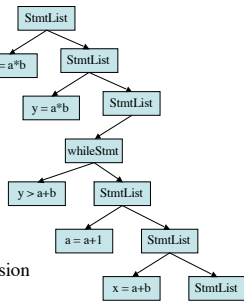


Data flow analysis

Abstract Syntax Trees

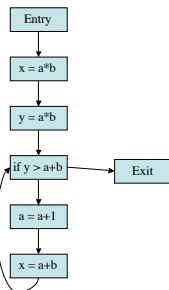
```
x = a*b;  
y = a*b;  
while (y > a+b) {  
  a = a+1;  
  x = a+b;  
}
```



AST stopped
at statement/expression
level for brevity

Control Flow Graph

```
x = a*b;  
y = a*b;  
while (y > a+b) {  
  a = a+1;  
  x = a+b;  
}
```



Choosing a representation

- Control flow graph is more general
- AST allows for more efficient algorithms
 - but new programming constructs require changing the algorithm
 - e.g., continue, break, switch, try-catch-finally, goto
 - program transformations may not leave the program in AST form
 - bytecode/machine code isn't in AST form
 - although you may be able to recover it

Data flow analysis

- A framework for proving facts about a program
 - reasoning about lots of little facts
 - little or no interaction between facts
 - based on all paths through program
 - including infeasible paths
 - e.g., which assignments to x can be seen at this read of x ?

Reaching definitions

- Each assignment to a variable is a definition
- $\text{defs}(v)$ represents the set of all definitions of v
- Assume all variables are scalars
 - no pointers or arrays

Gen and Kill

- $\text{Gen}(S)$ = facts that are true after S , regardless of the facts true before S
- $\text{Kill}(S)$ = facts that aren't true after S just because they were true before S
 - but might be true after S
- $\text{Out}(S) = \text{Gen}(S) \cup (\text{In}(S) - \text{Kill}(S))$

Initial conditions

- $\text{Out}(\text{Entry})$ needs to be separately defined
- What is appropriate for reaching definitions?

For reaching definitions

- $\text{Gen}(d: v = \text{exp}) = \{ d \}$
- $\text{Kill}(d: v = \text{exp}) = \text{defs}(v)$

Computing In(S)

- If S has one predecessor P, $In(S) = Out(P)$
- Otherwise,
 - $In(S) = \text{meet}_{P \in \text{Pred}(S)} Out(P)$
- The meet function defines how to combine alternatives
- For reaching definitions, meet = union

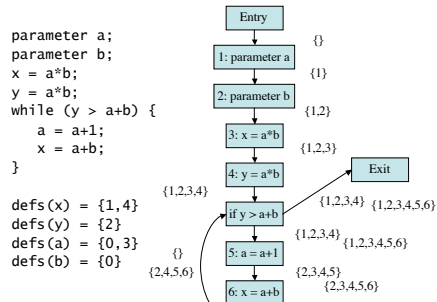
iterative solution

- For control flow graphs with cycles, can't directly solve the equations
 - compute final answer for values in terms of other final values already known
- Use iterative solution
 - Can compute dataflow values in *any* order
 - some orders are more efficient than others
 - computation will converge to right answer

Initial value

- For iterative solution
 - might need $Out(S)$ before we get a chance to compute $In(S)$
- Need an initial value for $Out(S)$ of all statements other than Entry

Control Flow Graph



More control flow programs

- Definitely uninitialized variables
- Possibly uninitialized variables
 - compare with definitely initialized variables
- What is Gen and Kill?
- What is Out(Entry)?
- What is the meet function?
- What is the initial value?

Available expressions

- An expression e is available at point p if on all paths to p , e must have been computed and since that computation, none of the variables in e have been modified
 - i.e., computation of e here would be redundant
- $Gen(x = a+b) = \{a+b\}$ - $Kill(x = a+b)$
- $Kill(x = a+b) = \text{any expression using } x$

Backwards problems

- Not all problems are computing from Entry towards exit
- Backwards problems start at Exit, and are computed backwards
- $\text{In}(S) = \text{Gen}(S) \cup (\text{Out}(S) - \text{Kill}(S))$
- $\text{Out}(S) = \text{meet}_{F \in \text{Succ}(S)} \text{In}(F)$

Backwards problems

- Live variables
 - which variables might be read before they are overwritten or discarded
- Very busy expressions
 - expressions that are guaranteed to be evaluated before any variable used in computing the expression is redefined

Constant Propagation

- Known constant values for variables

Questions

- Does it terminate?
- Does it compute a valid answer?

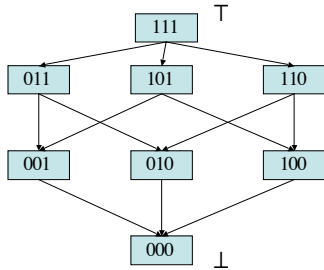
Definitions

- Meet function: \sqcap
- Meet function is commutative and associative
- $x \sqcap x = x$
- Unique bottom \perp and top \top element
 - $x \sqcap \perp = \perp$
 - $x \sqcap \top = x$

Ordering

- $x \sqsubseteq y$ if and only if $x \sqcap y = x$
- A function f is monotone if for all x and y ,
 - $x \sqsubseteq y$ implies $f(x) \sqsubseteq f(y)$

Lattice example



meet is bit-vector logical and

Relating to data flow analysis

- Top is value to initialize non-entry nodes to
 - the identity element for the meet function
- If node function is monotone
 - each re-evaluation of a node moves down the lattice, if it moves at all
- If height of lattice is finite, must terminate

Is it accurate?

- We want the meet over all paths solution
- $MOP(B) = \text{meet}_{\rho \text{ in Path}(\text{Entry}, B)} f_{\rho}(\text{Init})$
 - note that Paths can be infinite if there are loops
- As good as we can do given the framework
- Iterative analysis computes Maximum Fixed Point solution
 - largest solution, ordered by \sqsubseteq , that is a fixed point of the iterative computation
 - bottom is also a fixed point, but often not maximal

Is MOP correctly conservative?

- $MOP(B) = \sqcap_{\rho \text{ in Path}(\text{Entry}, B)} f_{\rho}(\text{Init})$
- Let $\text{AlmostTruth}(B) = \sqcap_{\rho \text{ in FeasiblePath}(\text{Entry}, B)} f_{\rho}(\text{Init})$
- Let $\text{Bogus}(B) = \sqcap_{\rho \text{ in InfeasiblePath}(\text{Entry}, B)} f_{\rho}(\text{Init})$
- $MOP(B) = \text{AlmostTruth}(B) \sqcap \text{Bogus}(B)$
 - $MOP(B) \sqsubseteq \text{AlmostTruth}(B)$

Distributive functions

- $x \sqsubseteq y$ implies $f(x) \sqsubseteq f(y)$
- Monotone implies
 - $f(x \sqcap y) \sqsubseteq f(x) \sqcap f(y)$
- f is distributive if and only if
 - $f(x \sqcap y) = f(x) \sqcap f(y)$
 - doing meet early doesn't cause any reduction in precision

Restatement of monotone

- A function f is monotone if for all x and y ,
 - $x \sqsubseteq y$ implies $f(x) \sqsubseteq f(y)$
- Prove $f(x \sqcap y) \sqsubseteq f(x) \sqcap f(y)$
- By definition, $x \sqsubseteq y$ if and only if $x \sqcap y = x$
- Prove $f(x \sqcap y) \sqcap f(x) \sqcap f(y) = f(x \sqcap y)$

We know $x \sqcap y \sqsubseteq x$

since f is monotone, $f(x \sqcap y) \sqsubseteq f(x)$

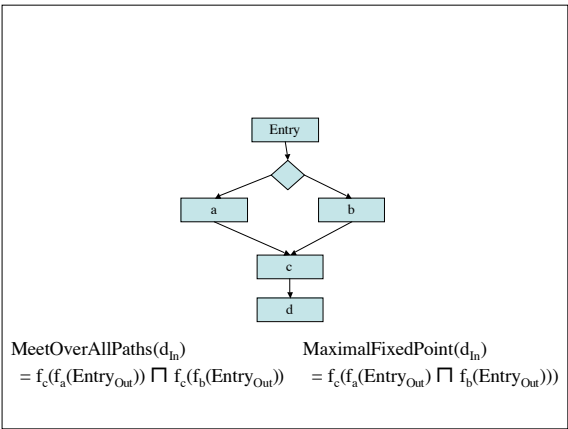
which means $f(x \sqcap y) \sqcap f(x) = f(x \sqcap y)$

and $f(x \sqcap y) \sqcap f(y) = f(x \sqcap y)$

$f(x \sqcap y) \sqcap f(x) \sqcap f(y)$

$= f(x \sqcap y) \sqcap f(y)$

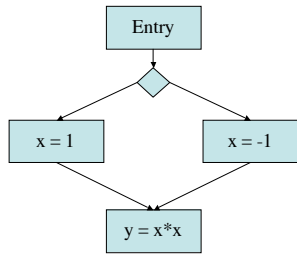
$= f(x \sqcap y)$



Distributive problems

- For a distributive problem
 - you can push transfer functions over meets without causing any reduction in accuracy
- Which problems are distributive?
 - reaching definitions, very busy expressions, live variables, available expressions
- Which are not?
 - most formulations of constant propagation

Constant propagation



All Gen/Kill problems are distributive

- If $Out_S = Gen_S \cup In_S - Kill_S$
- Problem is distributive
 - left at exercise for the reader
 - and/or exam question

Are all problems monotone?

- No, you have to be careful
- Consider constant propagation of truth values
 - What is the rule for `if x then y else z`

Basic Blocks

- When doing dataflow analysis for real
 - don't iterate through basic block and store in/out values for each statement
 - instead, store one in/out value for entire basic block
 - compose all of the transfer functions

Order matters

- For forward problems, visit nodes in reverse postorder
 - head visited before tail except for back edges
 - expected # of iterations = nesting depth
- For backwards problems
 - compute reverse postorder on reversed graph

Context Insensitive Analysis

- Analysis we've done so far depends on context and statement order
 - e.g., S1; S2 ≠ S2; S1
- Difficult to make context sensitive analysis scale to 100,000's of lines of code
 - let alone millions
- Context insensitive analysis simply combines information from statements
 - e.g., which variables are modified
