Static Analysis

ThanhVu Nguyen
CSCE467

October 31, 2019
“Program testing can be used to show the presence of bugs, but never to show their absence.” (DJK, 1972)
What is Static Analysis?

Static Analysis
A method for **automated reasoning** on a **representation** of program

- **Static**: apply to some static representation (e.g., source code) of a program (in contrast to testing, profiling, or run-time checking)
- **Automated**: “push-button” technology, i.e., little user intervention

Applications
- **Compilers**: optimization (runtime, memory), remove dead code, etc
- **Verification**: verify program **correctness**
The Dream

Static Analyzer

- **Inputs**: program, specifications (pre/post conditions, assertions)
- **Output**: correct/safe (provable), incorrect/unsafe (witness)

Requirements for a Perfect Analyzer

- **Soundness**: don’t miss errors (no false negative)
- **Completeness**: don’t raise false alarms (no false positive)
- **Termination**: always terminate

Question: is testing sound, complete, or terminate?
False and True Positives

Type I error
(false positive)

You’re pregnant

Type II error
(false negative)

You’re not pregnant
The Issue

Decision Problems

- Is the program $P$ free of null ptr error?
- Does the program $P$ satisfy given some given specification $S$?
- Does the program $P$ terminate?

Rice Theorem (1953)

All non-trivial semantic questions about programs from a universal programming language are undecidable.
Approximation / Abstraction

- Example: \( x = 42 \subseteq x \geq 40 \subseteq x \geq 0 \subseteq x \in \mathbb{Z} \)
- Approximate allows decidability and efficiency
- The approximation must still be *sound*, (often) sacrifice *completeness*, should preserve *termination*
- Properties:
  - **Precision**: must still be precise enough to give some *useful* answer
  - **Efficiency**: time/space usage
  - **Scalability**: work with realistic, real world programs
## The WHILE language

<table>
<thead>
<tr>
<th>Category</th>
<th>Domain</th>
<th>Meta variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numbers</td>
<td>$Z = {0, 1, -1, \ldots }$</td>
<td>$z$</td>
</tr>
<tr>
<td>Truth values</td>
<td>$B = {T, F}$</td>
<td>$t$</td>
</tr>
<tr>
<td>Variables</td>
<td>$Var = {x, y, \ldots}$</td>
<td>$x$</td>
</tr>
<tr>
<td>Arithmetic expressions</td>
<td>$AExp$</td>
<td>$a$</td>
</tr>
<tr>
<td>Boolean expressions</td>
<td>$BExp$</td>
<td>$b$</td>
</tr>
<tr>
<td>Commands (statements)</td>
<td>$Cmd$</td>
<td>$c$</td>
</tr>
</tbody>
</table>

### Context-Free Grammar of WHILE

- $a ::= z \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \ast a_2 \in AExp$
- $b ::= t \mid a_1 = a_2 \mid a_1 > a_2 \mid \neg b \mid b_1 \land b_2 \mid b_1 \lor b_2 \in BExp$
- $c ::= \text{skip} \mid x := a \mid$
  - if $b$ then $c_1$ else $c_2$ end |
  - while $b$ do $c$ end $\in Cmd$ |
  - $c_1; c_2$
Example of a WHILE program

x := 6;
y := 7;
z := 0;
while x > 0 do
  x := x - 1;
  v := y;
  while v > 0 do
    v := v - 1;
    z := z + 1;
  end
end