



Program Analysis
<https://www.cse.iitb.ac.in/~karkare/cs618/>

Data Flow Analysis

Amey Karkare
Dept of Computer Science and Engg
IIT Kanpur
Visiting IIT Bombay
karkare@cse.iitk.ac.in
karkare@cse.iitb.ac.in



Recap

- Optimizations
 - Machine Independent
 - Machine Dependent
- Analysis
 - Intraprocedural
 - Local
 - Global
 - Interprocedural

2

Agenda

- For the next few lectures
- *Intraprocedural* Data Flow analysis
 - Components
 - Classical examples

3

Assumptions

- Unless otherwise specified
- Intraprocedural: Restrict to a single procedure
- Input in 3-address code format

4

3 Address Code

- **Assignments**

$x = y \text{ op } z$

$x = \text{op } y$

$x = y$

- **Jump/Control statements**

goto L

if x relop y goto L

- **Statements can have label(s)**

L: ...

Arrays, Pointers and
Procedures to be added
later when needed

5

Data Flow Analysis

- Class of techniques to derive information about flow of data
 - along program execution paths
- Used to answer questions such as:
 - whether two identical expressions evaluate to same value
 - used in common subexpression elimination
 - whether the result of an assignment is used later
 - used by dead code elimination

6

Data Flow Abstraction

- **Basic Blocks (BB)**
 - sequence of 3-address code stmts
 - single entry at the first statement
 - single exit at the last statement
 - Typically we use “maximal” basic block (maximal sequence of such instructions)

7

Data Flow Abstraction

- **Leader:** First statement of a basic block
 - First instruction of program (procedure)
 - Target of a branch (goto)
 - Instruction immediately following a branch

8

Special Basic Blocks

- Two special BBs are added to simplify the analysis
 - empty (?) blocks!
- *Entry*: Assumed to be the *first* block to be executed for the procedure analyzed
- *Exit*: Assumed to be the *last* block to be executed

9

Data Flow Abstraction

- Control Flow Graph (CFG)
- A rooted directed graph $G = (N, E)$
- N = set of BBs
 - including *entry*, *exit*
- E = set of edges

10

CFG Edges

- Edge $B1 \rightarrow B2 \in E$ if control can transfer from $B1$ to $B2$
 - Fall through
 - Through jump (goto)
 - Edge from *entry* to (all?) real first BB(s)
 - Edge to *exit* from all last BBs
 - BBs containing *return*
 - Last *real* BB

11

Data Flow Abstraction

- Control Flow graph
 - Graph representation of paths that program may exercise during execution
 - Typically one graph per procedure
 - Graphs for separate procedures have to be combined/connected for interprocedural analysis
 - Later!
 - Single procedure, single flow graph for now.

12

Data Flow Abstraction

- Input state/Output state for Stmt
 - Program point before/after a stmt
 - Denoted IN[s] and OUT[s]
 - Within a basic block:
 - Program point after a stmt is same as the program point before the next stmt

13

Data Flow Abstraction

- Input state/Output state for BBs
 - Program point before/after a bb
 - Denoted IN[B] and OUT[B]
 - For B1 and B2:
 - if there is an edge from B1 to B2 in CFG, then the program point *after* the last stmt of B1 *may be* followed immediately by the program point *before* the first stmt of B2.

14

Data Flow Abstraction

- Execution Path
 - p_1, p_2, \dots, p_n
 - $p_i \rightarrow p_{i+1}$ are adjacent program points in the CFG
- Infinite number of possible execution paths.
- No finite upper bound on the length.
- Need to Summarize the information at a program point with a finite set of facts.

15

Data Flow Schema

- Data flow values associated with each program point
 - Summarize all possible states at that point
- Domain: set of all possible data flow values
- Different domains for different analysis/optimization

16

Data Flow Problem

- Constraints on data flow values
 - Transfer constraints
 - Control flow constraints
- AIM: To find a solution to the constraints
 - Multiple solutions possible
 - Trivial solutions,..., Exact solutions
- We typically compute approximate solution, close to the exact solution
 - Why not exact solution?

17

Data Flow Constraints

- Transfer functions
 - relationship between the data flow values before and after a stmt
- forward functions

$$\text{OUT}[s] = f_s(\text{IN}[s])$$
- backward functions

$$\text{IN}[s] = f_s(\text{OUT}[s])$$

18

Data Flow Constraints

- Control flow constraints
 - relationship between the data flow values of two points that are related by program execution semantics
- For a basic block having n statements:

$$\text{IN}[s_{i+1}] = \text{OUT}[s_i], i = 1, 2, \dots, n-1$$

IN[s₁], OUT[s_n] to come later

19

Data Flow Constraints: Basic Block

- Forward
 - For B consisting of s_1, s_2, \dots, s_n

$$f_B = f_{s_n} \circ \dots \circ f_{s_2} \circ f_{s_1}$$

$$\text{OUT}[B] = f_B(\text{IN}[B])$$
 - Control flow constraints

$$\text{IN}[B] = \bigcup_{P \text{ a predecessor of } B} \text{OUT}[P]$$
- Backward

$$\text{IN}[B] = f_B(\text{OUT}[B])$$

$$\text{OUT}[B] = \bigcup_{S \text{ a successor of } B} \text{IN}[S]$$

20

Data Flow Equations

- Typical Equation
 - $out[s] = in[s] - kill[s] \cup gen[s]$
 - $gen(s)$: information generated
 - $kill(s)$: information killed
- For example:
 - $a = b * c$ // generates expression $b*c$
 - $c = 5$ // kills expression $b*c$
 - $d = b * c$ // is $b*c$ redundant here?

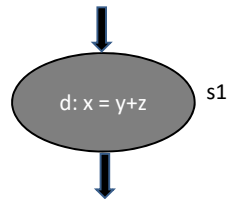
21

Analysis of Structured Programs

- Reaching Definitions Analysis
 - definition of a variable x :
 - $x = \dots something \dots$
 - Could be more complex (e.g. through pointers, references, implicit)
 - definition d reaches a point p if
 - there is a path from the point immediately following d to p
 - d is not “killed” along that path
 - “Kill” means redefinition of the left hand side (x in the above case)

22

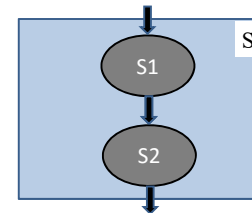
Analysis of Structured Programs



- $out(s1) = in(s1) - kill(s1) \cup gen(s1)$
- $gen(s1) = \{d\}$
- $kill(s1) = D_x - \{d\}$ // D_x : set of all defs of x
- $kill(s1) = D_x$ will also work here!
- But may not work in general!

23

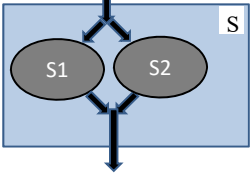
Analysis of Structured Programs



- $gen(s) = gen(s2) \cup (gen(s1) - kill(s2))$
- $kill(s) = kill(s2) \cup (kill(s1) - gen(s2))$
- $in(s1) = in(s)$
- $in(s2) = out(s1)$
- $out(s) = out(s2)$

24

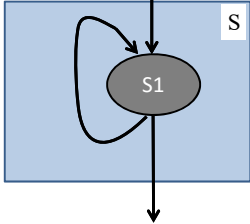
Analysis of Structured Programs



$gen(s) = gen(s1) \cup gen(s2)$
 $kill(s) = kill(s1) \cap kill(s2)$
 $in(s1) = in(s2) = in(s)$
 $out(s) = out(s1) \cup out(s2)$

25

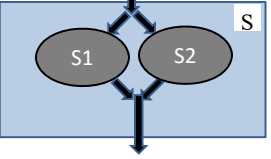
Analysis of Structured Programs



$gen(s) = gen(s1)$
 $kill(s) = kill(s1)$
 $in(s1) = in(s) \cup gen(s1)$
 $out(s) = out(s1)$

26

Conservative Analysis

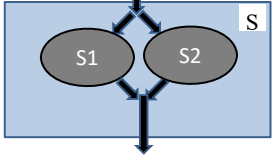


- Assumption: All paths are feasible.
 - Consider: if (true) s1; else s2
 - s2 is never executed

$gen(s) = gen(s1) \subseteq gen(s1) \cup gen(s2)$
 $kill(s) = kill(s1) \supseteq kill(s1) \cap kill(s2)$

27

Conservative Analysis



- Thus: $true\ gen(s) \subseteq analysis\ gen(s)$
 $true\ kill(s) \supseteq analysis\ kill(s)$
- True is what is computed at run time
- This is **SAFE** estimate
 - prevents optimization
 - but no wrong optimization

28