Software Tools
Static Analysis

Part II - Lecture 8
Today’s Outline

- Introduction to Static Analysis
- Static Analysis with JLint
- Detecting Null Pointers
Introduction to Static Analysis

“Use the source, Luke.”
Static Analysis

• Analyzing programs by looking at their code (i.e. before running them)

• Sad result from theory: In general, many analysis problems are undecidable, e.g. Turing’s halting problem

• However:
  - Many important cases that occur in practice can be analyzed and errors detected
  - For most cases analysis can be approximated, i.e. we can give warnings if something is likely to be wrong

• Static analysis is usually done on the AST
False Positives and False Negatives

**False Positive:**
- The analysis tool gives a warning but there is no error
- More work for the developers (distinguishing true positives from the false positives)

**False Negative:**
- There is an error but the analysis tool does not give a warning
- Errors go undetected

*Conservative* analysis means no false negatives are produced (i.e. no errors are missed)
Control Flow and Data Flow Analysis

Control Flow Analysis
Looking at the different paths of execution in a program
E.g. the red arrows

Data Flow Analysis
Looking at the possible values that occur at certain points in a program
E.g. the blue arrows

Pseudo-Evaluation
Analyzing a program by simulating its execution (with simplified execution semantics, e.g. only one iteration is executed per loop)
Example: Coverity Prevent

- Commercial tool for static analysis through pseudo evaluation, e.g. to detect:
  - Buffer overrun: trying to write over end of array
  - Memory leaks: allocating but never freeing memory
  - Use after free: freeing memory and then accessing
  - Uninitialized variables: using variable before init
  - Dead code/data: code or data never used

- Uses mostly heuristics, not precise analysis rules
- Produces false positives and false negatives
- Has helped many open-source projects to fix numerous bugs
JLint
JLint

- JLint is a simple static checker for Java
- It works directly on the compiled classes
  - Does not require the source code
  - Does not require human specification
  - Very easy to use, but limited capabilities
- JLint can give warnings for some concurrency, data flow and code clarity problems
- Some warnings might be false alarms
- Call JLint from the command line with a class file e.g. jlint MyClass.class
- Call JLint without parameters to get help information
Data Flow Problems

- **Null-pointers**
  - A method is possibly invoked with null as parameter but the method does not check for null argument
  - Value of dereferenced variable may be null

- **Value range**
  - Range of assigned expression value has no intersection with target type range
  - Possible overflow, e.g. `int z = (int)x * (int)y;`

- **Redundancy**
  - Comparison always produces the same result e.g. `1+1==3` will always be false
Unclear Code Problems

- Checked with a separate tool called AntiC
- Unclear operator precedence, e.g. \( x || y == z \)
- \( = \) and \( == \) possibly confused, e.g. if \( (x = y) \) {} 
- Unclear nested block structure
  ```java
  while (x != 0)
      x >>= 1;
      n += 1;
  return x;
  ```
- Unclear else-association, e.g. if \( (x) \) if \( (y) \) i++; else j++;
- Method is overridden by method with the same name but different parameters
- Field in class shadows field of superclass
- Local variable name shadows field of class
Detecting Null Pointers
Null Pointers

- Null pointers are one of the main causes for runtime errors
- In Java, if a null reference is dereferenced then a NullPointerException is thrown
  - Method call: x.m() and x==null
  - Field access: x.y and x==null
- Often problems in code that lead to null pointer errors are quite simple, e.g.
  - Forgot to initialize variable properly
  - Forgot an if-statement checking for a special case
- Can we detect potential null pointer errors?
Detecting Null Pointers

• Define analysis functions:
  - MayReturnNull: Method → \{true, false\}
  - VarMaybeNull: Variable → \{true, false\}
  - ExprMaybeNull: Expression → \{true, false\}

• For every method \( M \): MayReturnNull(\( M \)) tells us if \( M \) may return null

• At every point in the program, for every variable \( X \) defined at that point: VarMaybeNull(\( X \)) tells us if \( X \) may be null (depends on program state)

• For every expression \( E \) defined in a program: ExprMaybeNull(\( E \)) tells us if \( E \) may be null (depends on VarMaybeNull and MayReturnNull)

• If an expression that may be null is dereferenced, then generate a warning
VarMaybeNull Example

We calculate MaybeNull at all positions in a program:

```java
void m(String t, int n) {  // VarMaybeNull(t)==true
    String s = null;  // VarMaybeNull(s)==true
    if (n / 10 + 1 > 100)
        s = "hello1";  // VarMaybeNull(s)==false
    else
        s = t;  // VarMaybeNull(s)==true
    // VarMaybeNull(s)==true
    s.substring(1);  // Warning!!!
}
```

- If we don’t know much about \( x \): \( \text{VarMaybeNull}(x) \) is true
- After assigning non-null value to \( x \): \( \text{VarMaybeNull}(x) \) is false
- \( \text{VarMaybeNull}(x)==\text{true} \) after an if-statement, if \( \text{VarMaybeNull}(x)==\text{true} \) after the if- or after the else-part
- If we dereference variable \( x \) and \( \text{VarMaybeNull}(x)==\text{true} \) then give warning ("NullPointerException may happen")
Defining ExprMaybeBeNull

Defined on a simple Java subset using MayReturnNull and VarMaybeBeNull:

- **Constant expressions:** `c`
  
  \[ \text{ExprMaybeBeNull}(c) = (c == "null") \]
  
  If a constant is null, then true, otherwise false

- **Method calls:** `m(\ldots)`
  
  \[ \text{ExprMaybeBeNull}(m(\ldots)) = \text{MayReturnNull}(m) \]
  
  If `m` may return null, then the expression may be null

- **Variable access:** `x`
  
  \[ \text{ExprMaybeBeNull}(x) = \text{VarMaybeBeNull}(x) \]
  
  The expression may be null if the variable may be null

- **Most other expressions can never be null,** e.g. `x + y`
Defining VarMaybeNull

Go through the statements one by one:

- Most statements do not affect VarMaybeNull(x), e.g. statements where variable x is not involved
- Assignment: \( x = \text{expr} \);
  \[
  \text{VarMaybeNull}(x) = \text{ExprMaybeNull}(\text{expr})
  \]
  If the expression may be null, then the var may be null
- If: \( \text{if}(\ldots) \text{s1}; \text{else} \text{s2}; \)
  \[
  \text{VarMaybeNull}(x) = \text{VarMaybeNull}(x) \text{ after s1};
  \]
  \[
  \text{VarMaybeNull}(x) \text{ after s2};
  \]
- For loop: \( \text{for}(\ldots) \text{s1}; \)
  \[
  \text{VarMaybeNull}(x) = \text{VarMaybeNull}(x) \text{ after s1};
  \]
  \[
  \text{VarMaybeNull}(x) \text{ before for}
  \]
  If the loop is executed, then look at the loop body;
  if it is not executed then VarMaybeNull is unchanged
Defining MayReturnNull

Look at all return statements “return e;” in method m:

- If for all these statements ExprMaybeNull(e)==false, then MayReturnNull(m)==false
- If for least one statement ExprMaybeNull(e)==true, then MayReturnNull(m)==true
- If we do not know enough about a method m (e.g. we don’t have the source code) then let’s be careful and say MayReturnNull(m)==true
- Similar with VarMaybeNull for method parameters: we do often not know what actual parameters a method gets, so we say VarMaybeNull(parameter)==true
Null Pointer Detection Example

```java
String m1(int n) {
    String s = null;    // VarMaybeNull(s)==true
    for (int i=0; i<n; i++)
        s = "hello"+i;    // VarMaybeNull(s)==false
    // VarMaybeNull(s)==true
    return s;          // MayReturnValue(m1)==true
}

int m2(String x) {    // VarMaybeNull(x)==true
    String y = "foo";    // VarMaybeNull(y)==false
    if (x.equals("hello"))
        y = "hello";    // VarMaybeNull(y)==false
    else
        y = x;            // VarMaybeNull(y)==true
    // VarMaybeNull(y)==true
    return y.length;    // Warning: y may be null !!!
}
```
Summary
Today’s Summary

- **Static Analysis:**
  Analyzing programs by looking at their code
- Tools can do it automatically, e.g. finding resource leaks, buffer overruns, dead code
- Many analysis problems are undecidable; heuristics are used that produce false positives and false negatives
- Null pointer detection can be done by defining functions on variables, expressions and methods

References:

- **Security Report. Static Analysis Tools.**
- **Peter Schachte. A Gentle Introduction to Static Analysis.**
Quiz

1. What are false positives and false negatives?
2. What are controls flow and data flow analysis?
3. Take a small Java program and try to do the null pointer detection on it.