Motivation for Studying Region-Based 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: on 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. Basic Idea

Basic Idea

- In Iterative Analysis:
  - DEFINITION: Transfer function \( F_B \):
    summarize effect from beginning to end of basic block \( B \)
- In Region-Based 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
    \[ \text{out}(B) = F_B(v) \]
    \[ \text{in}(B') = A_{B'} \cdot \text{out}(B') \], where \( B' \) is a predecessor of \( B \)
II. Algorithm

1. Operations on transfer functions
2. How to build nested regions?
3. How to construct transfer functions that correspond to the larger regions?

1. Operations on Transfer Functions

- Example: Reaching Definitions
  - $F(x) = \text{Gen} \cup (x - \text{Kill})$
  - $F_2(F_1(x)) = \text{Gen}_2 \cup (F_1(x) - \text{Kill}_2)$
    \[= \text{Gen}_2 \cup (\text{Gen}_1 \cup (x - \text{Kill}_1)) - \text{Kill}_2)\]
  - $F_1(x) \land F_2(x) = \text{Gen}_1 \cup (x - \text{Kill}_1) \cup \text{Gen}_2 \cup (x - \text{Kill}_2)$
    \[= (\text{Gen}_1 \cup \text{Gen}_2) \cup (x - (\text{Kill}_1 \cap \text{Kill}_2))\]

- $F^*(x) \leq F^n(x)$, $n \geq 0$
  \[= x \cup F(x) \cup F(F(x)) \cup ...\]
  \[= x \cup (\text{Gen} \cup (x - \text{Kill}) \cup (\text{Gen} \cup ((\text{Gen} \cup (x - \text{Kill}) - \text{Kill})) \cup ...\]
  \[= \text{Gen} \cup (x - \emptyset)\]

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 $n \rightarrow n$, 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
**Transfer function**

- \( F_{R,B} \): summarizes the effect from beginning of \( R \) to end of \( B \)
- \( F_{R,H2} \): summarizes the effect from beginning of \( R \) to beginning of \( H2 \)
  - Unchanged for blocks \( B \) in region \( R \) if \( F_{R,B} = F_{R1,B} \)
  - \( F_{R,H2} = A \circ F_{R,B} \), where \( p \) is a predecessor of \( H \)
- For blocks \( B \) in region \( R \): \( F_{R,B} = F_{R2,B} \cdot F_{R1,in(H)} \)

\[ F_{R,B} = \left( \wedge F_{R2,B} \right) \cdot F_{R1,in(H)} \]

\[ F_{R,B} = F_{R1,B} \cdot F_{R2,in(H)} \]

\[ F_{R,B} = F_{R1,B} \cdot F_{R2,in(H)} \]

\[ F_{R,B} = F_{R1,B} \cdot F_{R2,in(H)} \]

- \( R \): region name
- \( R' \): region whose header will be subsumed
III. Complexity of Algorithm

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>B2</td>
<td>B3</td>
<td>B4</td>
<td>B5</td>
<td>B'1</td>
<td>B'2</td>
</tr>
<tr>
<td>R1</td>
<td>R2</td>
<td>R3</td>
<td>R4</td>
<td>R5</td>
<td>R'1</td>
<td>R'2</td>
</tr>
<tr>
<td>fB1</td>
<td>fB2</td>
<td>fB3</td>
<td>fB4</td>
<td>fB5</td>
<td>fR'1</td>
<td>fR'2</td>
</tr>
</tbody>
</table>

Optimization

- Let m = number of edges, n = number of nodes
- Ideas for optimization
  - If we compute $F_{EA}$ 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 $F_{EA}$ for every $B_i$.
    - There are many common subexpressions between $F_{EA}$ and $F_{EB}$...
    - Number of $F_{EA}$ calculated = m
  - Also, we need to compute $F_{EA(B')}$ where B' represents the region whose header is subsumed.
    - Number of $F_{EA}$ calculated, where $B_i$ is not final = n
- Total number of $F_{EA}$ 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

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 algorithm
  - More important for interprocedural optimization
- Speed
  - Irreducible graphs
    - Iterative algorithm can process irreducible parts uniformly
  - Reducible graph & Cycles do not add information (common)
    - Iterative: $O(m + 2)$ passes
    - Depth is 2.75 average, independent of code length
  - Region-based analysis: Theoretically almost linear, typically $O(m \log n)$
  - Reducible & Cycles add information
    - Iterative takes longer to converge
    - Region-based analysis remains the same

Reducibility

- If no $T_1$, $T_2$ is applicable before graph is reduced to single node, then split node and continue
- Worst case: exponential
- Most graphs (including GOTO programs) are reducible