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
  - 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

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 $F$ be the region for the entire program,
  and $v$ be initial value at entry node
  - $\text{out}(B) = F_{R,F}(v)$
  - $\text{in}[B] = \land_{B'} \text{out}[B']$, 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
  
  $F(x) = \text{Gen} \cup (x - \text{Kill})$
  
  $F_2(F_1(x)) = \text{Gen}_2 \cup (F_1(x) - \text{Kill}_2)
  = \text{Gen}_2 \cup (\text{Gen}_1 \cup (x - \text{Kill}_1)) - \text{Kill}_2
  = \text{Gen}_2 \cup (\text{Gen}_1 - \text{Kill}_2) \cup (x - (\text{Kill}_1 \cup \text{Kill}_2))$

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

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

2. 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 $n\rightarrow n$, 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 \( \Rightarrow \) reducible
- Can define larger regions (e.g. Allen&Cocke's intervals)
  - simple regions \( \Rightarrow \) simple composition rules for transfer functions
### Transfer Functions for T2 Rule

- **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 $R_1$: $F_{R,B} = F_{R_1,B}$
  - $F_{R,(H2)} = \bigwedge P F_{R,P}$, where $p$ is a predecessor of $H2$
  - For blocks $B$ in region $R_2$: $F_{R,B} = F_{R_2,B} \cdot F_{R,(H2)}$

### Transfer Functions for T1 Rule

- **Transfer Function** $F_{R,B}$:
  - $F_{R,(H)} = (\bigwedge P F_{R,P})^*$, where $p$ is a predecessor of $H$ in $R$
  - $F_{R,B} = F_{R_1,B} \cdot F_{R,(H)}$

### First Example

<table>
<thead>
<tr>
<th>R</th>
<th>T0</th>
<th>T1</th>
<th>R'</th>
<th>$F_{R,B_1}$</th>
<th>$F_{R,B_2}$</th>
<th>$F_{R,B_3}$</th>
<th>$F_{R,B_4}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>T2</td>
<td>B2</td>
<td>R2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>R2</td>
<td>T2</td>
<td>B1</td>
<td>R1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>R3</td>
<td>T1</td>
<td>R2</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>R4</td>
<td>T2</td>
<td>B4</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- **R**: region name
- **R’**: region whose header will be subsumed
III. Complexity of Algorithm

<table>
<thead>
<tr>
<th>R</th>
<th>F_{R,B}</th>
<th>F_{R,B}</th>
<th>F_{R,B}</th>
<th>F_{R,B}</th>
<th>F_{R,B}</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>T_1</td>
<td>T_2</td>
<td>R'</td>
<td>F_{R,B}</td>
<td>F_{R,B}</td>
</tr>
<tr>
<td>R_2</td>
<td>T_2</td>
<td>R_1</td>
<td>F_{R,B}</td>
<td>F_{R,B}</td>
<td>F_{R,B}</td>
</tr>
<tr>
<td>R_3</td>
<td>T_3</td>
<td>R_2</td>
<td>F_{R,B}</td>
<td>F_{R,B}</td>
<td>F_{R,B}</td>
</tr>
<tr>
<td>R_4</td>
<td>T_4</td>
<td>R_3</td>
<td>F_{R,B}</td>
<td>F_{R,B}</td>
<td>F_{R,B}</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}, F_{E,B}...
    - 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_{E,B} calculated, where R is not final = n
  - Total number of F_{E,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

Reductibility

- If no T1, T2 is applicable before graph is reduced to single node, then 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 algorithm.
    - More important for interprocedural optimization
- Speed
  - Irreducible graphs
    - Iterative algorithm can process irreducible parts uniformly
    - Serious “irreducibility” can be slow with region-based analysis
  - Reducible graph & Cycles do not add information (common)
    - Iterative: (depth + 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