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 \((L, \subseteq, \cap, \lor, \top)\) where:
    • \(L\) is the dataflow information
    • \(\subseteq\) is the ordering relation
    • \(\cap\) is the merge operation (GLB)
    • \(\top\) is the bottom element
  – Transfer functions \(F_n : L \to L\) for each CFG node \(n\)
  – Boundary dataflow information \(d\)
    • Before CFG entry node for a forward analysis
    • After CFG exit node for a backward analysis

Dataflow Equations

• Forward dataflow analysis:
  \[
in[n] = d_n, \text{ where } n_0 = \text{ CFG entry node} \\
out[n] = F_n (in[n]), \text{ for all } n \\
in[n] = \cap \{out[n] \mid n_0 \notin \text{pred}(n)\}, \text{ for all } n
\]

• Backward dataflow analysis:
  \[
out[n] = d_n, \text{ where } n_n = \text{ CFG exit node} \\
in[n] = F_n (out[n]), \text{ for all } n \\
in[n] = \cap \{in[n] \mid n_0 \notin \text{succ}(n)\}, \text{ for all } n
\]

Solving the Dataflow Equations

• The constraints (forward analysis):
  \[
in[n] = d_n, \text{ where } n_0 = \text{ CFG entry node} \\
out[n] = F_n (in[n]), \text{ for all } n \\
in[n] = \cap \{in[n] \mid n_0 \notin \text{pred}(n)\}, \text{ for all } n
\]

• 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)

\[
in[n_0] = d_0 \\
in[n] = \top, \text{ for all } n \neq n_0 \\
out[n] = \top, \text{ for all } n \\
worklist = (n_0) \\
while (worklist \neq \emptyset) \\
  \quad \text{Remove a node } n \text{ from the worklist} \\
  \quad \text{out}[n] = F_n(in[n]) \\
  \quad \text{for each successor } n': \\
  \quad \quad \text{in}[n'] = in[n'] \cap out[n] \\
  \quad \quad \text{if (in[n'] has changed) add } n' \text{ to the worklist}
\]

An Implementation

```java
void analyzeForward(Method m, DataflowInfo do) {
  result.puts(m.getCFG().getEntryNode(), do);
  Stack<CFGNode> worklist = new Stack<CFGNode>();
  while (!worklist.isEmpty()) {
    CFGNode n = worklist.pop();
    DataflowInfo info = result.get(n);
    DataflowInfo out = transferFunction(n, info);
    for (CFGNode succ : n.getSuccs())
      if (merge(succ, out))
        worklist.add(succ);
  }
}
```

1
An Implementation

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

Correctness

- Correctness of the worklist algorithm:
  - At the end, all dataflow equations are satisfied
- Algorithm:
  - Loop maintains the invariant that the constraints
    \( \text{in}[n] = \cap \{\text{out}[n] \mid n \in \text{pred}(n)\} \)
    \( \text{out}[n] = F_x(\text{in}[n]) \)
    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
  \( x \subseteq y \) implies \( F(x) \subseteq F(y) \)
- Distributivity: function \( F : L \rightarrow L \) is distributive if
  \( F(x \cap y) = F(x) \cap F(y) \)
- Property: \( F \) is monotonic iff \( F(x \cap y) \subseteq F(x) \cap F(y) \)
  - Any distributive function is monotonic

Termination

- Do these algorithms terminate?
- Key observation: at each iteration, information increases in the lattice
  \( \text{in}_{n+1}[n] \subseteq \text{in}_n[n] \) and \( \text{out}_{n+1}[n] \subseteq \text{out}_n[n] \)
- 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: \( \text{in}_n[n] \supseteq \text{in}_{n+1}[n] \supseteq \text{in}_{n+2}[n] \) ...

Chains in Lattices

- A chain in a lattice \( L \) is a totally ordered subset \( S \) of \( L \):
  \( x \subseteq y \) or \( y \subseteq x \) for any \( x, y \in S \)
- In other words:
  Elements in a totally ordered subset \( S \) can be indexed to form an ascending sequence:
  \( x_1 \sqsupseteq x_2 \sqsupseteq x_3 \sqsupseteq ... \)
- 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 \( n \):
  \( \{\text{in}_0[n], \text{in}_1[n], ...\} \)
  is a chain in the lattice
- If lattice has finite height then there is a number \( k \) such that \( \text{in}_n[n] = \text{in}_{n+k}[n] \) for all \( i \geq k \) and all \( n \)
- If \( \text{in}_n[n] = \text{in}_{n+k}[n] \) then also \( \text{out}_n[n] = \text{out}_{n+k}[n] \)
- Algorithm terminates in at most \( kN \) 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: \[ I_1 = I_2 - (y) \]
\[ I_3 = (I_4 - (x)) \cup \{ y \} \]
\[ I_2 = I_1 \cup I_3 \]
\[ I_4 = \{ x \} \]

Solution 1: \[ I_1 = \{ \}, I_2 = \{ y \}, I_3 = \{ y \}, I_4 = \{ x \} \]
Solution 2: \[ I_1 = \{ x \}, I_2 = \{ x, y \}, I_3 = \{ y \}, I_4 = \{ x \} \]

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 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 too imprecise.

- In the lattice framework: \( d_1 \) is more precise than \( d_2 \) if \( d_1 \) is higher in the lattice than \( d_2 \):

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 \( p = [n_0, n_1, ..., n_k] \) from entry to node \( n_k \):
    \[ i_{n_k}[p_k] = F_{n_k - 1} \left( ... (F_{n_1}(F_{n_0}(n_0))) \right) \]
  - Compute solution as \( i_{n}(p) = \{ i_{n_k}[p_k] \mid \text{all paths } p_k \text{ from } n_k \text{ to } n \} \)
- 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 statements, there will be \( 2^N \) 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:
  - \( FP \subset MFP \subset MOP \subset IDEAL \)
- MFP = MOP if distributive transfer functions
- MOP and IDEAL are intractable
- Compilers use dataflow analysis and MFP