Lecture 13 — Static Analysis

David J. Pearce

School of Engineering and Computer Science
Victoria University of Wellington

What is Static Analysis?

*"**Static analysis**, also called static code analysis, is a method of computer program debugging that is done by examining the code without executing the program."*

–Margaret Rouse

*"**Static code analysis** is the process of detecting errors and defects in software's source code. Static analysis can be viewed as an automated code review process."*

–viva64.com

*"**Static program analysis** is the analysis of computer software that is performed without actually executing programs (analysis performed on executing programs is known as dynamic analysis)."*

–Wikipedia

Facebook **Infer**

"Facebook Infer is a **static analyzer** *that runs incrementally on code submitted internally for review by their teams, finding bugs before the code is committed to the codebase or deployed on people's devices.*
Infer currently signals Null pointer accesses, resource and memory leaks, it runs on C, Java and Objective-C code, and has been written in OCaml. Facebook uses Infer to automatically verify the code of their mobile applications for iOS and Android, correctly reporting on bugs in 80% of the cases."

–InfoQ

- `http://www.infoq.com/news/2015/06/facebook-infer`
- `http://fbinfer.com/`

PVS **Studio**

The PVS-Studio product contains a set of general-purpose **static analysis** *rules intended to detect a wide range of various defects in C/C++/C++11 applications.*

–viva64.com

- Finds common errors, including:
  - Copy-Paste errors and typos
  - Array index of bound, buffer overrun
  - Undefined/unspecified behavior
  - Arithmetic over/underflow, check for integer division by zero.
  - Null pointer dereferences

- Can find errors in e.g. **Linux Kernel**:
  - **See**: “PVS-Studio Probes into Linux’ Innards”

Other Examples

"CodeSonar, GrammaTech's flagship **static analysis** *software, identifies programming bugs that can result in system crashes, memory corruption, leaks, data races, and security vulnerabilities."*

–grammatech.com

"PMD is a **source code analyzer***. It finds common programming flaws like unused variables, empty catch blocks, unnecessary object creation, and so forth. It supports Java, JavaScript, PLSQL, Apache Velocity, XML, XSL."*

–pmd.sourceforge.net

Example: Definite Assignment

```
int abs(int x) {
  int r;
  if(x < 0) {
    r = -x;
  }
  return r;
}
```

- Above function **does not compile** in Java
  - But, it **does compile** in C or C++!

- Java uses "**Definite Assignment**" Analysis
  - This is a simple form of **static analysis**

Quiz: Does this Compile?

```
int looper(int n) {
  int i = n+1;
  int r;
  while(i != n) {
    i = i + 1;
    if(i == n) { i = i + 1; }
  }
  return r;
}
```

Or, put another way, can we ever reach the ***`return`****?*

Example: Definite Assignment (cont'd)

"Each local variable (§14.4) and every blank final (§4.12.4) field (§8.3.1.2) must have a definitely assigned value when any access of its value occurs. An access to its value consists of the simple name of the variable occurring anywhere in an expression except as the left-hand operand of the simple assignment operator =. A Java compiler must carry out **a specific conservative flow analysis** to make sure that, for every access of a local variable or blank final field f, f is **definitely assigned** before the access; otherwise a compile-time error must occur."

–Java Language Specification §16:

Quiz: What about this?

```
int abs(int x) {
  int r;
  if(x < 0) { r = -x; }
  if(x >= 0) { r = x; }
  return r;
}
```

Does this compile?

Quiz: And this?

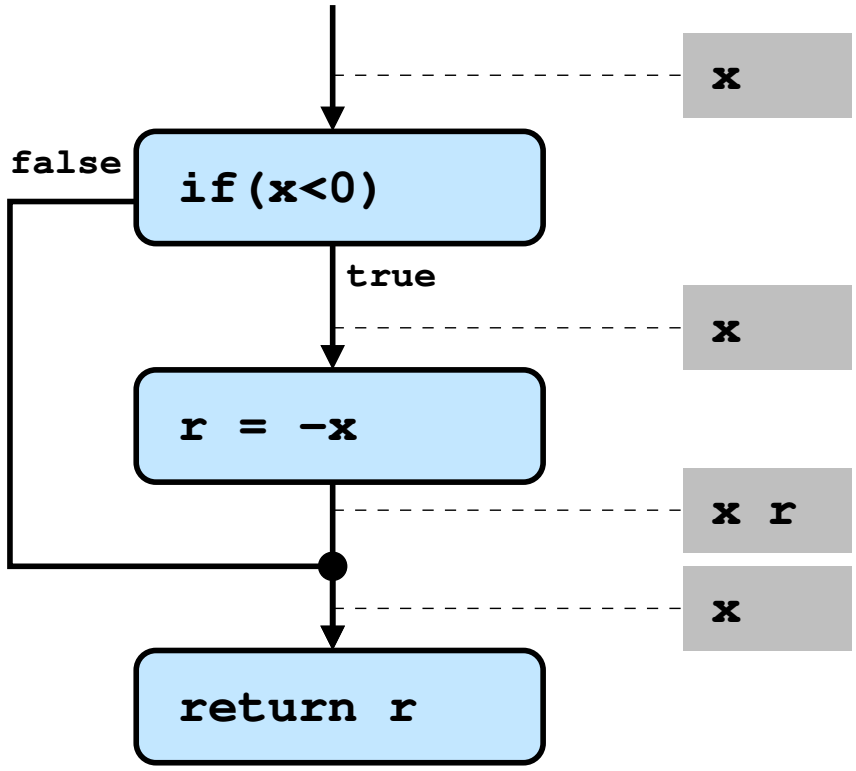
```
int thrower() {
  throw new RuntimeException();
}

int f(int x) {
  int r;
  if(x < 0) { thrower(); }
  else { r = x; }
  return r;
}
```

Does this compile?

How does it work?

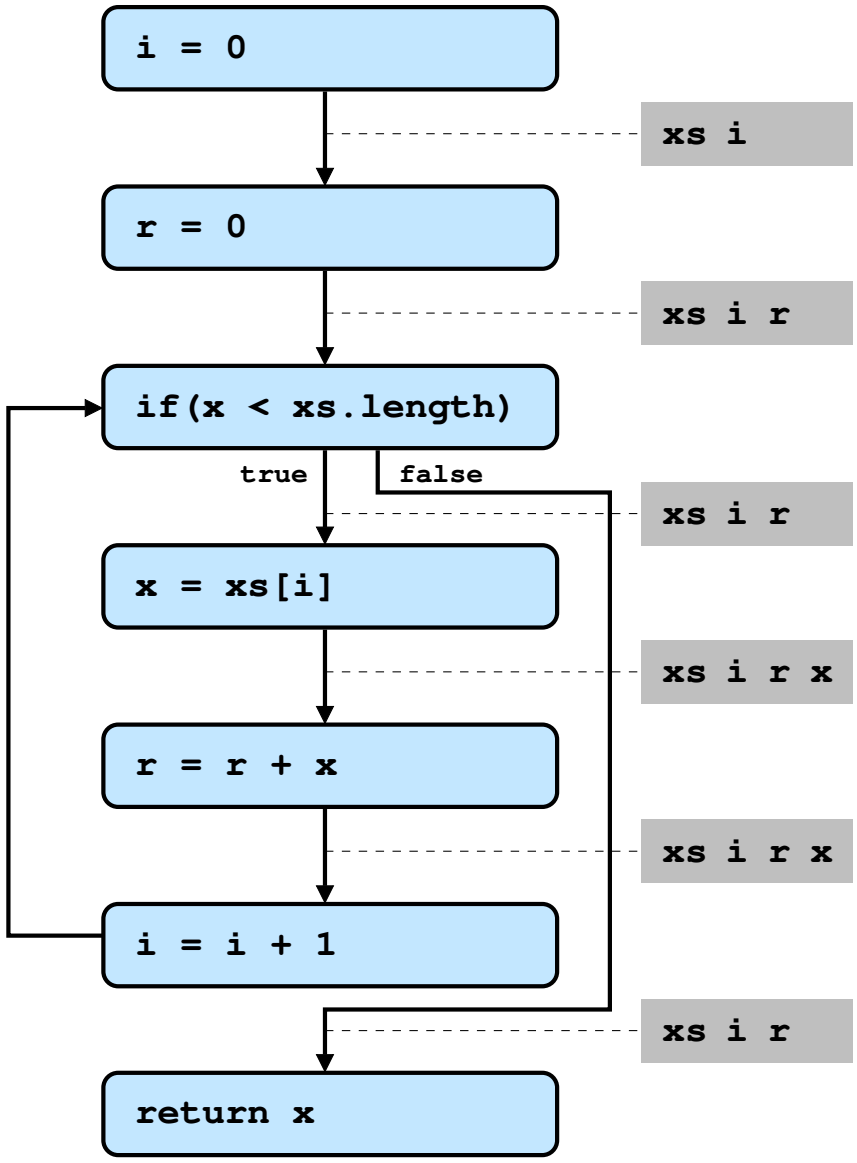
```
int abs(int x) {
  int r;
  if(x < 0) {
    r = -x;
  }
  return r;
}
```



- **Algorithm**:
  - Generate **Control Flow Graph** (CFG)
  - Perform **Traversal** of CFG
  - Maintain **set of assigned variables** at each point
  - **Intersect** assigned variables at join points

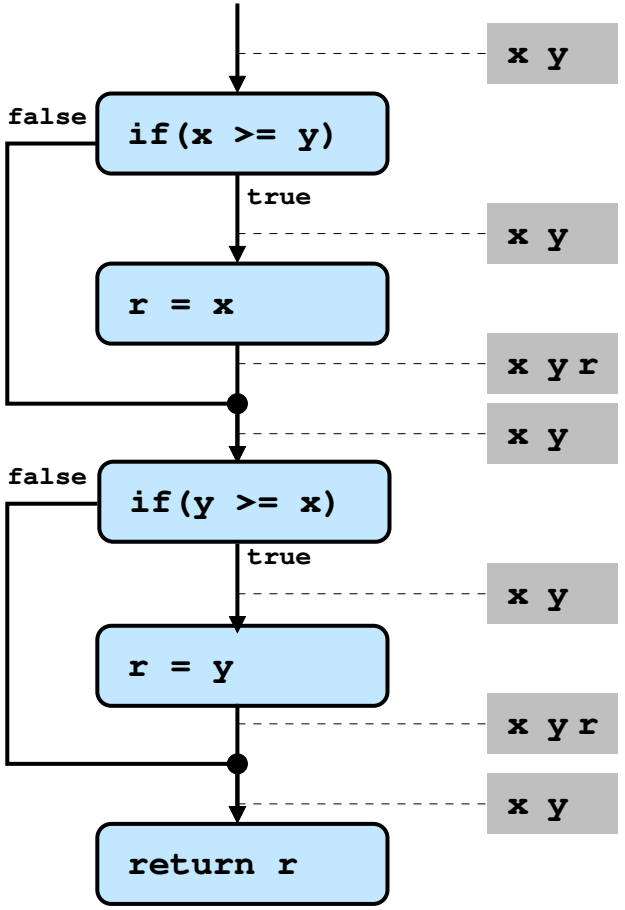
How does it work? (loops)

```
int sum(int[] xs) {
  int i = 0;
  int r = 0;
  int x;
  while(i < xs.length) {
    x = xs[i];
    r = r + x;
    i = i + 1;
  }
  return x;
}
```



Conservatism

```
int max(int x, int y) {
   int r;
   if(x >= y) {
      r = x;
   }
   if(y >= x) {
      r = y;
   }
   return r;
}
```



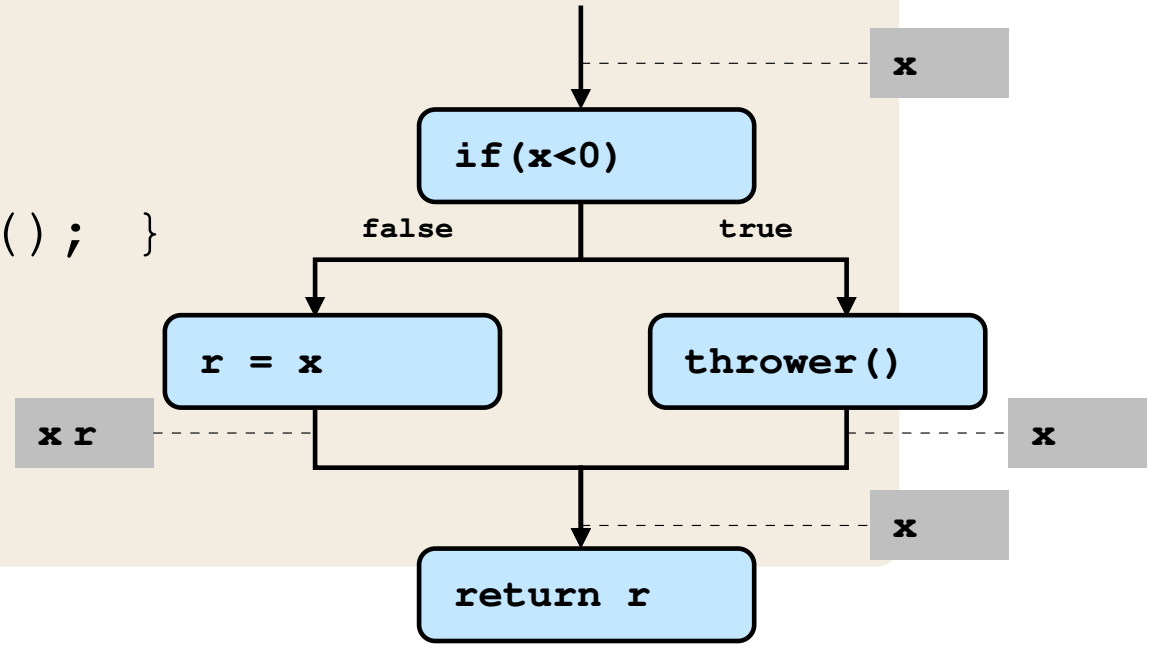
- Definite assignment is **conservative** meaning ...

  ... **infeasible paths** are ignored

Conservatism

```
int thrower() { throw new RuntimeException(); }

int f(int x) {
  int r;
  if(x < 0) { thrower(); }
  else { r = x; }
  return r;
}
```



- ... **method effects** are also ignored

Further Reading

- *"What is Static Analysis and What is it Good For?"*

  `http://www.cloudcomputing-news.net/news/2011/oct/05/what-is-static-analysis-and-what-is-it-good-for/`

- *"What is Static Analysis?"*

  `http://stackoverflow.com/questions/49716/what-is-static-code-analysis`

- *"What is static program analysis"*, Matt Might

  `http://matt.might.net/articles/intro-static-analysis/`