CS412/413

Introduction to Compilers

Radu Rugina

Lecture 25: More Dataflow Analysis

31 Mar 06

Dataflow Analysis Framework

- A dataflow analysis framework consists of:
  - A lattice $(L, \sqsubseteq, \sqcap, \top)$ where:
    - L is the dataflow information
    - $\sqsubseteq$ is the ordering relation
    - $\sqcap$ is the merge operation (GLB)
    - $\top$ is the bottom element
  - Transfer functions $F_n : L \rightarrow L$ for each CFG node n
  - Boundary dataflow information d_0
    - Before CFG entry node for a forward analysis
    - After CFG exit node for a backward analysis

Dataflow Equations

- Forward dataflow analysis:

$in[n_0] = d_0$, where n_0 = CFG entry node
$out[n] = F_n(in[n])$, for all n
$in[n] = \sqcap \{out[n'] \mid n' \in pred(n)\}$, for all n

- Backward dataflow analysis:

$out[n_0] = d_0$, where n_0 = CFG exit node
$in[n] = F_n(out[n])$, for all n
$out[n] = \sqcap \{in[n'] \mid n' \in succ(n)\}$, for all n

Solving the Dataflow Equations

- The constraints (forward analysis):

$in[n_0] = d_0$, where n_0 = CFG entry node
$out[n] = F_n(in[n])$, for all n
$in[n] = \sqcap \{out[n'] \mid n' \in pred(n)\}$, for all n

- Solution = the set of all in[n], out[n] for all n's, such that all constraints are satisfied
- Fixed-point algorithm to solve constraints:
  - Initialize $in[n_0]=d_0$
  - Initialize everything else to $\top$
  - Repeatedly enforce constraints
  - Stop when dataflow solution

Worklist Algorithm (Forward)

$in[n_0] = d_0$
$in[n] = \top$, for all $n \neq n_0$
$out[n] = \top$, for all n

$worklist = \{n_0\}$
while ($worklist \neq \emptyset$)
    Remove a node n from the worklist
    out[n] = Fn(in[n])

    for each successor n' :
        $in[n'] = in[n'] \sqcap out[n]$
        if (in[n'] has changed)
            add n' to the worklist

An Implementation

```
void analyzeForward(Method m, DataflowInfo d0) {
  result.put(m.getCFG().getEntryNode(), d0);

  Stack<CFGNode> worklist = new Stack<CFGNode>();
  while (!worlist.isEmpty()) {
    CFGNode n = worklist.pop();
    DataflowInfo in = result.get(n);
    DataflowInfo out = transferFunction(n,in);
    for (CFGNode succ : n.getSuccessors())
      if (merge(succ, out))
        worklist.add(succ);
    }
}
```

An Implementation

```
boolean merge(CFGNode n, DataflowInfo d) {
  DataflowInfo info = result.get(n);
  if (info == null) {
     result.put(n, d.clone());
     return true;
  }
  return info.meet(d);
}
```


Correctness

- Correctness of the worklist algorithm:
  - At the end, all dataflow equations are satisfied
- Agument:
  - Loop maintains the invariant that the constraints

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

    $out[n] = F_n(in[n])$

    hold for all the nodes n not in the worklist
  - At the end, worklist is empty

Transfer Functions

- Transfer functions are required to be monotonic:

  $F : L \rightarrow L$ is monotonic if

  $x \sqsubseteq y$ implies $F(x) \sqsubseteq F(y)$
- Distributivity: function $F : L \rightarrow L$ 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

Termination

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

  $in_{k+1}[n] \sqsubseteq in_k[n]$ and $out_{k+1}[n] \sqsubseteq out_k[n]$
- Proof by induction:
  - Induction basis: true, because we start with bottom element, which is less than everything
  - Induction step: use monotonicity of transfer functions and join operation
- Information forms a chain: $in_1[n] \sqsupseteq in_2[n] \sqsupseteq in_3[n] \ldots$

Chains in Lattices

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

  $x \sqsubseteq y$ or $y \sqsubseteq x$ for any $x, y \in S$
- In other words:

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

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

Termination

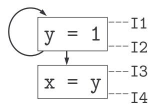
- In the iterative algorithm, for each node n:

  $\{in_1[n], in_2[n], \ldots\}$

  is a chain in the lattice
- If lattice has finite height then there is a number k such that $in_i[n] = in_{i+1}[n]$, for all $i \geq k$ and all n
- If $in_i[n] = in_{i+1}[n]$ then also $out_i[n] = out_{i+1}[n]$
- Algorithm terminates in at most k*N iterations, where N is the number of CFG nodes
- 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}) ∪ {y}`
`I2 = I1 ∪ I3`
`I4 = {x}`

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

Safety and Precision

- Safety: any solution that satisfies the dataflow equations is safe
- Precision: a solution to an analysis problem is more precise if it 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 too imprecise
- In the lattice framework: d1 is more precise than d2 if d1 is higher in the lattice than d2: $d2 \sqsubseteq d1$

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 node n_0
  - For each path $p_k=[n_0, n_1, \ldots, n_k]$ from entry to node n_k:
    $in[p_k] = F_{nk-1}(\ldots (F_{n1}(F_{n0}(d_0))))$
  - Compute solution as
    $in[n] = \sqcup \{ in[p_k] \mid \text{all paths } p_k \text{ from } n_0 \text{ to } n_k \}$
- 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 statements, 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

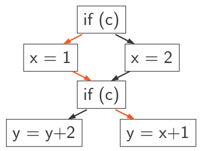
Better Than MOP?

- 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, MOP, IDEAL
- All are safe solutions, but some are more precise:

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

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