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
read(x);
if (x > 0)
    then y := 1
else y := 0; // S does not affect y
z := 2 / y; // could this be an error?

• What is a safe approximation for the value of y?
  – {1}? no
  – {0}? no
  – {0,1}? yes
  – {0,1,43}? yes
  – NAT? yes

• Intuition: we want to ensure we find all divide by zero errors
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?

• It is **safe** to say that the value of y is in \{0,1\}
  – We will catch all divide-by-zero errors this way
• Approximating the value of y as \{1\} is **unsafe**
  – Missing possible behaviors of the program
• **Conservative/Safe Analysis**
  – Computes a larger set of possibilities than will actually occur in program execution
• Would like to prove that analyses are safe
Precise Approximations

read(x);
if (x > 0)
    then y := 1
else y := 2; // S does not affect y
z := 2 / y; // could this be an error?

• What is the most precise approximation for the value of y?
  – $\emptyset$ is the most precise possible answer
  – {1,2} is the most precise safe approximation for y
  – {1,2,3} is worse, {0,1,2,3} is worst still, NAT is worst of all
    • Sets containing 0 may lead to a false positive
    • Other inaccuracies could cause problems later on

• A **precise** analysis will compute as small a set of possibilities for program execution as it can
**WHILE: An Imperative Language**

- **Categories**
  - \( a \in \text{AExp} \) arithmetic expressions
  - \( b \in \text{BExp} \) boolean expressions
  - \( S \in \text{Stmt} \) statements
  - \( x, y \in \text{Var} \) variables
  - \( n \in \text{Num} \) numerals
  - \( \ell \in \text{Lab} \) labels

- **Syntax**
  - \( a ::= x \mid n \mid a_1 \ op_a \ a_2 \)
  - \( b ::= \text{true} \mid \text{false} \mid \text{not} \ b \mid b_1 \ op_b \ b_2 \mid a_1 \ op_r \ a_2 \)
  - \( S ::= [x := a]^\ell \mid [\text{skip}]^\ell \mid S_1; \ S_2 \)
  - \( \mid \text{if} \ [b]^\ell \text{ then } S_1 \text{ else } S_2 \mid \text{while} \ [b]^\ell \text{ do } S \)
Example WHILE Program

\[ y := x \]
\[ z := 1 \]
while \[ y > 1 \] do
\[ z := z \times y \]
\[ y := y - 1 \]
\[ y := 0 \]

Computes the factorial function, with the input in \( x \) and the output in \( z \)
Reaching Definitions Analysis

• A variable definition of the form \([x := a]^l\) \textit{may reach} program point \(P\) if there is an execution of the program where \(x\) was last assigned a value at \(l\) 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

\[
\begin{array}{llll}
\text{RD at entry} & x & y & z \\
\text{RD at exit} & x & y & z \\
\hline
[y := x]^1; & ? & ? & ? \\
[z := 1]^2; & ? & 1 & ? \\
\text{while } [y>1]^3 \text{ do} & ? & 1,5 & 2,4 \\
\quad [z := z \times y]^4; & ? & 1,5 & 2,4 \\
\quad [y := y - 1]^5; & ? & 1,5 & 4 \\
[y := 0]^6; & ? & 1,5 & 2,4 \\
\end{array}
\]