Introduction to **Program Analysis**

Reading: NNH 1.1-1.3, 1.7-1.8

17-654/17-765 Analysis of Software Artifacts Jonathan Aldrich

Applications of Program Analysis

- Optimization
 - Avoid redundant/unnecessary computation
 - Compute in a more efficient way
- Verifying correctness
 - Assurance of software
 - Finding bugs
- Determining properties
 - Performance
 - Security and reliability
 - Design and architecture

Analysis as an Approximation

• Example: finding divide-by-zero errors

read(x); if (x > 0)

then y := 1

else y := 0; S; // S is some other statement z := 2 / y;// could this be an error?

· What could y hold at the last statement? In general, anything (since S could assign to y)
 If S doesn't affect y, one would think the answer is the set {0,1}

Analysis as an Approximation

- If S doesn't terminate normally, y cannot be 0
- Problem: undecidable to tell if S terminates!
- In general program analysis must compute an approximation

Quick Undecidability Proof

- Theorem: There does not exist a program Q that can decide for all programs P, whether P terminates.
- Proof: By contradiction.

- Assume there exists a program Q(x) that returns true if x terminates, false if it does not.

- Consider the program "R = if Q(R) then loop."
- If R terminates, then Q returns true and R loops (does not terminate).
- If R does not terminate, then Q returns false and R terminates.
- Thus we have a contradiction, and termination must be undecidable

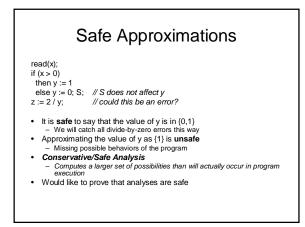
Safe Approximations

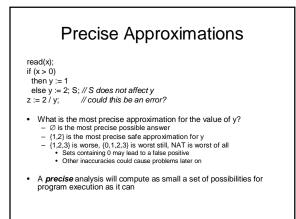
read(x); if (x > 0)then y := 1 else y := 0; S; // S does not affect y

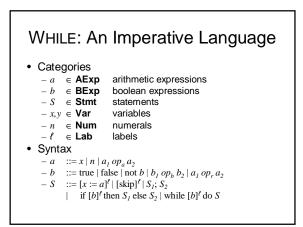
z := 2 / y; // could this be an error?

- What is a safe approximation for the value of y?
 - {1}? {0}? no no

 - {0,1}? yes {0,1,43}? yes NAT? ves
- · Intuition: we want to ensure we find all divide by zero errors







Example WHILE Program

 $\begin{array}{l} [y:=x]^1;\\ [z:=1]^2;\\ \text{while } [y>1]^3 \ \text{do}\\ [z:=z^*y]^4;\\ [y:=y-1]^5;\\ [y:=0]^6; \end{array}$

Computes the factorial function, with the input in \boldsymbol{x} and the output in \boldsymbol{z}

Reaching Definitions Analysis

 A variable definition of the form [x := a]^t may reach program point P if there is an execution of the program where x was last assigned a value at t when P is reached.

Uses

Optimization

- Does a constant assignment reach a variable's use?
 Bug finding
 - Does a NULL assignment reach a pointer dereference?
 - Does a 0 assignment reach a divisor?

Reaching Definitions Example

	ite at only			116	ITD UT OAT		
	x	у	z	<u>x</u>	У	z	
[y := x]¹;	?	?	?	?	1	?	
[z := 1] ² ;	?	1	?	?	1	2	
while [y>1] ³ do	?	1,5	2,4	?	1,5	2,4	
[z := z * y] ⁴ ;	?	1,5	2,4	?	1,5	4	
[y := y − 1] ⁵ ;	?	1,5	4	?	5	4	
[y := 0] ⁶ ;	?	1,5	2,4	?	6	2,4	