

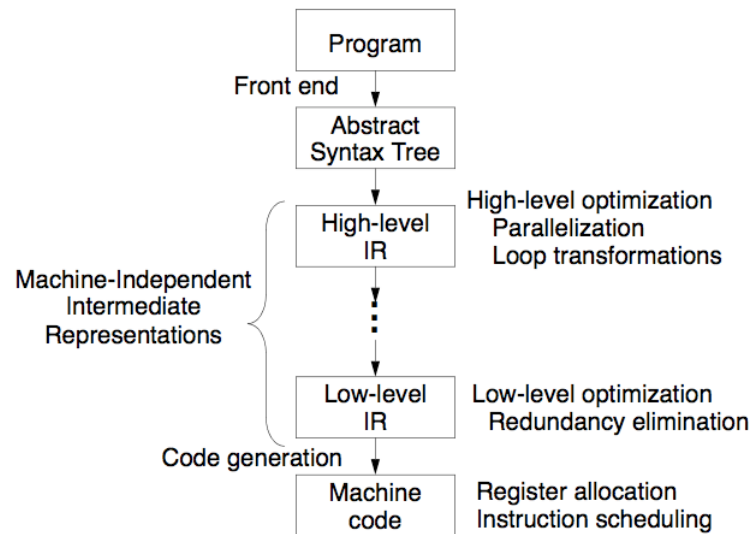
# Lecture 2

## Introduction to Data Flow Analysis

- I. Introduction
- II. Example: Reaching definition analysis
- III. Example: Liveness analysis
- IV. A General Framework  
(Theory in next lecture)

Reading: Chapter 9.2

### I. Compiler Organization

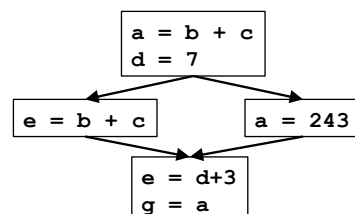


## Flow Graph

- **Basic block** = a maximal sequence of consecutive instructions s.t.
  - flow of control only enters at the beginning
  - flow of control can only leave at the end  
(no halting or branching except perhaps at end of block)
- **Flow Graphs**
  - Nodes: basic blocks
  - Edges
    - $B_i \rightarrow B_j$ , iff  $B_j$  can follow  $B_i$  immediately in execution

## What is Data Flow Analysis?

- **Data flow analysis:**
  - Flow-sensitive: sensitive to the control flow in a function
  - intraprocedural analysis
- **Examples of optimizations:**
  - Constant propagation
  - Common subexpression elimination
  - Dead code elimination

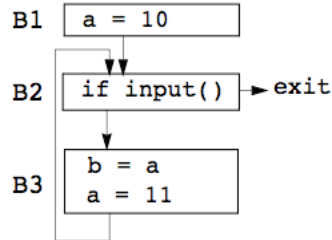


Value of  $x$ ?

Which "definition" defines  $x$ ?

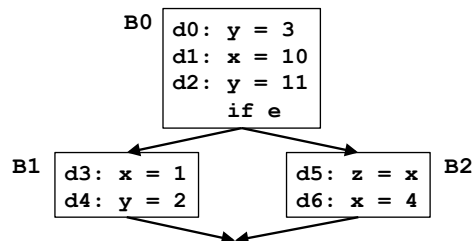
Is the definition still meaningful (live)?

## Static Program vs. Dynamic Execution



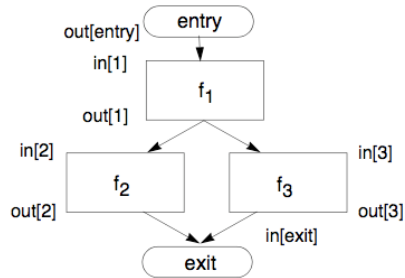
- **Statically:** Finite program
- **Dynamically:** Can have infinitely many possible execution paths
- **Data flow analysis abstraction:**
  - For each point in the program:  
combines information of all the instances of the same program point.
- **Example of a data flow question:**
  - Which definition defines the value used in statement "b = a"?

## Reaching Definitions



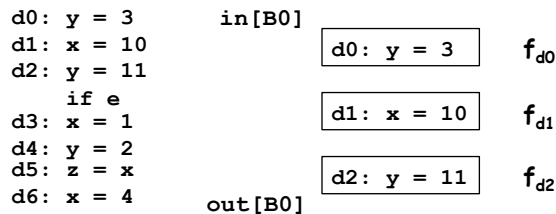
- Every assignment is a definition
- A **definition  $d$  reaches** a point  $p$  if **there exists** path from the point immediately following  $d$  to  $p$  such that  $d$  is not killed (overwritten) along that path.
- Problem statement
  - For each point in the program, determine  
if each definition in the program reaches the point
  - A bit vector per program point, vector-length = #defs

## Data Flow Analysis Schema



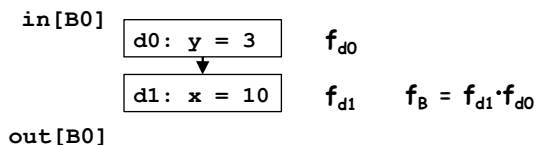
- Build a flow graph (nodes = basic blocks, edges = control flow)
- Set up a set of equations between  $in[b]$  and  $out[b]$  for all basic blocks  $b$ 
  - Effect of code in basic block:
    - Transfer function  $f_b$  relates  $in[b]$  and  $out[b]$ , for same  $b$
  - Effect of flow of control:
    - relates  $out[b_1]$ ,  $in[b_2]$  if  $b_1$  and  $b_2$  are adjacent
- Find a solution to the equations

## Effects of a Statement



- $f_s$ : A transfer function of a statement
  - abstracts the execution with respect to the problem of interest
- For a statement  $s$  ( $d: x = y + z$ )
  - $out[s] = f_s(in[s]) = Gen[s] \cup (in[s] - Kill[s])$ 
    - **Gen[s]**: definitions generated:  $Gen[s] = \{d\}$
    - **Propagated** definitions:  $in[s] - Kill[s]$ ,  
 where **Kill[s]**=set of all other defs to  $x$  in the rest of program

## Effects of a Basic Block

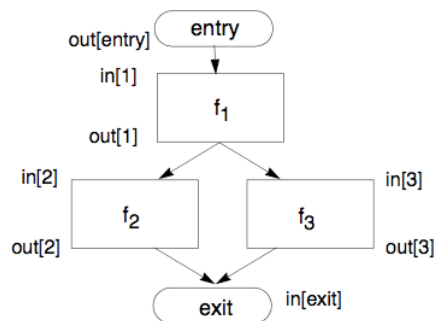


- Transfer function of a statement  $s$ :
  - $out[s] = f_s(in[s]) = Gen[s] \cup (in[s] - Kill[s])$
- Transfer function of a basic block  $B$ :
  - Composition of transfer functions of statements in  $B$
- $out[B] = f_B(in[B])$ 
  - $= f_{d_1} f_{d_0}(in[B])$
  - $= Gen[d_1] \cup (Gen[d_0] \cup (in[B] - Kill[d_0])) - Kill[d_1]$
  - $= (Gen[d_1] \cup (Gen[d_0] - Kill[d_1])) \cup in[B] - (Kill[d_0] \cup Kill[d_1])$
  - $= Gen[B] \cup (in[B] - Kill[B])$

$Gen[B]$ : locally exposed definitions (available at end of bb)

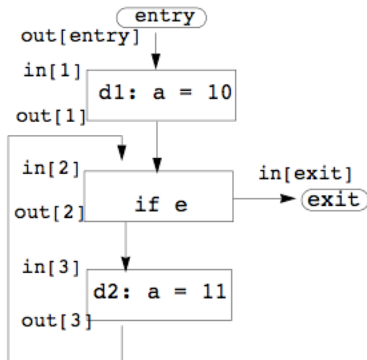
$Kill[B]$ : set of definitions killed by  $B$

## Effects of the Edges (acyclic)



- Join node: a node with multiple predecessors
- **meet** operator ( $\wedge$ ):  $\cup$ 
  - $in[b] = out[p_1] \cup out[p_2] \cup \dots \cup out[p_n]$ , where
  - $p_1, \dots, p_n$  are all predecessors of  $b$

## Cyclic Graphs



- Equations still hold
  - $out[b] = f_b(in[b])$
  - $in[b] = out[p_1] \cup out[p_2] \cup \dots \cup out[p_n]$ ,  $p_1, \dots, p_n$  pred.
- Find: fixed point solution

## Reaching Definitions: Iterative Algorithm

input: control flow graph  $CFG = (N, E, Entry, Exit)$

```
// Boundary condition
out[Entry] =  $\emptyset$ 
```

```
// Initialization for iterative algorithm
For each basic block B other than Entry
  out[B] =  $\emptyset$ 
```

```
// iterate
While (Changes to any out[] occur) {
  For each basic block B other than Entry {
    in[B] =  $\cup$  (out[p]), for all predecessors p of B
    out[B] =  $f_B(in[B])$  //  $out[B] = gen[B] \cup (in[B] - kill[B])$ 
  }
}
```

## Summary of Reaching Definitions

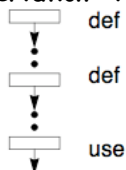
	<b>Reaching Definitions</b>
Domain	Sets of definitions
Transfer function $f_b(x)$	forward: $out[b] = f_b(in[b])$ $f_b(x) = Gen_b \cup (x - Kill_b)$ $Gen_b$ : definitions in $b$ $Kill_b$ : killed defs
Meet Operation	$in[b] = \cup out[predecessors]$
Boundary Condition	$out[entry] = \emptyset$
Initial interior points	$out[b] = \emptyset$

## III. Live Variable Analysis

- **Definition**
  - A variable  $v$  is **live** at point  $p$  if
    - the value of  $v$  is used along some path in the flow graph starting at  $p$ .
  - Otherwise, the variable is **dead**.
- **Problem statement**
  - For each basic block
    - determine if each variable is live in each basic block
  - Size of bit vector: one bit for each variable

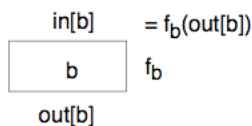
## Effects of a Basic Block (Transfer Function)

- **Observation:** Trace uses back to the definitions



example:

$$\begin{aligned} m &= n+q \\ p &= m \end{aligned}$$



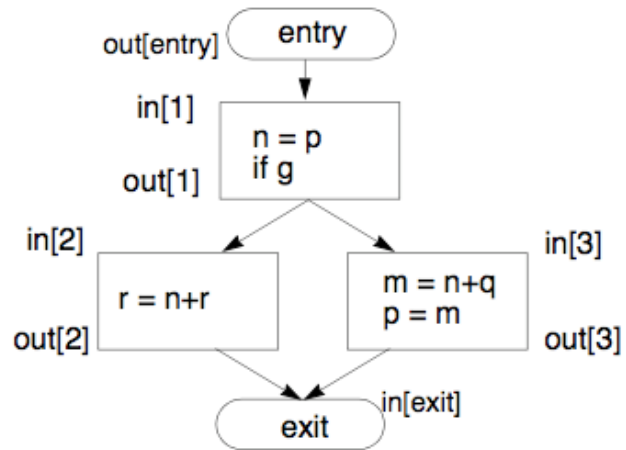
- **Direction: backward:**  $in[b] = f_b(out[b])$
- **Transfer function** for statement  $s: x = y + z$ 
  - generate live variables:  $Use[s] = \{y, z\}$
  - propagate live variables:  $out[s] - Def[s], Def[s] = x$
  - $in[s] = Use[s] \cup (out(s)-Def[s])$
- **Transfer function** for basic block  $b$ :
  - $in[b] = Use[b] \cup (out(b)-Def[b])$
  - $Use[b]$ , set of locally exposed uses in  $b$ , uses not covered by definitions in  $b$
  - $Def[b]$ = set of variables defined in  $b$ .

## Across Basic Blocks

- **Meet operator ( $\wedge$ ):**
  - $out[b] = in[s_1] \cup in[s_2] \cup \dots \cup in[s_n], s_1, \dots, s_n$  are successors of  $b$
- **Boundary condition:**



## Example



## Liveness: Iterative Algorithm

input: control flow graph  $CFG = (N, E, Entry, Exit)$

*// Boundary condition*

$in[Exit] = \emptyset$

*// Initialization for iterative algorithm*

For each basic block B other than Exit

$in[B] = \emptyset$

*// iterate*

While (Changes to any  $in[]$  occur) {

For each basic block B other than Exit {

$out[B] = \cup (in[s])$ , for all successors s of B

$in[B] = f_B(out[B])$  //  $in[B] = Use[B] \cup (out[B] - Def[B])$

}

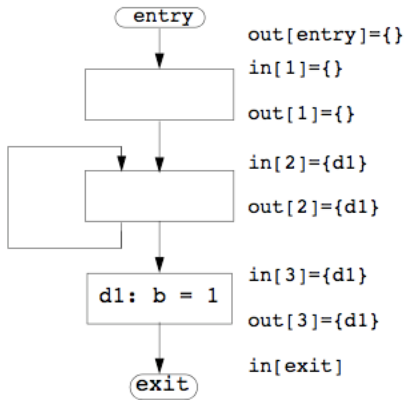
## IV. Framework

	Reaching Definitions	Live Variables
Domain	Sets of definitions	Sets of variables
Direction	forward: $out[b] = f_b(in[b])$ $in[b] = \wedge out[pred(b)]$	backward: $in[b] = f_b(out[b])$ $out[b] = \wedge in[succ(b)]$
Transfer function	$f_b(x) = Gen_b \cup (x - Kill_b)$	$f_b(x) = Use_b \cup (x - Def_b)$
Meet Operation ( $\wedge$ )	$\cup$	$\cup$
Boundary Condition	$out[entry] = \emptyset$	$in[exit] = \emptyset$
Initial interior points	$out[b] = \emptyset$	$in[b] = \emptyset$

### Thought Problem 1. "Must-Reach" Definitions

- **A definition D ( $a = b+c$ ) must reach point P iff**
  - D appears at least once along on all paths leading to P
  - a is not redefined along any path after last appearance of D and before P
- **How do we formulate the data flow algorithm for this problem?**

## Problem 2: A legal solution to (May) Reaching Def?



- Will the worklist algorithm generate this answer?

## Problem 3. What are the algorithm properties?

- **Correctness**
- **Precision:** how good is the answer?
- **Convergence:** will the analysis terminate?
- **Speed:** how long does it take?