Depth First Spanning tree
  - Advancing Edge
  - Retreating edge
  - Back edge
  - Cross Edges
- Reducible if all retreating edges are back edges
- Natural Loops
  - Single entry node (header)
  - There must be a back edge that enters loop header
  - If there is backedge $n \rightarrow d$, natural loop will be d and all the nodes that can reach n without going through d.

Thank You