CSCI 7135
Introduction

Amer Diwan

Goals of the class

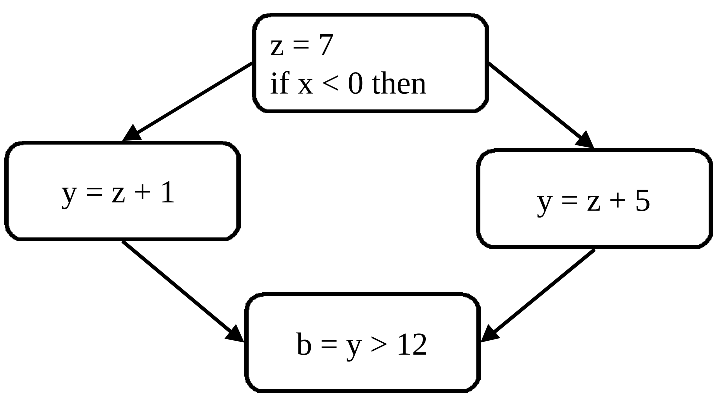
- A deep and up-to-date understanding of
  - compile-time program analyses
  - run-time program analyses

  and their applications
- Method
  - Read and critique recent and influential papers
  - Implement some ideas

Compile-time program analyses

- Discovers properties of programs by looking at its source
  - Local (a few lines of straight line code)
  - Global or intraprocedural (full procedure)
  - Interprocedural (several procedures)
    - Special case: Whole program analysis

Example of compile-time program analysis

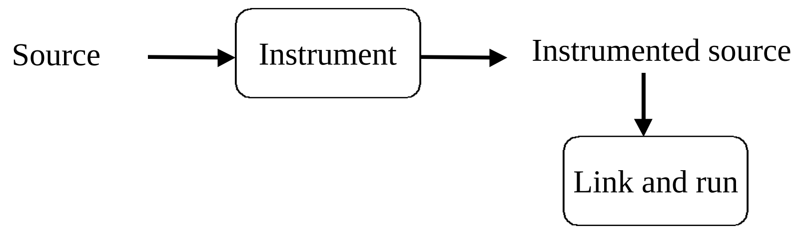


What are possible values of y?

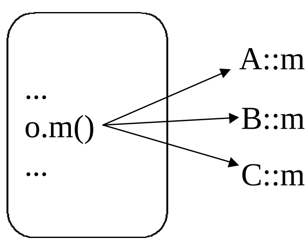
Does this need local, global(intraprocedural), or interprocedural analysis?

Run-time program analyses

- Discovers properties of programs by examining its runs



Example of run-time program analysis



Which is the most common target of o.m()?

Hybrid analyses

- Combines run-time analysis and compile-time analysis
  - May use a compile-time analysis to reduce overhead of run-time analysis
  - May use run-time analysis to guide compile-time analysis to hot-spots

First topic: Data-flow analysis

- A commonly used technique for compile-time analysis
- Readings:
  - Aho, Sethi, and Ullman Sections 10.1 to 10.6; or
  - Muchnick Sections 8.1 to 8.4; or
  - Relevant sections from your favorite compiler text

Outline

- Preliminaries
  - Control flow graphs and basic blocks
- Fundamentals of data flow analysis
- Examples

Basic blocks

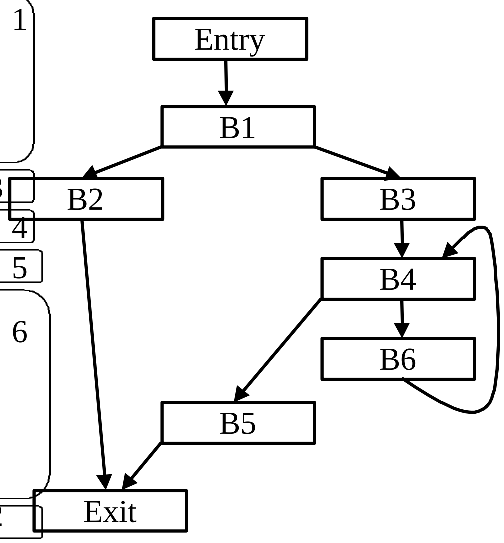
- A maximal sequence of instructions s.t.:
  - Only the first statement can be reached from outside the block
  - All the statements are executed consecutively if the first one is

Control flow graphs

- Nodes: basic blocks
- Edges: B_i -> B_j iff B_j can follow B_j immediately in some execution
- It is convenient to insert special entry and exit nodes

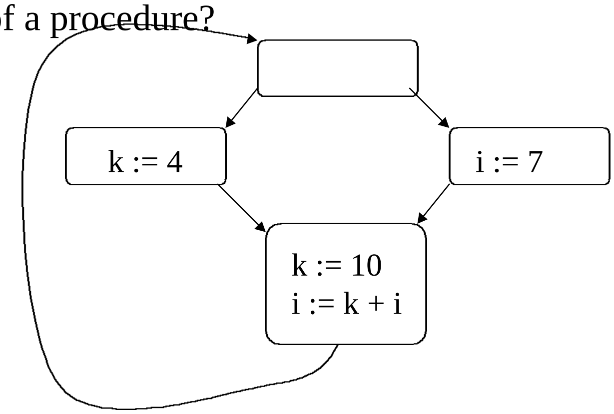
CFG and Basic Block Example

```
    receive m              1
    f0 := 0
    f1 := 1
    if m <= 1 goto L3
    i := 2                 3
L1  if i <= m goto L2      4
    return f2              5
L2  f2 := f0 + f1          6
    f0 := f1
    f1 := f2
    i := i + 1
    goto L1
L3  return m               2
```



Data-flow analysis example: reaching definitions

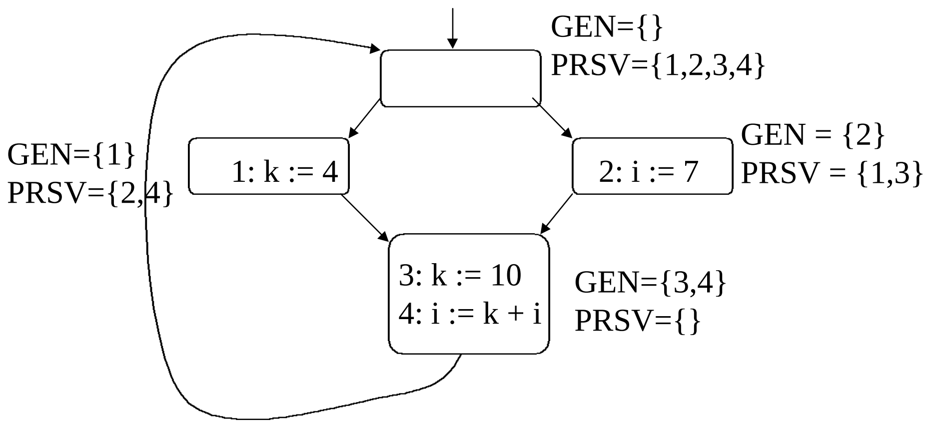
- What definitions of each variable reach each point of a procedure?



PRSV and GEN sets

- A basic block preserves a property if it does not alter it (i.e., kill it)
- A basic block generates a property if it creates and and doesn't subsequently kill it
- In our example, the property of interest is whether or not a definition reaches a point in the program

Example continued



What definitions reach the end of each block?

Definitions reaching beginning of block that are preserved in the block

+

Definitions generated and not subsequently killed by the block

RCHout(i) = GEN(i) ∪ (RCHin(i) ∩ PRSV(i))

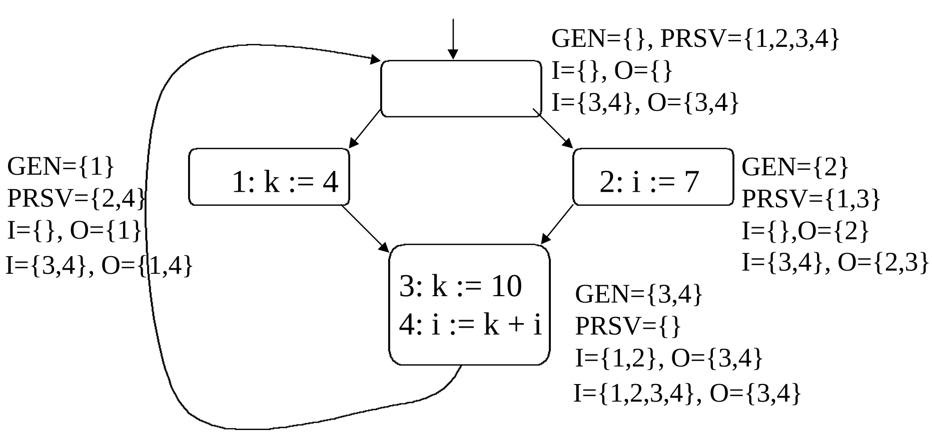
What definitions reach the beginning of each block?

Definitions reaching end of at least one of its predecessors

RCHin(i) = ∪ RCHout(j), s.t. j is a predecessor of i

Example: RCHin and RCHout sets

Initialize in and out sets to empty; Assume “top-down” visit order



Observations from example

- IN and OUT are recursive
  - May need multiple iterations to solve equations
- When is one iteration surely enough?
- For many data-flow problems, the IN, OUT, PRSV, and GEN sets can be represented as bit vectors

Union = Bit OR

Intersection = Bit AND

Steps in data flow analysis (simplified)

Analysis Dependent

I Formulate the problem to be solved

Analysis Independent

II Solve the equations induced by I

III Propagate the data-flow values to all points in the program from entries to blocks

I Formulating the problem

(a) Lattice

- the abstract quantities over which the analysis will operate (lattice)
- e.g., sets of definitions for a variable

(b) Flow functions

- how each control-flow and computational construct affects the abstract quantities (flow functions)
- e.g., build the OUT equations for each statement

I(a) Lattice

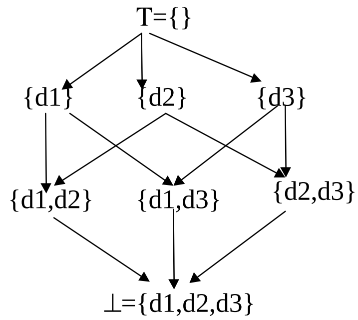
A lattice L consists of a set of values and two operations meet($\wedge$) and join($\vee$)

Properties ($x, y, z, w \in L$):

- $\exists$ unique z and w s.t. $x \wedge y = z$ and $x \vee y = w$
- $x \wedge y = y \wedge x$ and $x \vee y = y \vee x$ (commutativity)
- $(x \wedge y) \wedge z = x \wedge (y \wedge z)$ and $(x \vee y) \vee z = x \vee (y \vee z)$
- there are unique elements $\perp$, $T \in L$ s.t. $x \wedge \perp = \perp$ and $x \vee T = T$

Example lattice: Reaching definitions

d1, d2, and d3 are definitions of some variable in the program



Meet of two elements: follow lines downwards from them until they meet = set union

Another useful view

- Define $x \subseteq y$ if and only if $x \wedge y = x$
- $\subseteq$ is a partial order
  - Reflexive: $x \subseteq x$
  - Antisymmetric: if $x \subseteq y$ and $y \subseteq x$ then $x = y$
  - Transitive: if $x \subseteq y$ and $y \subseteq z$ then $x \subseteq z$
- The height of the lattice is the longest ascending chain in it ($\perp$, x1, ..., xn, T)
- What is the lattice height for reaching definitions?

I(b) Flow functions

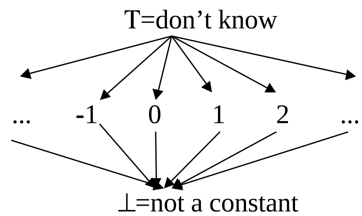
- f: L $\rightarrow$ L
- Models the effect of a programming language construct
- It is monotone if $\forall$ x, y $\in$ L, x $\subseteq$ y $\Rightarrow$ f(x) $\subseteq$ f(y)

Intuition for data-flow analysis

- Starts by assuming most optimistic values (T) and applying flow functions until it reaches a fixed point
- At each stage the abstract value of some "variables" descend the lattice
- If the effective lattice height w.r.t. the flow functions is finite, then the analysis is guaranteed to terminate

Example lattice: Constant propagation

- At every basic block boundary, for each variable v
  - determine if v is a constant
  - if so, what is its value



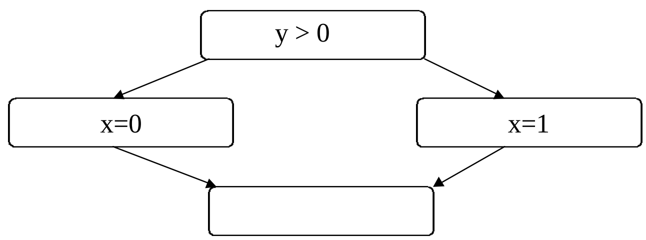
Flow function for constant propagation

Let an assignment be of the form $x_3 = x_2 \oplus x_1$
OUT[b,x] = IN[b,x] if x ≠ x3, otherwise

IN[b,x1]	IN[b,x2]	OUT[b,x3]
	top	top
top	c2	top
	bottom	bottom
	top	top
c1	c2	c1+c2
	bottom	bottom
	top	bottom
bottom	c2	bottom
	bottom	bottom

II Solving the data flow equations: Ideal solution

- For each node n : $\wedge f_p$(start-val), for all possibly executed paths p reaching n



Determining all possibly executed paths is undecidable

Solving the data flow equations: Meet over all paths

- Err in the conservative direction
- Meet over all paths (MOP)
  - Assume a path exists as long as there is a sequence of edges in the code
  - MOP(n) = $\wedge f_p$(start-val), for all paths p reaching n
- More conservative than ideal
  - MOP = IDEAL $\cap$ Result(unexecuted-paths)
  - MOP $\subseteq$ IDEAL
- MOP is also undecidable in the general case

Solving the data flow equations: Maximal fixed point

- More conservative than MOP
- Focuses on edges rather than paths
- MFP $\subseteq$ MOP $\subseteq$ IDEAL
- MFP = MOP if all flow functions are distributive
  - $f(x \wedge y) = f(x) \wedge f(y)$
- Is the constant propagation flow function distributive?

Solving data-flow equations: Iterative style

```
∀ nodes n != Entry, OUT(n) := T
OUT(Entry) := init_value
change = TRUE

While Change {
  Change := FALSE
   ∀ nodes i in reverse postorder {
     in[i] = ∧ out[p], p is a predecessor of i
     oldout := out[i]
     out[i] := f_i(in[i])
     if oldout != out[i] then change := TRUE
   }
}
```

Wrapping up

- Data-flow analysis is a common technique for static program analysis. Other approaches include
  - constraint based analyses, and
  - abstract interpretation