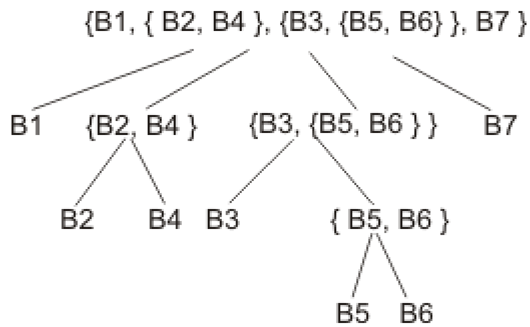
Module 14: Approaches to Control Flow Analysis

Lecture 28: Structural Analysis

The Lecture Contains:

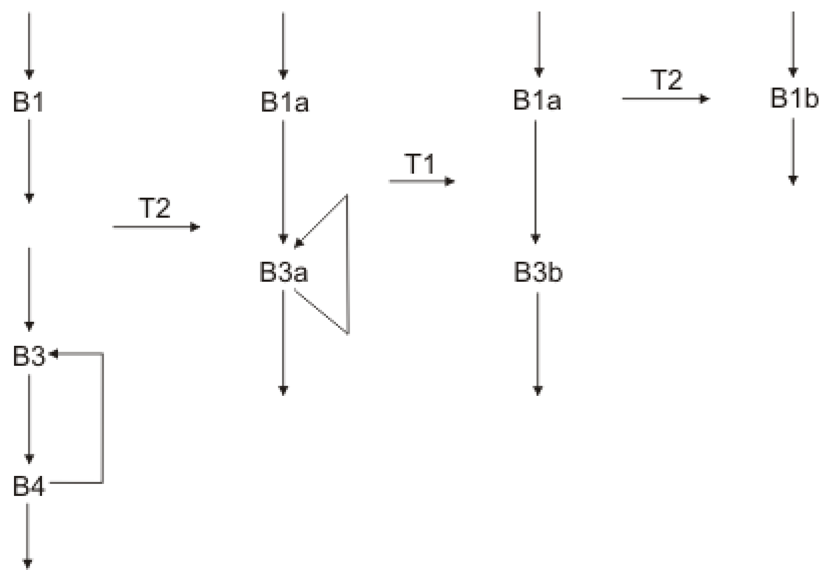
- T1- T2 Analysis
- Structural Analysis
- Dataflow Analysis
- Typical Equation
- Reaching Definitions
- Analysis of Structured Programs
- Reaching Definition Analysis
- Constant Folding
- Example

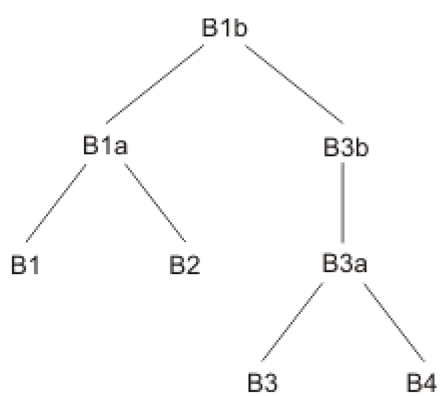
Previous Next



T1- T2 Analysis

- T1 transformation collapses one node self loop to a node
- T2 transformation collapses sequence of two nodes into one if the second node has only one predecessor





Previous Next

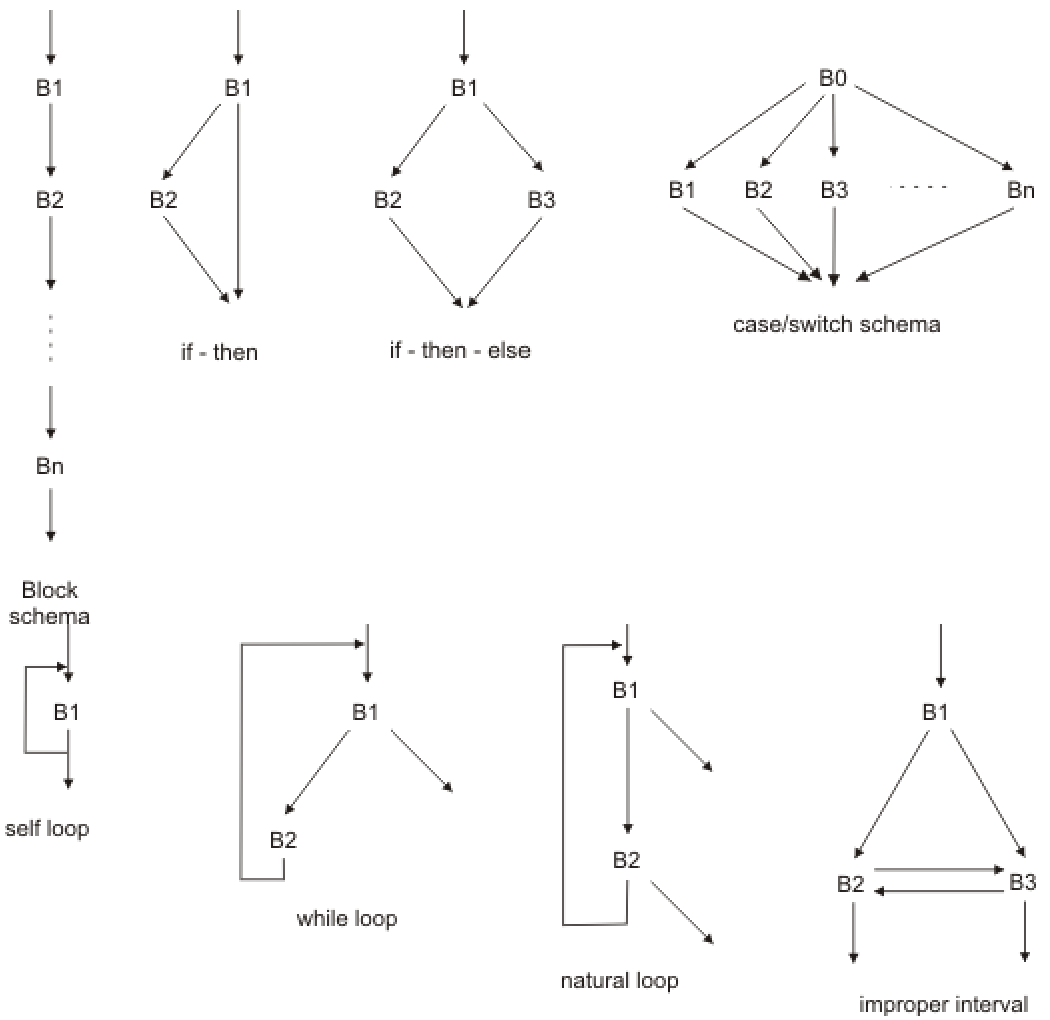
Module 14: Approaches to Control Flow Analysis

Lecture 28: Structural Analysis

Structural Analysis

- It is a more refined form of interval analysis
- It uses syntax directed method of dataflow analysis
- For each structure in the source it gives a formula
- It is more efficient than iterative method
- It has a construct for each type of region
- Control tree is larger than the one generated by interval analysis
- Each region is simple and small
- Every region has exectly one entry point

Some types of cyclic regions used in structural analysis

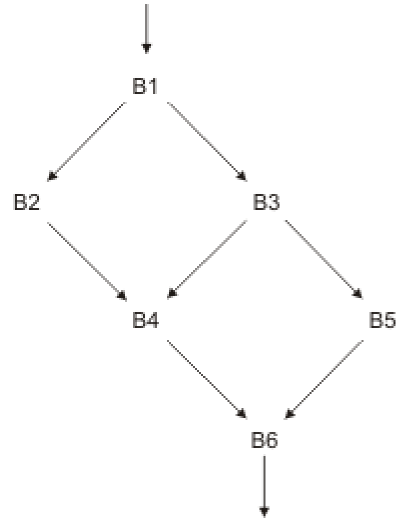


Previous Next

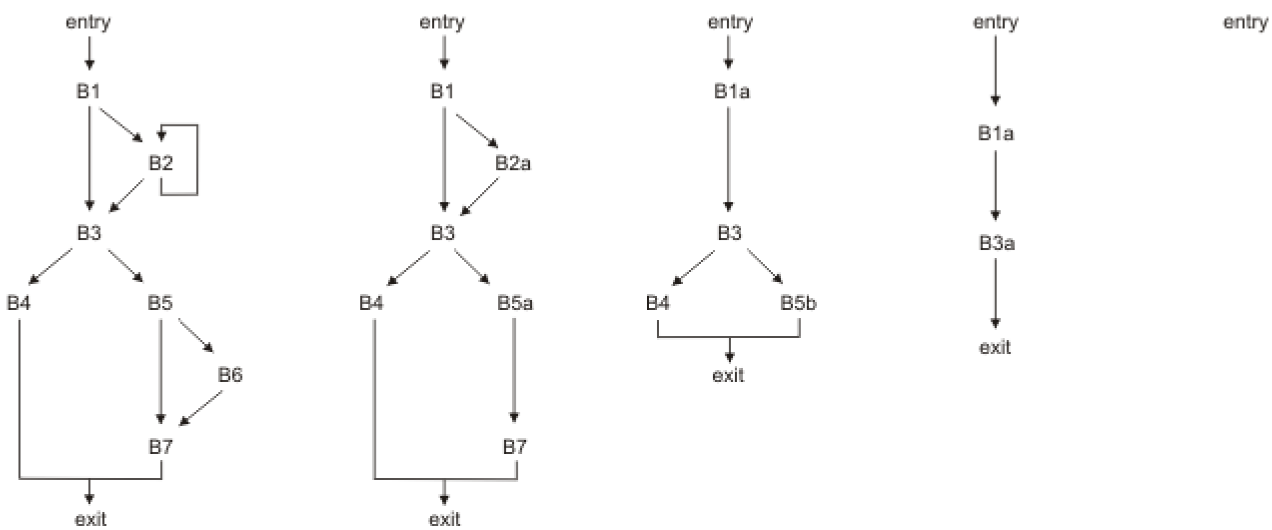
Module 14: Approaches to Control Flow Analysis

Lecture 28: Structural Analysis

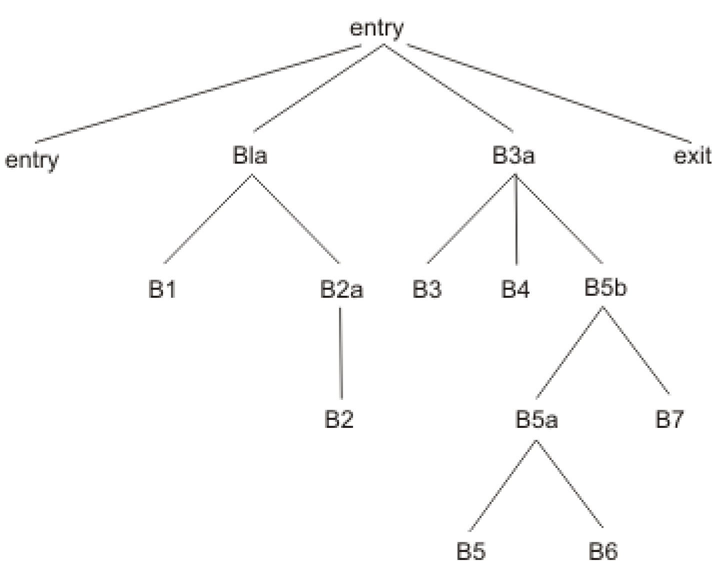
An acyclic region that does not fit any of the simple categories and so is identified as a proper interval



Structural Analysis of a flow graph

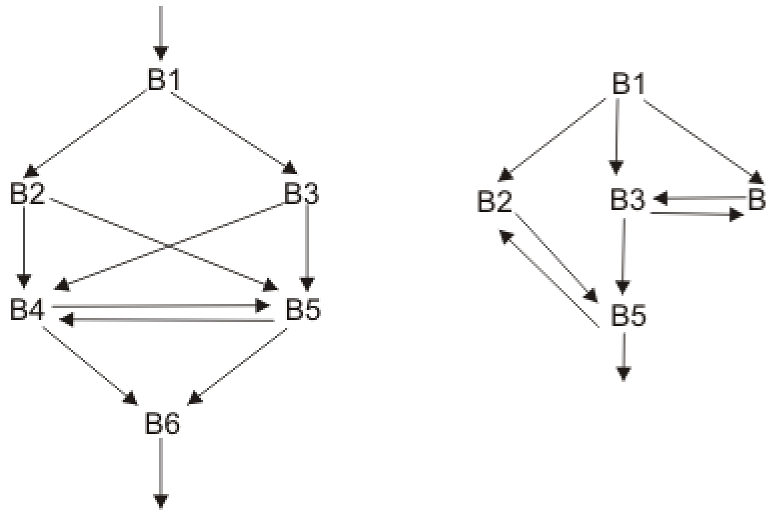


Control tree of the flow graph analyzed in the previous slide



Module 14: Approaches to Control Flow Analysis

Lecture 28: Structural Analysis



Improper Intervals

Dataflow Analysis

- Provide info about how a program segment manipulates data
- Analysis must be conservative and aggressive
- Collect information for optimization
  - Reaching definition
  - Available expression
  - Live variable
  - Busy expression

Reaching Definition : A definition d reaches a point p if there is a path from d to p and d is not killed on the path
Available Expression : An expression X+Y is available at point p if every path to p evaluates X+Y and after the last such evaluation no assignment to X or Y
Live Variable : For a variable X and point p whether value of X at p can be used along some path starting from p. If yes X is live at p else X is dead at p
Busy Expression : An expression B op C is busy at point p if along every path from p we come to computation B op C before any definition of B or C

Typical Equation

out(S) = gen(S) [in(S) - kill(S)
Gen : definitions generated
Kill : definitions killed
In : input definitions
Out : output definitions

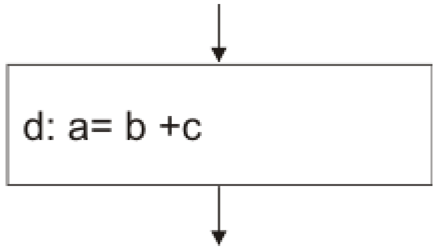
Reaching Definitions

Unambiguous Definitions :	Assignments
Ambiguous Definitions :	–procedure call with X as var parameter –procedure that can

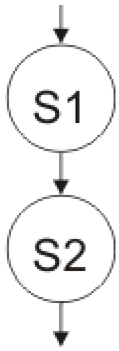
access X
–pointer *q = y

Previous Next

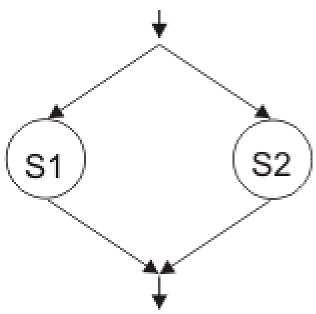
Analysis of Structured Programs



gen(S) = {d}
kill(S) = Da - {d}
out(S) = gen(S) S in(S) - kill(S)

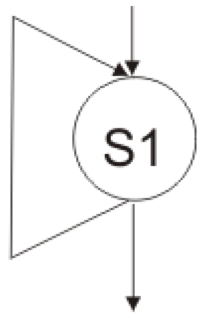


gen(S) = gen(S2) S gen(S1) - kill(S2)
kill(S) = kill(S2) S kill(S1) - gen(S2)
in(S1) = in(S)
in(S2) = out(S1)
out(S) = out(S2)



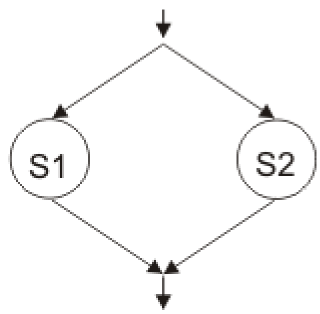
gen(S) = gen(S1) S gen(S2)
kill(S) = kill(S1) T kill(S2)
out(S) = out(S1) S out(S2)

Previous Next



gen(S) = gen(S1)
kill(S) = kill(S1)
out(S) = out(S1)
in(S1) = in(S) S gen(S1)

Assumptions : All paths in the flow graph are possible



Suppose E is true and it never goes to S2
gen(S) = gen(S1)
kill(S) = kill(S1)
out(S) = out(S1)

Therefore

true gen(S)   gen(S)
true kill(S)   kill(S)

True is what is computed during execution therefore, this is safe estimate

- Prevents optimization
- No wrong optimization

Reaching Definition Analysis

$$in(B) = \bigcup_{P \text{ is pred of } B} out(P)$$

$$out(B) \quad gen(B) \cup in(B)$$

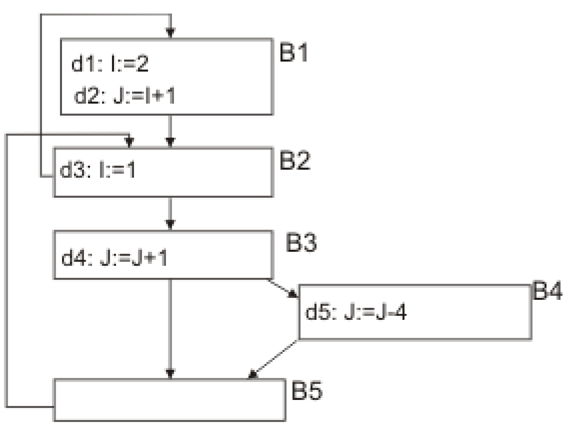
A definition d reaches end of a block iff either

- It is generated in the block
- It reaches block and not killed

Kill & gen known for each block. A program with N blocks has 2N equations with 2N unknowns and therefore, solution is possible.

- Use iterative forward bit vector approach

for each block B do
in(B) = Ø ;
out(B) = gen(B)
endfor;
change = true;
while change do
change = false;
for each block B do
newin = S out(P)
if newin 6= in(B) then {
change = true;
in(B) = newin;
out(B) = in(B) - kill(B) S gen(B);
}
endfor
endwhile



Block	gen	kill
B_1	d_1d_2	$d_3d_4d_5$
B_2	d_3	d_1
B_3	d_4	d_2d_5
B_4	d_5	d_2d_4
B_5	ø	ø

block	init		pass1		pass2		pass3	
	in	out	in	out	in	out	in	out
B_1		d_1d_2	d_3	d_1d_2	d_2d_3	d_1d_2	$d_2d_3d_4d_5$	d_1d_2
B_2		d_3	d_1d_2	d_2d_3	$d_1d_2d_3d_4d_5$	$d_2d_3d_4d_5$	$d_1d_2d_3d_4d_5$	$d_2d_3d_4d_5$
B_3		d_4	d_2d_3	d_3d_4	$d_2d_3d_4d_5$	d_3d_4	$d_2d_3d_4d_5$	d_3d_4
B_4		d_5	d_3d_4	d_3d_5	d_3d_5	d_3d_5	d_3d_4	d_3d_5
B_5			$d_3d_4d_5$	$d_3d_4d_5$	$d_3d_4d_5$	$d_3d_4d_5$	$d_3d_4d_5$	$d_3d_4d_5$

ud(I,d2) = d1
ud(J,d4) = d2d4d5
ud(J,d5) = d4
ud(I,B5) = d3

Constant Folding

While changes occur do
for all the stmts S of the program do
for each operand B of S do
if there is a unique definition of B
that reaches S and is a constant C
then replace B by C in S;
if all the operands of S are constant
then replace rhs by eval(rhs);
endfor
endfor
endwhile

Example