CS412/413

Introduction to Compilers
Radu Rugina

Lecture 23: More Dataflow Analysis
24 Mar 03

Lattices

- Lattice:
  - Set augmented with a partial order relation $\sqsubseteq$
  - Each subset has a LUB and a GLB
  - Can define: meet $\sqcap$, join $\sqcup$, top $\top$, bottom $\perp$
- Use lattice in the compiler to express information about the program
- To compute information: build constraints which describe how the lattice information changes
  - Effect of instructions: transfer functions
  - Effect of control flow: meet operation

Properties of Meet and Join

- The meet and join operators are:
  1. Associative $(x \sqcap y) \sqcap z = x \sqcap (y \sqcap z)$
  2. Commutative $x \sqcap y = y \sqcap x$
  3. Idempotent: $x \sqcap x = x$
- Property: If "$\sqcap$" is an associative, commutative, and idempotent operator, then the relation "$\sqsubseteq$" defined as $x \sqsubseteq y$ iff $x \sqcap y = x$ is a partial order
- Above property provides an alternative definition of a partial orders and lattices starting from the meet (join) operator

Transfer Functions

- Let L = dataflow information lattice
- Transfer function $F_I : L \to L$ for each instruction I
  - Describes how I modifies the information in the lattice
  - If in[I] is info before I and out[I] is info after I, then
    Forward analysis: $out[I] = F_I(in[I])$
    Backward analysis: $in[I] = F_I(out[I])$
- Transfer function $F_B : L \to L$ for each basic block B
  - Is composition of transfer functions of instructions in B
  - If in[B] is info before B and out[B] is info after B, then
    Forward analysis: $out[B] = F_B(in[B])$
    Backward analysis: $in[B] = F_B(out[B])$

Monotonicity and Distributivity

- Two important properties of transfer functions
- Monotonicity: function $F : L \to L$ is monotonic if
  $x \sqsubseteq y$ implies $F(x) \sqsubseteq F(y)$
- Distributivity: function $F : L \to L$ is distributive if
  $F(x \sqcap y) = F(x) \sqcap F(y)$
- Property: F is monotonic iff $F(x \sqcap y) \sqsubseteq F(x) \sqcap F(y)$
  - any distributive function is monotonic!

Proof of Property

- Prove that the following are equivalent:
  1. $x \sqsubseteq y$ implies $F(x) \sqsubseteq F(y)$, for all x, y
  2. $F(x \sqcap y) \sqsubseteq F(x) \sqcap F(y)$, for all x, y
- Proof for "1 implies 2"
  - Need to prove that $F(x \sqcap y) \sqsubseteq F(x)$ and $F(x \sqcap y) \sqsubseteq F(y)$
  - Use $x \sqcap y \sqsubseteq x$, $x \sqcap y \sqsubseteq y$, and property 1
- Proof of "2 implies 1"
  - Let x, y such that $x \sqsubseteq y$
  - Then $x \sqcap y = x$, so $F(x \sqcap y) = F(x)$
  - Use property 2 to get $F(x) \sqsubseteq F(x) \sqcap F(y)$
  - Hence $F(x) \sqsubseteq F(y)$

Control Flow

- Meet operation models how to combine information at split/join points in the control flow
  - If in[B] is info before B and out[B] is info after B, then:

    Forward analysis: $in[B] = \sqcap \{out[B'] \mid B' \in pred(B)\}$

    Backward analysis: $out[B] = \sqcap \{in[B'] \mid B' \in succ(B)\}$

- Can alternatively use join operation $\sqcup$ (equivalent to using the meet operation $\sqcap$ in the reversed lattice)

Monotonicity of Meet

- Meet operation is also monotonic over L x L:

  $x1 \sqsubseteq y1$ and $x2 \sqsubseteq y2$ implies $(x1 \sqcap x2) \sqsubseteq (y1 \sqcap y2)$

- Proof:
  - any lower bound of {x1,x2} is also a lower bound of {y1,y2}, because $x1 \sqsubseteq y1$ and $x2 \sqsubseteq y2$
  - $x1 \sqcap x2$ is a lower bound of {x1,x2}
  - So $x1 \sqcap x2$ is a lower bound of {y1,y2}
  - But $y1 \sqcap y2$ is the greatest lower bound of {y1,y2}
  - Hence $(x1 \sqcap x2) \sqsubseteq (y1 \sqcap y2)$

Forward Dataflow Analysis

- Control flow graph G with entry (start) node B_s
- Lattice $(L, \sqsubseteq)$ represents information about program
  - Meet operator $\sqcap$, top element $\top$
- Monotonic transfer functions
  - Transfer function $F_I: L \to L$ for each instruction I
  - Can derive transfer functions F_B for basic blocks
- Goal: compute the information at each program point, given the information at entry of B_s is X_0
- Require the solution to satisfy:

  $out[B] = F_B(in[B])$, for all B

  $in[B] = \sqcap \{out[B'] \mid B' \in pred(B)\}$, for all B

  $in[B_s] = X_0$

Backward Dataflow Analysis

- Control flow graph G with exit node B_e
- Lattice $(L, \sqsubseteq)$ represents information about program
  - Meet operator $\sqcap$, top element $\top$
- Monotonic transfer functions
  - Transfer function $F_I: L \to L$ for each instruction I
  - Can derive transfer functions F_B for basic blocks
- Goal: compute the information at each program point, given the information at exit of B_e is X_0
- Require the solution to satisfy:

  $in[B] = F_B(out[B])$, for all B

  $out[B] = \sqcap \{in[B'] \mid B' \in succ(B)\}$, for all B

  $out[B_e] = X_0$

Dataflow Equations

- The constraints are called dataflow equations:

  $out[B] = F_B(in[B])$, for all B

  $in[B] = \sqcap \{out[B'] \mid B' \in pred(B)\}$, for all B

  $in[B_s] = X_0$

- Solve equations: use an iterative algorithm
  - Initialize $in[B_s] = X_0$
  - Initialize everything else to $\top$
  - Repeatedly apply rules
  - Stop when reach a fixed point

Algorithm

$in[B_S] = X_0$

$out[B] = \top$, for all B

Repeat

  For each basic block $B \neq B_s$

    $in[B] = \sqcap \{out[B'] \mid B' \in pred(B)\}$

  For each basic block B

    $out[B] = F_B(in[B])$

Until no change

Efficiency

- Algorithm is inefficient
  - Effects of basic blocks re-evaluated even if the input information has not changed
- Better: re-evaluate blocks only when necessary
- Use a worklist algorithm
  - Keep of list of blocks to evaluate
  - Initialize list to the set of all basic blocks
  - If out[B] changes after evaluating out[B] = F_B(in[B]), then add all successors of B to the list

Worklist Algorithm

in[B_S] = X_0
out[B] = ⊤, for all B
worklist = set of all basic blocks B

Repeat
    Remove a node B from the worklist
    in[B] = ⊓ {out[B′] | B′∈pred(B)}
    out[B] = F_B(in[B])
    if out[B] has changed, then
        worklist = worklist ∪ succ(B)
Until worklist = ∅

Correctness

- Initial algorithm is correct
  - If dataflow information does not change in the last iteration, then it satisfies the equations
- Worklist algorithm is correct
  - Maintains the invariant that

    in[B] = ⊓ {out[B′] | B′∈pred(B)}
    out[B] = F_B(in[B])

    for all the blocks B not in the worklist
  - At the end, worklist is empty

Termination

- Do these algorithms terminate?
- Key observation: at each iteration, information decreases in the lattice

  $in_{k+1}[B] \sqsubseteq in_k[B]$ and $out_{k+1}[B] \sqsubseteq out_k[B]$

  where in_k[B] is info before B at iteration k and out_k[B] is info after B at iteration k
- Proof by induction:
  - Induction basis: true, because we start with top element, which is greater than everything
  - Induction step: use monotonicity of transfer functions and meet operation
- Information forms a chain: $in_1[B] \sqsupseteq in_2[B] \sqsupseteq in_3[B]$ …

Chains in Lattices

- A chain in a lattice L is a totally ordered subset S of L:

  x ⊑ y or y ⊑ x for any x, y ∈ S
- In other words:

  Elements in a totally ordered subset S can be indexed to form an ascending sequence:

  $x_1 \sqsubseteq x_2 \sqsubseteq x_3 \sqsubseteq \dots$

  or they can be indexed to form a descending sequence:

  $x_1 \sqsupseteq x_2 \sqsupseteq x_3 \sqsupseteq \dots$
- Height of a lattice = size of its largest chain
- Lattice with finite height: only has finite chains

Termination

- In the iterative algorithm, for each block B:

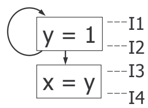
  {in_1[B], in_2[B], …}

  is a chain in the lattice, because transfer functions and meet operation are monotonic
- If lattice has finite height then these sets are finite, i.e. there is a number k such that $in_i[B] = in_{i+1}[B]$, for all i ≥ k and all B
- If $in_i[B] = in_{i+1}[B]$ then also $out_i[B] = out_{i+1}[B]$
- Hence algorithm terminates in at most k iterations
- To summarize: dataflow analysis terminates if
  1. Transfer functions are monotonic
  2. Lattice has finite height

Multiple Solutions

- The iterative algorithm computes a solution of the system of dataflow equations
- … is the solution unique?
- No, dataflow equations may have multiple solutions !
- Example: live variables

Equations: I1 = I2-{y}
I3 = (I4-{x}) U {y}
I2 = I1 U I3
I4 = {}



Solution 1: I1={}, I2={y}, I3={y}, I4={}
Solution 2: I1={x}, I2={x,y}, I3={y}, I4={}

Safety

- Solution for live variable analysis:
  - Sets of live variables must include each variable whose values will further be used in some execution
  - … may also include variables never used in any execution!
- The analysis is safe if it takes into account all possible executions of the program
  - … may also characterize cases which never occur in any execution of the program
  - Say that the analysis is a conservative approximation of all executions
- In example
  - Solution 2 includes x in live set I1, which is not used later
  - However, analysis is conservative

Safety and Precision

- Safety: dataflow equations guarantee a safe solution to the analysis problem
- Precision: a solution to an analysis problem is more precise if it is less conservative
- Live variables analysis problem:
  - Solution is more precise if the sets of live variables are smaller
  - Solution which reports that all variables are live at each point is safe, but is the least precise solution
- In the lattice framework: S1 is less precise than S2 if the result in S1 at each program point is less than the corresponding result in S2 at the same point
  - Use notation $S1 \sqsubseteq S2$ if solution S1 is less precise than S2

Maximal Fixed Point Solution

- Property: among all the solutions to the system of dataflow equations, the iterative solution is the most precise
- Intuition:
  - We start with the top element at each program point (i.e. most precise information)
  - Then refine the information at each iteration to satisfy the dataflow equations
  - Final result will be the closest to the top
- Iterative solution for dataflow equations is called Maximal Fixed Point solution (MFP)
- For any solution FP of the dataflow equations: $FP \sqsubseteq MFP$

Meet Over Paths Solution

- Is MFP the best solution to the analysis problem?
- Another approach: consider a lattice framework, but use a different way to compute the solution
  - Let G be the control flow graph with start block B_0
  - For each path $p_n=[B_0, B_1, \ldots, B_n]$ from entry to block B_n:
    $in[p_n] = F_{Bn-1}(\ldots(F_{B1}(F_{B0}(X0))))$
  - Compute solution as
    $in[B_n] = \sqcap \{ in[p_n] \mid \text{all paths } p_n \text{ from } B_0 \text{ to } B_n \}$
- This solution is the Meet Over Paths solution (MOP)

MFP versus MOP

- Precision: can prove that MOP solution is always more precise than MFP
  $MFP \sqsubseteq MOP$
- Why not use MOP?
- MOP is intractable in practice
  1. Exponential number of paths: for a program consisting of a sequence of N if statement, there will 2^N paths in the control flow graph
  2. Infinite number of paths: for loops in the CFG

Importance of Distributivity

- Property: if transfer functions are distributive, then the solution to the dataflow equations is identical to the meet-over-paths solution

  MFP = MOP

- For distributive transfer functions, can compute the intractable MOP solution using the iterative fixed-point algorithm

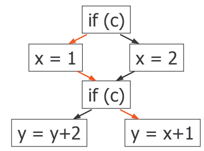
Better Than MOP?

- Is MOP the best solution to the analysis problem?

- MOP computes solution for all path in the CFG
- There may be paths which will never occur in any execution
- So MOP is conservative

- IDEAL = solution which takes into account only paths which occur in some execution

- This is the best solution
- … but it is undecidable



Summary

- Dataflow analysis
  - sets up system of equations
  - iteratively computes MFP
  - Terminates because transfer functions are monotonic and lattice has finite height
- Other possible solutions: FP, MFP, MOP, IDEAL
- All are safe solutions, but some are more precise:

  $FP \sqsubseteq MFP \sqsubseteq MOP \sqsubseteq IDEAL$
- MFP = MOP if distributive transfer functions

- MOP and IDEAL are intractable
- Compilers use dataflow analysis and MFP