Lecture 13

Region-Based Analysis

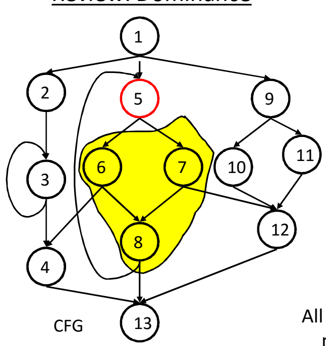
I. Basic Idea
II. Algorithm
III. Optimization and Complexity
IV. Comparing region-based analysis with iterative algorithms

[ALSU 9.7]

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: 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**

Review: Dominance



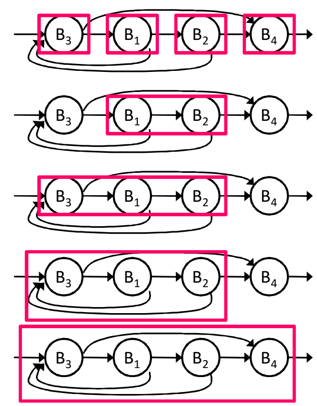
CFG

All paths to 6, 7, or 8 must visit 5 first

x strictly dominates w (x sdom w) iff impossible to reach w without passing through x first

x dominates w (x dom w) iff x sdom w OR x = w

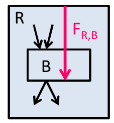
I. Big Picture



A **region** in a flow graph is a set of nodes with a **header** that dominates all other nodes in a region

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
    – out[B] = $F_{P,B}$ (v)
    – in [B] = $\wedge_{B'}$ 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?

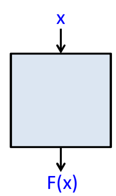
1. Operations on Transfer Functions

Example: **Reaching Definitions**

- Transfer function over a block:

$$F(x) = \mathbf{Gen} \cup (x - \mathbf{Kill})$$

Input parameters



- Resulting transfer functions (after operations) must be consistent with this form:
  – same equation
  – updated values for **Gen** and **Kill** set parameters

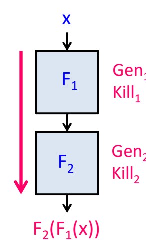
Operations on Transfer Functions: Composition

$F_2 \circ F_1$

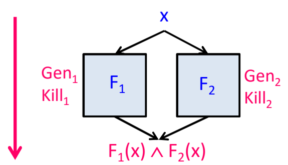
$$F_2(F_1(x)) = Gen_2 \cup (F_1(x) - Kill_2)$$
$$= Gen_2 \cup (Gen_1 \cup (x - Kill_1)) - Kill_2)$$
$$= Gen_2 \cup (Gen_1 - Kill_2) \cup (x - (Kill_1 \cup Kill_2))$$

Gen set after composition

Kill set after composition



Operations on Transfer Functions: Meet

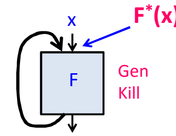


(Recall that for Reaching Definitions, $\wedge = \cup$.)

$$F_1(x) \wedge F_2(x) = Gen_1 \cup (x - Kill_1) \cup Gen_2 \cup (x - Kill_2)$$
$$= (Gen_1 \cup Gen_2) \cup (x - (Kill_1 \cap Kill_2))$$

Gen set after $\wedge$ **Kill** set after $\wedge$

Operations on Transfer Functions: Closure



New Feature!
(We don't have this in iterative data flow analysis.)

What is the value at the input of the block?

- *including* the possible effects of the back edge
  → it may iterate 0, 1, 2, …, ∞ number of times

For Reaching Definitions

$$F^*(x) = \wedge_{\{n \geq 0\}} F^n(x)$$
$$= x \wedge F(x) \wedge F(F(x)) \wedge \ldots$$
$$= x \cup (Gen \cup (x - Kill)) \cup (Gen \cup ((Gen \cup (x - Kill)) - Kill)) \cup \ldots$$
$$= Gen \cup (x - \varnothing)$$

Gen set   **Kill** set (after closure)

Recap of Operations on Transfer Functions

For **Reaching Definitions:**

- Transfer Function (**F(x)**):

  $F(x) = \mathbf{Gen} \cup (x - \mathbf{Kill})$

- Composition ($F_2(F_1(x))$):

  Gen = $Gen_2 \cup (Gen_1 - Kill_2)$
  Kill = $Kill_1 \cup Kill_2$

- Meet: ($F_1(x) \wedge F_2(x)$):

  Gen = $Gen_1 \cup Gen_2$
  Kill = $Kill_1 \cap Kill_2$

- Closure: ($F^*(x)$):

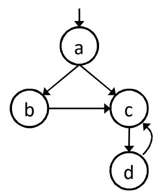
  Gen = Gen
  Kill = $\varnothing$

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) for Reducible Flow Graphs**
  - 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

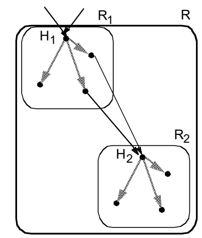
T1: Remove a n->n loop
T2: Remove a vertex w/unique predecessor

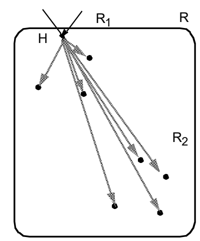


- In reduced graph:
  - each vertex represents a subgraph of original graph (a **region**).
  - each edge represents an edge in original graph
- **Limit flow graph**: result of exhaustive application of T1 and T2
  - independent of order of application
  - reducible flow graph: limit flow graph has a single vertex

3. Transfer Functions for T2 Rule

T2: Remove a vertex w/unique predecessor

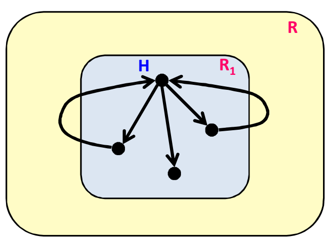




- **Transfer function**
  $F_{R,B}$: summarizes the effect from beginning of R to end of B
  $F_{R,in(H2)}$: summarizes the effect from beginning of R to beginning of H2
  - Unchanged for blocks B in region R_1 ($F_{R,B} = F_{R1,B}$)
  - $F_{R,in(H2)} = \wedge_p F_{R,p}$, where p is a predecessor of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

Transfer Functions for T1 Rule

T1: Remove a n->n loop

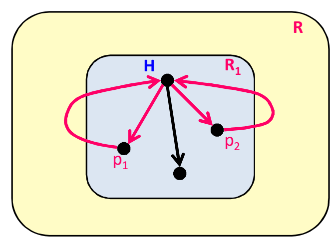


R: new region
(subsumes back edges from $R_1 \rightarrow R_1$)

Observations:
- the header of R_1 (i.e. **H**) is also the header of **R**
- we already know how to get from **H** to **B** for every block B in R_1: i.e. $F_{R1,B}$
  - this will be the *last step* in getting from the new **R** to **B** (**composition**)
- what's new: we need to get from **R** to the input of **H**, *including back edges!*
  - this involves both **meet** ($\wedge$) and **closure** (*) operations

Transfer Functions for T1 Rule

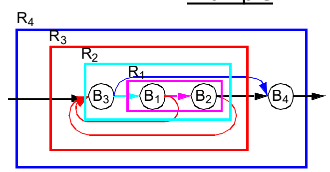
T1: Remove a n->n loop

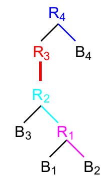


R: new region
(subsumes back edges from $R_1 \rightarrow R_1$)

- **Transfer Function $F_{R,B}$**
  - $F_{R,in(H)} = (\wedge_p F_{R1,p})^*$, where p is a predecessor of H in R
  - $F_{R,B} = F_{R1,B} \circ F_{R,in(H)}$

Example

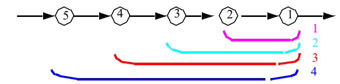




R	Rule	R'	$F_{R,in(R')}$	$F_{R,B1}$	$F_{R,B2}$	$F_{R,B3}$	$F_{R,B4}$
R_1	T_2	B_2					
R_2	T_2	R_1					
R_3	T_1	R_2					
R_4	T_2	B_4					

- R: region name; R': region whose header will be subsumed
- T_2: $F_{R,in(R')} = \wedge_P F_{R,P}$, $p \in pred(HR')$; $F_{R,BR'} = F_{R',BR'} \circ F_{R,in(R')}$
- T_1: $F_{R,in(R')} = (\wedge_P F_{R',P})^*$, $p \in pred(HR')$; $F_{R,B} = F_{R',B} \circ F_{R,in(R')}$

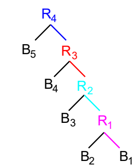
III. Complexity of Algorithm



R	Rule	R'	$F_{R,in(R')}$	$F_{R,B1}$	$F_{R,B2}$	$F_{R,B3}$	$F_{R,B4}$	$F_{R,B5}$
R_1	T_2	B_1	F_{B2}	$F_{B1} \cdot F_{B2}$	F_{B2}			
R_2	T_2	R_1	F_{B3}	$F_{R1,B1} \cdot F_{B3}$	$F_{R1,B2} \cdot F_{B3}$	F_{B3}		
R_3	T_2	R_2	F_{B4}	$F_{R2,B1} \cdot F_{B4}$	$F_{R2,B2} \cdot F_{B4}$	$F_{R2,B3} \cdot F_{B4}$	F_{B4}	
R_4	T_2	R_3	F_{B5}	$F_{R3,B1} \cdot F_{B5}$	$F_{R3,B2} \cdot F_{B5}$	$F_{R3,B3} \cdot F_{B5}$	$F_{B4} \cdot F_{B5}$	F_{B5}

R	$F_{R4,in(R)}$
R_4	I
R_3	$F_{B5} \cdot F_{R4,in(R4)}$
R_2	$F_{B4} \cdot F_{R4,in(R3)}$
R_1	$F_{B3} \cdot F_{R4,in(R2)}$
B_1	$F_{B2} \cdot F_{R4,in(R1)}$

B	$F_{R4,B}$
B_5	$F_{B5} \cdot I$
B_4	$F_{B4} \cdot F_{R4,in(R3)}$
B_3	$F_{B3} \cdot F_{R4,in(R2)}$
B_2	$F_{B2} \cdot F_{R4,in(R1)}$
B_1	$F_{B1} \cdot F_{R4,in(B1)}$

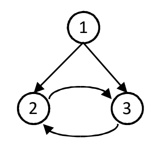


Optimization

- **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 $F_{E,B1}$, $F_{E,B2}$, ...
    - 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: $O(m\alpha(m,n))$, α is inverse Ackermann function

Reducibility

T1: Remove a n->n loop
T2: Remove a vertex w/unique predecessor



- If no T1, T2 is applicable before graph is reduced to single node, then **split node** (make k copies of node, one per predecessor) 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

Wednesday's Class

- Register Allocation [ALSU 8.8]
- Assignment #2 due Wednesday midnight