Lecture 10

Interval Analysis

I Basic Idea
II Algorithm
III Optimization and Complexity
IV Comparing interval analysis with iterative algorithms

Reference: Muchnick 7.5-7.7, 8.8
Advanced readings (optional):

Motivation for Studying Interval Analysis

• Exploit the structure of block-structured programs in data flow
• Tie in several concepts studied
  • Use of structure in induction variables, loop invariant
    • motivated by nature of the problem
    • This lecture: can we use structure for speed?
  • Iterative algorithm for data flow
    • This lecture: an alternative algorithm
  • Reducibility
    • all retreating edges of DFST are back edges
    • reducible graphs converge quickly
    • This lecture: algorithm exploits & requires reducibility
• Usefulness in practice
  • Faster for “harder” analyses
  • Useful for analyses related to structure
• Theoretically interesting - better understanding of data flow
I. Big Picture

Basic Idea

• In iterative analysis
  - DEFINITION: Transfer function $F_B$: summarize effect from beginning to end of basic block $B$

• In interval analysis
  - DEFINITION: Transfer function $F_{R,B}$: summarize effect from beginning of $R$ to end of basic block $B$
  - Recursively construct a larger region $R$ from smaller regions construct $F_{R,B}$ from transfer functions for smaller regions until the program is one region
  - Let $P$ be the region for the entire program, and $v$ be initial value at entry node
    • $\text{out}[B] = F_{P,B}(v)$
    • $\text{in}[B] = \land_{B'} \text{out}[B']$, where $B'$ is a predecessor of $B$
II. Algorithm

- (a) Operations on transfer functions
- (b) How to build nested regions?
- (c) How to construct transfer functions that correspond to the larger regions?

(a) Operations on Transfer Functions

- Example: Reaching Definitions

- \( F(x) = \text{Gen} \cup (x - \text{Kill}) \)
- \( F_2(F_1(x)) = \text{Gen}_2 \cup (F_1(x) - \text{Kill}_2) = \text{Gen}_2 \cup (\text{Gen}_1 \cup (x - \text{Kill}_1)) - \text{Kill}_2 \) = \( \text{Gen}_2 \cup (\text{Gen}_1 \cup (x - \text{Kill}_1)) - \text{Kill}_2 \) = \( \text{Gen}_2 \cup \text{Gen}_1 - \text{Kill}_2 \) = \( \text{Gen}_2 \cup (\text{Gen}_1 - \text{Kill}_2) \cup (x - (\text{Kill}_1 \cup \text{Kill}_2)) \)

- \( F_1(x) \land F_2(x) = \text{Gen}_1 \cup (x - \text{Kill}_1) \cup \text{Gen}_2 \cup (x - \text{Kill}_2) = (\text{Gen}_1 \cup \text{Gen}_2) \cup (x - (\text{Kill}_1 \cap \text{Kill}_2)) \)

- \( F^*(x) \leq F^n(x), \forall n \geq 0 \)

- \( x \cup F(x) \cup F(F(x)) \cup ... = x \cup (\text{Gen} \cup (x - \text{Kill})) \cup (\text{Gen} \cup ((\text{Gen} \cup (x - \text{Kill})) - \text{Kill})) \cup ... = \text{Gen} \cup (x - \emptyset) \)
(b) Structure of Nested Regions (An example)

- A region in a flow graph is a set of nodes that
  - includes a header, which dominates all other nodes in a region
- T1-T2 rule (Hecht & Ullman)
  - T1: Remove a loop
    If n is a node with a loop, i.e. an edge n→n, delete that edge
  - T2: Remove a vertex
    If there is a node n that has a unique predecessor, m, then m may consume n by deleting n and making all successors of n be successors of m.

Example

- In reduced graph:
  - each vertex represents a subgraph of original graph (a region).
  - each edge represents an edge in original graph
- Limit flow graph: result of exhaustive application of T1 and T2
  - independent of order of application.
  - if limit flow graph has a single vertex => reducible
- Can define larger regions (e.g. Allen&Cocke’s intervals)
  simple regions => simple composition rules for transfer functions
(c) Transfer Functions for T2 Rule

- **Transfer function**
  
  $F_{R,B}^{}$: summarizes the effect from beginning of R to end of B
  
  $F_{R,in(H2)}^{}$: summarizes the effect from beginning of R to beginning of H2
  
  - Unchanged for blocks B in region $R_1$ ($F_{R,B}^{} = F_{R1,B}^{}$)
  
  - $F_{R,in(H2)}^{} = \bigwedge_p F_{R,p}^{}$, where $p$ is a predecessor of $H_2$
  
  - For blocks B in region $R_2$: $F_{R,B}^{} = F_{R2,B}^{} \cdot F_{R,in(H2)}^{}$

---

**Transfer Functions for T1 Rule**

- **Transfer function** $F_{R,B}^{}$
  
  - $F_{R,in(H)}^{} = (\bigwedge p F_{R1,p}^{})^*$, where $p$ is a predecessor of $H$ in R
  
  - $F_{R,B}^{} = F_{R1,B}^{} \cdot F_{R,in(H)}^{}$
**First Example**

- **R**: region name
- **R’**: region whose header will be subsumed

<table>
<thead>
<tr>
<th>R</th>
<th>T1/T2</th>
<th>R'</th>
<th>F_{R,in(R')}</th>
<th>F_{R,B1}</th>
<th>F_{R,B2}</th>
<th>F_{R,B3}</th>
<th>F_{R,B4}</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>T2</td>
<td>B2</td>
<td>F_{B1}</td>
<td>F_{B1}</td>
<td>F_{B2}</td>
<td>F_{B2}</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>T2</td>
<td>R1</td>
<td>F_{B3}</td>
<td>F_{R1,B1}F_{R2,in(R1)}</td>
<td>F_{R1,B2}F_{R2,in(R1)}</td>
<td>F_{B3}</td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td>T1</td>
<td>R2</td>
<td>(F_{R2B1} \wedge F_{R2B2})^x</td>
<td>F_{R2,B1}F_{R3,in(R2)}</td>
<td>F_{R2,B2}F_{R3,in(R2)}</td>
<td>F_{R2,B3}F_{R3,in(R2)}</td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td>T2</td>
<td>B4</td>
<td>F_{R3B3} \wedge F_{R3B2}</td>
<td>F_{R3,B1}</td>
<td>F_{R3,B2}</td>
<td>F_{R3,B3}</td>
<td>F_{B4}F_{R4,in(B4)}</td>
</tr>
</tbody>
</table>

**III. Complexity of Algorithm**

<table>
<thead>
<tr>
<th>R</th>
<th>T1/T2</th>
<th>R'</th>
<th>F_{R,in(R')}</th>
<th>F_{R,B1}</th>
<th>F_{R,B2}</th>
<th>F_{R,B3}</th>
<th>F_{R,B4}</th>
<th>F_{R,B5}</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>T2</td>
<td>B1</td>
<td>F_{B2}</td>
<td>F_{B1}</td>
<td>F_{B2}</td>
<td>F_{B2}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>T2</td>
<td>R1</td>
<td>F_{B3}</td>
<td>F_{R1,B1}F_{R2,B3}</td>
<td>F_{R1,B2}F_{R2,B3}</td>
<td>F_{B3}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td>T2</td>
<td>R2</td>
<td>F_{B4}</td>
<td>F_{R2,B1}F_{R3,B4}</td>
<td>F_{R2,B2}F_{R3,B4}</td>
<td>F_{R2,B3}F_{R3,B4}</td>
<td>F_{B4}</td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td>T2</td>
<td>R3</td>
<td>F_{B5}</td>
<td>F_{R3,B1}F_{R4,B5}</td>
<td>F_{R3,B2}F_{R4,B5}</td>
<td>F_{R3,B3}F_{R4,B5}</td>
<td>F_{B4}F_{R4,B5}</td>
<td>F_{B5}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R</th>
<th>F_{R,in(R)}</th>
<th>B</th>
<th>F_{R,B}</th>
</tr>
</thead>
<tbody>
<tr>
<td>R4</td>
<td>F_{R4,in(R)}</td>
<td>B5</td>
<td>F_{B5}</td>
</tr>
<tr>
<td>R3</td>
<td>F_{B5}</td>
<td>B4</td>
<td>F_{B4}</td>
</tr>
<tr>
<td>R2</td>
<td>F_{B4}</td>
<td>B3</td>
<td>F_{B3}</td>
</tr>
<tr>
<td>R1</td>
<td>F_{B1}</td>
<td>B2</td>
<td>F_{B2}</td>
</tr>
<tr>
<td>B1</td>
<td>F_{B1}</td>
<td>B1</td>
<td>F_{B1}</td>
</tr>
</tbody>
</table>
 Optimization

• Let m = number of edges, n = number of nodes

• Ideas for optimization
  • If we compute $F_{R,B}$ for every region B is in, then it is very expensive
  • We are ultimately only interested in the entire region (E); we need to compute only $F_{E,B}$ for every B.
    - There are many common subexpressions between $F_{E,B_1}$, $F_{E,B_2}$, ...
    - Number of $F_{E,B}$ calculated = m
  • Also, we need to compute $F_{R,in(R')}$, where R’ represents the region whose header is subsumed.
    - Number of $F_{R,B}$ calculated, where R is not final = n

• Total number of $F_{R,B}$ calculated: (m + n)
  • Data structure keeps “header” relationship
    • Practical algorithm: $O(m \log n)$
    • Complexity: $O(m \alpha(m,n))$, $\alpha$ is inverse Ackermann function

 Reducibility

• If no T1, T2 is applicable before graph is reduced to single node
  • split node and continue
• Worst case: exponential
• Most graphs (including GOTO programs) are reducible
IV. Comparison with Iterative Data Flow

• Applicability
  - Definitions of F* can make technique more powerful than iterative algorithms
  - Backward flow -- reverse graph is not typically reducible. Requires more effort to adapt to backward flow than iterative alg.
  - More important for interprocedural optimization

• Speed
  - Irreducible graphs
    - Iterative algorithm can process irreducible parts uniformly
    - Serious “irreducibility” can be slow with elimination
  - Reducible graph & Cycles do not add information (common)
    - Iterative: (depth + 2) passes
      depth is 2.75 average, independent of code length
    - Elimination: Theoretically almost linear, typically O(m log n)
  - Reducible & Cycles add information
    - Iterative takes longer to converge
    - Elimination remains the same