Lecture 19

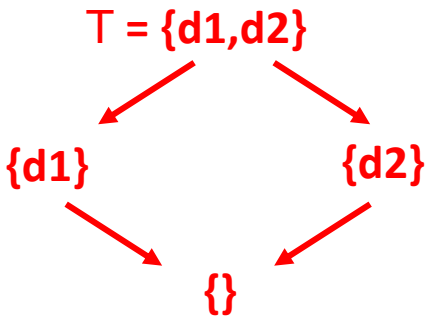
Region-Based Analysis

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

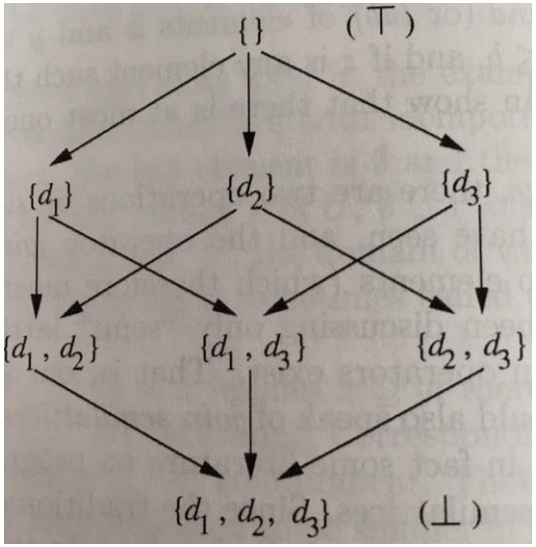
[ALSU 9.7]

Review: Iterative Data Flow Analysis

- **Semi-lattice**
  - set of values **V**
  - meet operator ∧
  - Top T
  - finite descending chain?



Meet Operator:
Intersection



Meet Operator:
Union

Review: Iterative Data Flow Analysis

- **Semi-lattice**
  - set of values **V**
  - meet operator ∧
  - Top T
  - finite descending chain?
- **Transfer functions**
  - function of a basic block **f: V → V**
  - closed under composition
  - meet-over-paths **MOP**
  - monotone
  - distributive?

For each node *n*: MOP(*n*) = $\wedge f_{p_i}(\top)$, **for all paths p_i in data-flow graph reaching *n*.**

If data flow framework is monotone (i.e., x ≤ y implies f(x) ≤ f(y)) then if the algorithm converges, IN[b] ≤ MOP[b] *, so analysis is ? safe.

Data flow framework (monotone) converges if its lattice has ? a finite descending chain.

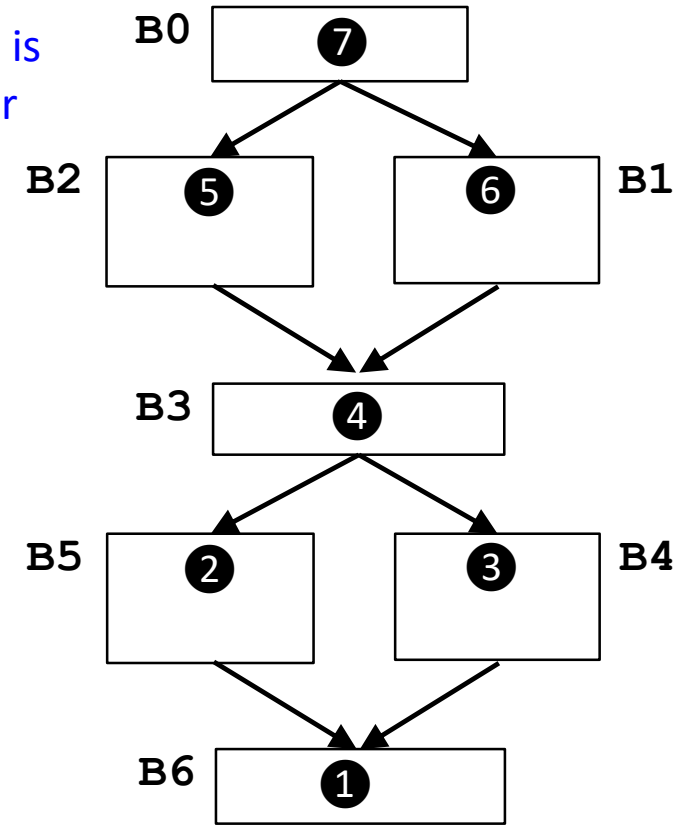
If data flow framework is distributive (i.e., f(x ∧ y) = f(x) ∧ f(y)) then if the algorithm converges, IN[b] = MOP[b] *, so ? precision is high.

* for backward analysis OUT[b]

Review: Iterative Data Flow Analysis

- **Semi-lattice**
  - set of values **V**
  - meet operator ∧
  - Top ⊤
  - finite descending chain?
- **Transfer functions**
  - function of a basic block **f: V → V**
  - closed under composition
  - meet-over-paths **MOP**
  - monotone
  - distributive?
- **Algorithm**
  - initialization step (entry/exit, other nodes)
  - repeated passes until find fixedpoint solution
  - visit order of each pass: **rPostOrder**

B0,B1,…,B6 is rPostOrder

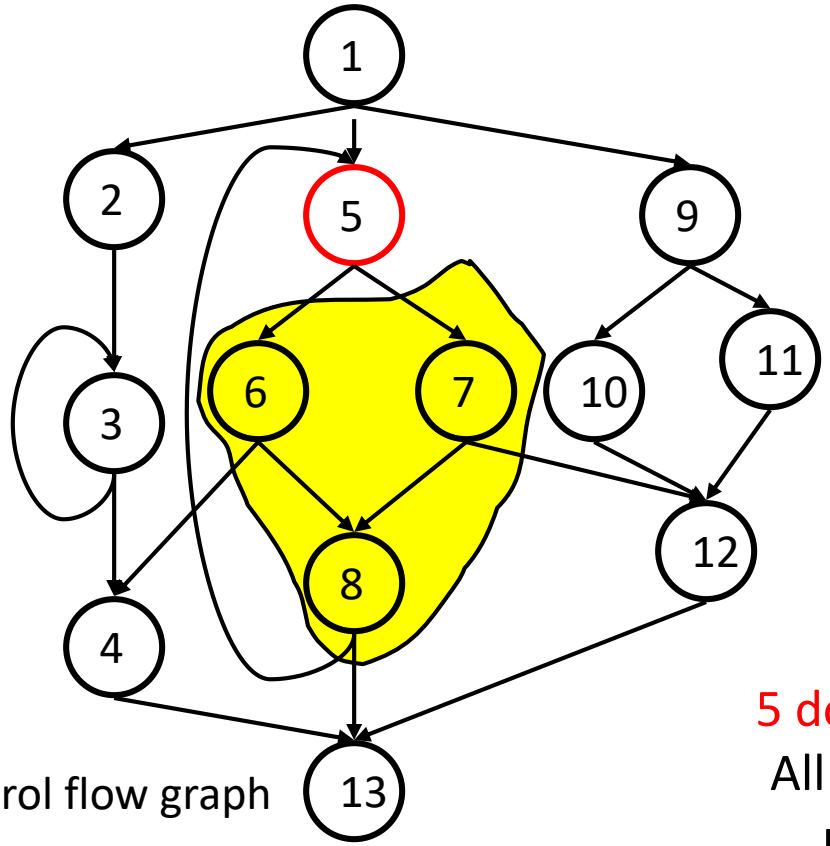


Number of passes = number of back edges in any acyclic path + 2

Region-Based Analysis: Motivation

- **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 DFS Tree are back edges (t->h, h dominates t)
    - reducible graphs converge quickly
    - *This lecture: algorithm exploits & requires reducibility*
- **Usefulness in practice**
  - Faster for "harder" analyses
    - e.g., where paths have cycles that may change the data-flow values
  - Useful for analyses related to structure
    - e.g., interprocedural analysis
  - But not so well-suited to backward analyses
- **Theoretically interesting: better understanding of data flow**

Review: Dominance



Control flow graph

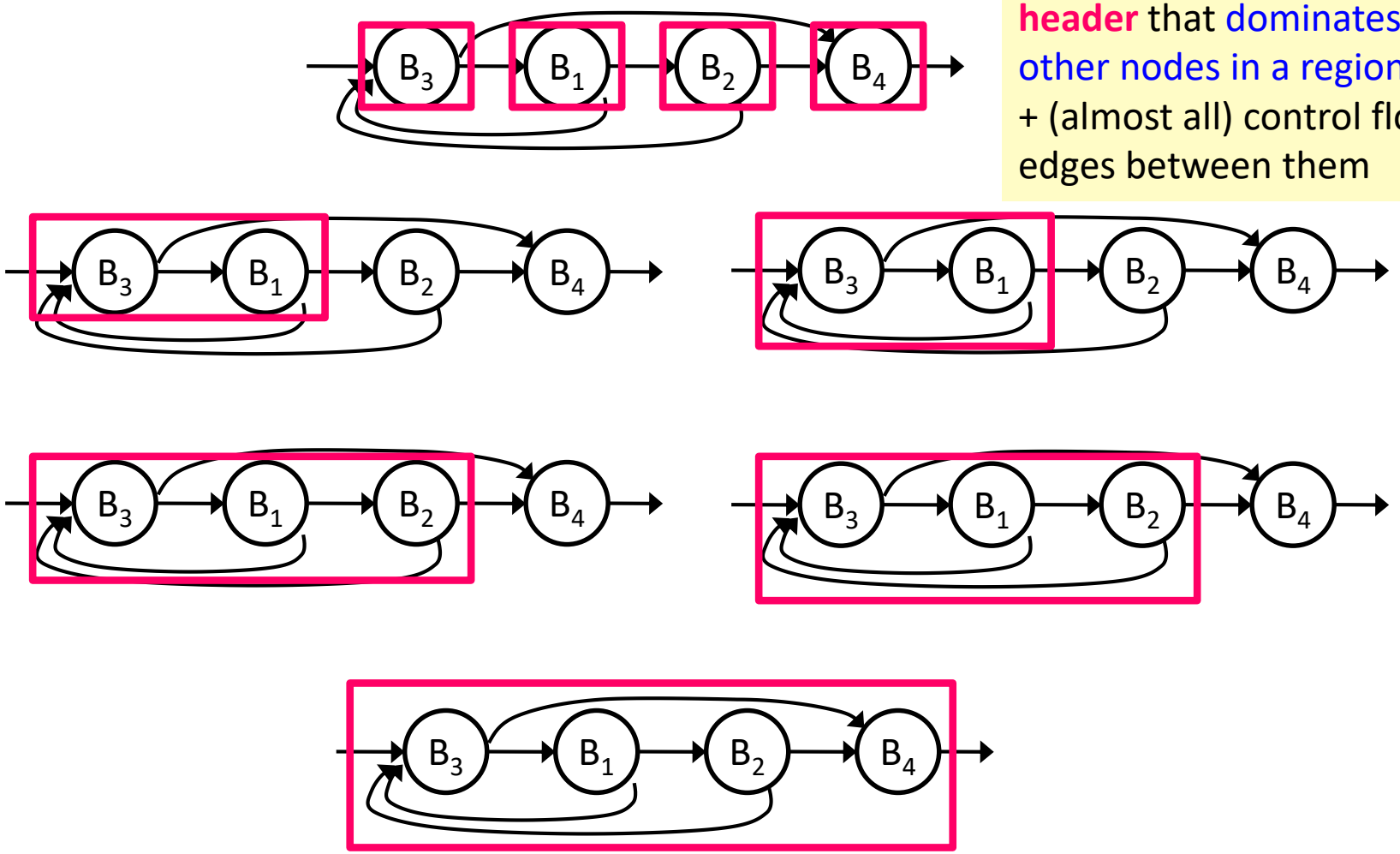
5 dominates {5,6,7,8}:
All paths to 6, 7, or 8
must visit 5 first

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

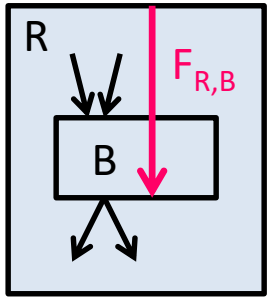
I. Region Analysis: Big Picture

A **region** in a flow graph is a set of nodes with a **header** that dominates all other nodes in a region + (almost all) control flow edges between them

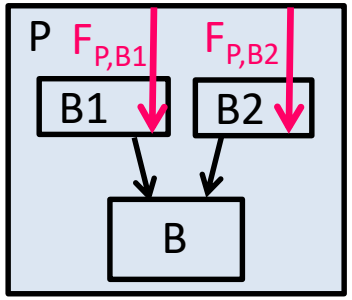


Basic Idea

- **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 region 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



In[B] = $F_{P,B1}(v) \wedge F_{P,B2}(v)$

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?

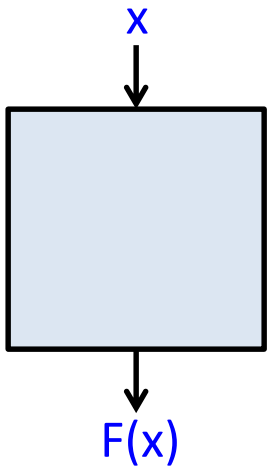
1. Operations on Transfer Functions

Example: **Reaching Definitions**

- Transfer function over a block:

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

Input parameters



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

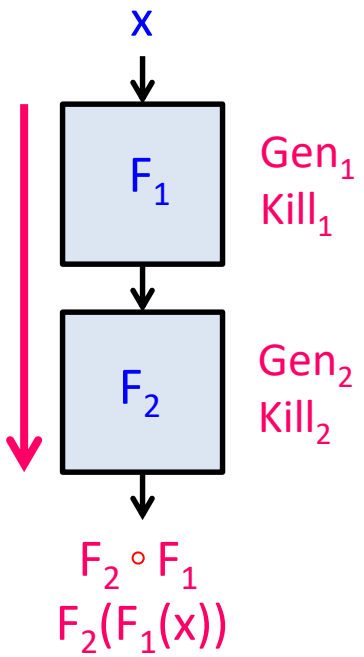
Operations on Transfer Functions: Composition

$F_2 \circ F_1$

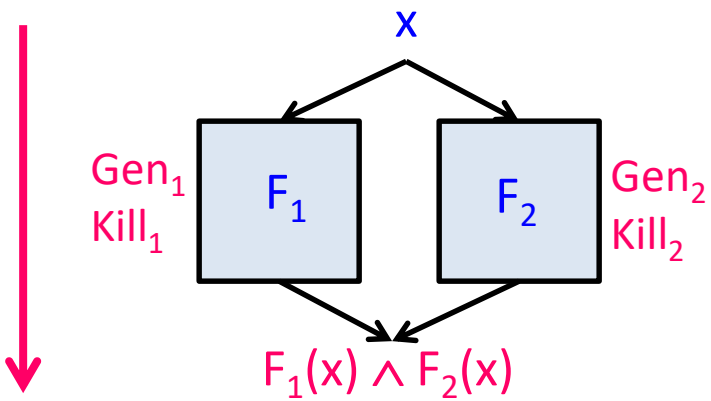
$$F_2(F_1(x)) = Gen_2 \cup (F_1(x) - Kill_2)$$
$$= Gen_2 \cup (Gen_1 \cup (x - Kill_1)) - Kill_2)$$
$$= Gen_2 \cup (Gen_1 - Kill_2) \cup (x - (Kill_1 \cup Kill_2))$$

Gen set after composition

Kill set after composition



Operations on Transfer Functions: Meet



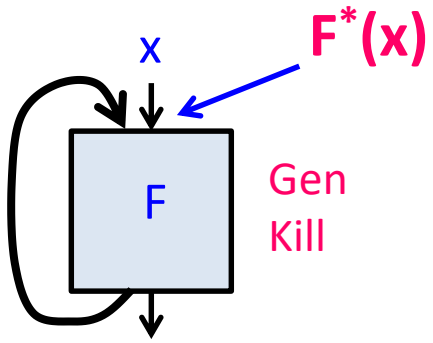
(Recall that for Reaching Definitions, $\wedge = \cup$.)

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

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

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. Enables better summarization of the effect of the loop.)

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

$F^*(x) = \wedge_{\{n \geq 0\}} F^n(x)$

$= x \wedge F(x) \wedge F(F(x)) \wedge \ldots$

For Reaching Definitions

$= x \cup (Gen \cup (x - Kill)) \cup (Gen \cup ((Gen \cup (x - Kill)) - Kill)) \cup \ldots$

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

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

Recap of Operations on Transfer Functions

For **Reaching Definitions:**

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

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

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

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

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

$$\textbf{Gen} = Gen_1 \cup Gen_2$$
$$\textbf{Kill} = Kill_1 \cap Kill_2$$

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

$$\textbf{Gen} = Gen$$
$$\textbf{Kill} = \varnothing$$

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

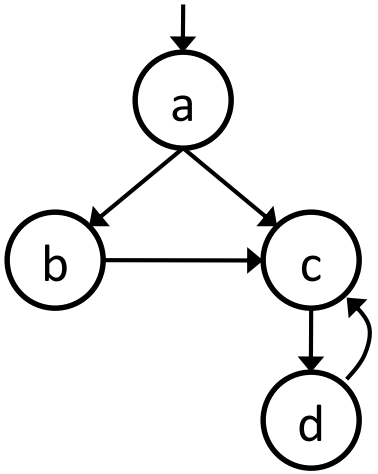
- **T1-T2 rule (Hecht & Ullman) for Flow Graphs**

  - T1: Remove a loop
    If n is a node with a loop, i.e. an edge n->n, delete that edge (all such edges for n)

  - 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

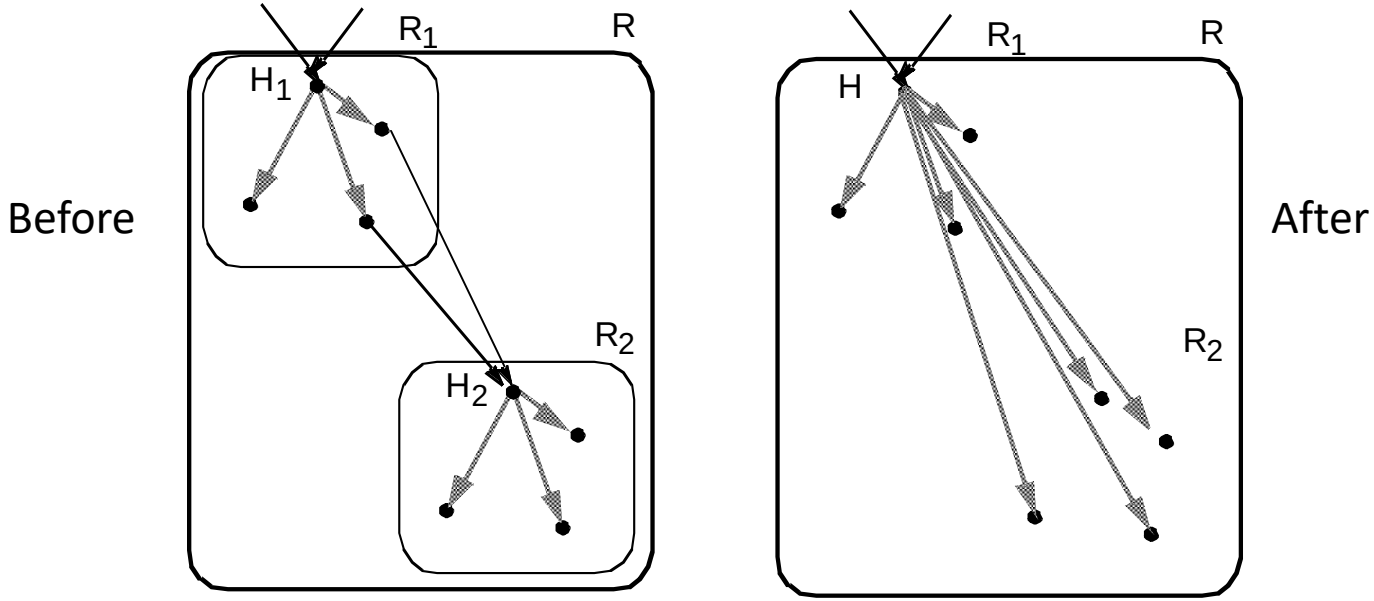
T1: Remove n->n loops
T2: Remove a vertex w/unique predecessor



- 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



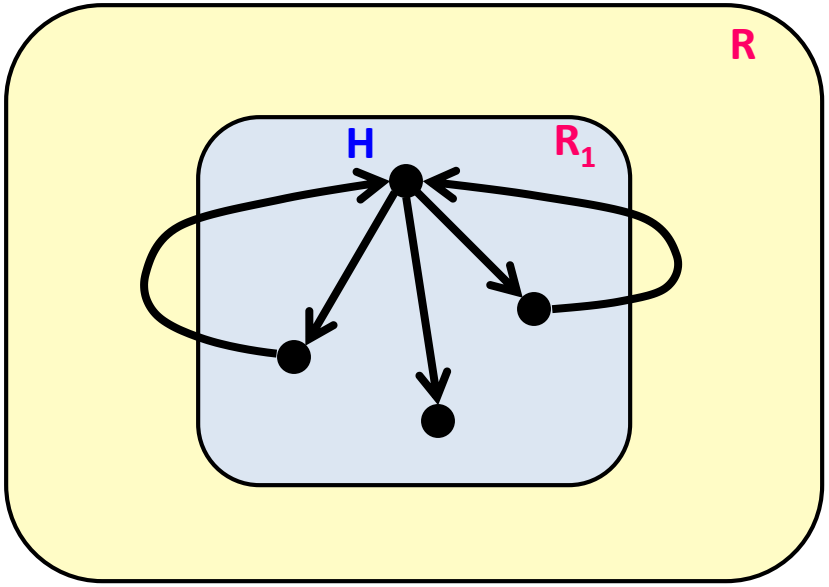
- **Transfer function**

  $F_{R,B}$: 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 block of H_2
  - For blocks B in region R_2: $F_{R,B} = F_{R2,B} \circ F_{R,in(H2)}$

4. Transfer Functions for T1 Rule

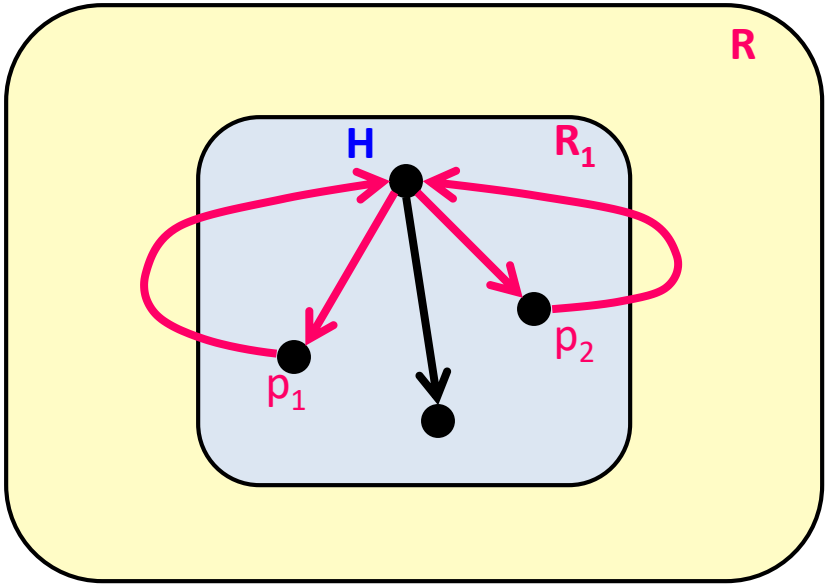


R: new region
(subsumes back edges from $R_1 \rightarrow R_1$)

Observations:

- the header of $\mathbf{R_1}$ (i.e. **H**) is also the header of **R**
- we already know how to get from **H** to **B** for every block B in $\mathbf{R_1}$: i.e. $\mathbf{F_{R1,B}}$
  - 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

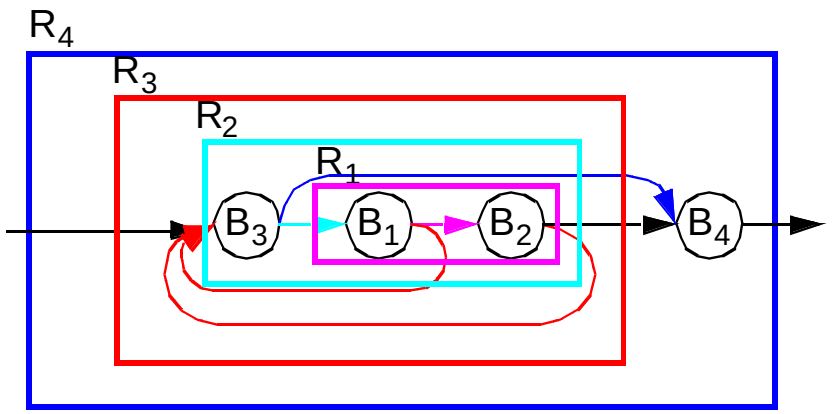
Transfer Functions for T1 Rule

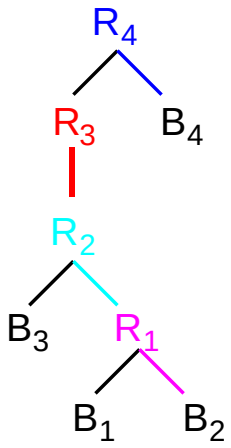


R: new region
(subsumes back edges from $R_1 \rightarrow R_1$)

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

Example

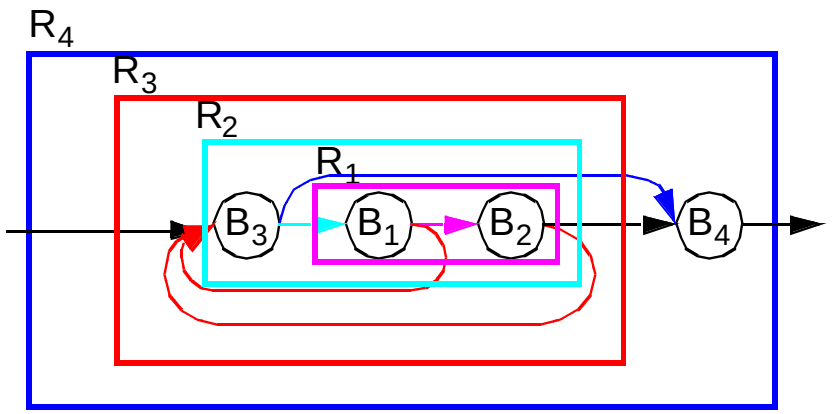


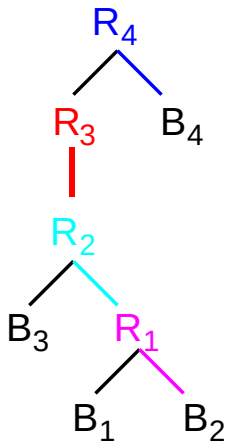


R	Rule	R'	$F_{R,in(R')}$	$F_{R,B1}$	$F_{R,B2}$	$F_{R,B3}$	$F_{R,B4}$
R_1	T_2	B_2					
R_2	T_2	R_1					
R_3	T_1	R_2					
R_4	T_2	B_4					

- R: region name; R': region whose header will be subsumed; R'': T_2's other region
- **T_2**: $F_{R,in(R')} = \wedge_P F_{R,P}$, P ∈ pred(header of R'); for B ∈ R': $F_{R,B} = F_{R',B} \circ F_{R,in(R')}$; for B ∈ R'': $F_{R,B} = F_{R'',B}$
- **T_1**: $F_{R,in(R')} = (\wedge_P F_{R',P})^*$, P ∈ pred(header of R'); $F_{R,B} = F_{R',B} \circ F_{R,in(R')}$

Example

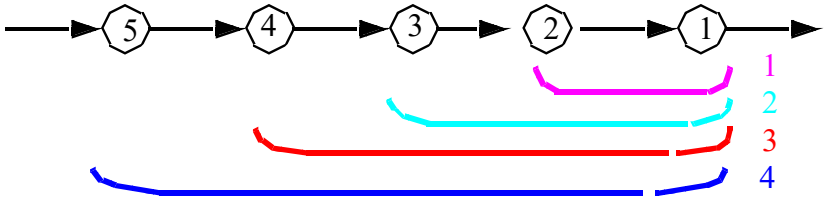




R	Rule	R'	$F_{R,in(R')}$	$F_{R,B1}$	$F_{R,B2}$	$F_{R,B3}$	$F_{R,B4}$
R_1	T_2	B_2	$F_{R1,B1} = F_{B1}$	$F_{R1,B1} = F_{B1}$	$F_{B2} \circ F_{R1,in(B2)} = F_{B2} \circ F_{B1}$		
R_2	T_2	R_1	$F_{R2,B3} = F_{B3}$	$F_{R1,B1} \circ F_{R2,B3} = F_{B1} \circ F_{B3}$	$F_{R1,B2} \circ F_{B3} = F_{B2} \circ F_{B1} \circ F_{B3}$	F_{B3}	
R_3	T_1	R_2	$(F_{R2,B1} \wedge F_{R2,B2})^*$	$F_{R2,B1} \circ F_{R3,in(R2)}$	$F_{R2,B2} \circ F_{R3,in(R2)}$	$F_{B3} \circ F_{R3,in(R2)}$	
R_4	T_2	B_4	$F_{R3,B3} \wedge F_{R3,B2}$	$F_{R3,B1}$	$F_{R3,B2}$	$F_{R3,B3}$	$F_{B4} \circ F_{R4,in(B4)}$

- R: region name; R': region whose header will be subsumed; R'': T_2's other region
- **T_2**: $F_{R,in(R')} = \wedge_P F_{R,P}$, P ∈ pred(header of R'); for B ∈ R': $F_{R,B} = F_{R',B} \circ F_{R,in(R')}$; for B ∈ R'': $F_{R,B} = F_{R'',B}$
- **T_1**: $F_{R,in(R')} = (\wedge_P F_{R',P})^*$, P ∈ pred(header of R'); $F_{R,B} = F_{R',B} \circ F_{R,in(R')}$

III. Complexity of Algorithm



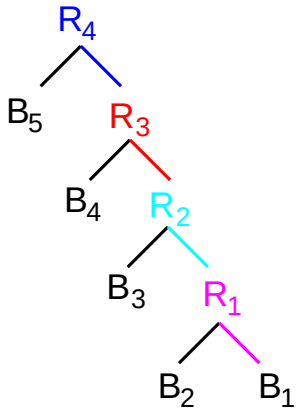
R	Rule	R'	$F_{R,in(R')}$	$F_{R,B1}$	$F_{R,B2}$	$F_{R,B3}$	$F_{R,B4}$	$F_{R,B5}$
R_1	T_2	B_1	F_{B2}	$F_{B1} \circ F_{B2}$	F_{B2}			
R_2	T_2	R_1	F_{B3}	$F_{R1,B1} \circ F_{B3}$	$F_{R1,B2} \circ F_{B3}$	F_{B3}		
R_3	T_2	R_2	F_{B4}	$F_{R2,B1} \circ F_{B4}$	$F_{R2,B2} \circ F_{B4}$	$F_{R2,B3} \circ F_{B4}$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

$O(n^2)$ entries

$O(n)$ entries

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

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

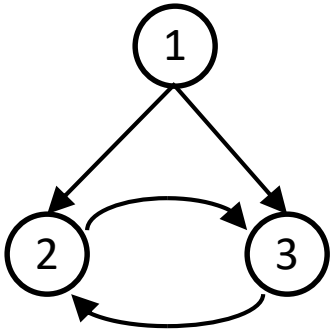


Optimization

- **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 quadratic complexity
  - 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 (union-find) keeps "header" relationship
    - Practical algorithm: O(m log n)
    - Complexity: O(mα(m,n)), α is inverse Ackermann function

Reducibility

T1: Remove n->n loops
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 Analysis

- **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
    - e.g., see next lecture on Global Scheduling
- **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

Today's Class: Region-Based Analysis

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

Friday's Class

- Global Scheduling
  - ALSU 10.4-10.5